Hewlett-Packard
Business Users Conference

I·N·T·E·R·E·X

ORLANDO

PROCEEDINGS

August 7-12
1988

Volume 1

INTEREX

the International Association of
Hewlett-Packard Computer Users

Proceedings

of the
1988 Conference of
HP Business Computer Users

at
Orlando, Florida
August 7-12, 1988

Volume 1

William M. Lyons, Editor

INDEX BY PAPER NUMBER

INDEX BY PAPER NUMBER

INDEX BY PAPER NUMBER

INDEX BY PAPER NUMBER

INDEX BY PAPER NUMBER

INDEX BY PAPER NUMBER

INDEX BY PAPER NUMBER

INDEX BY PAPER NUMBER

INDEX BY PAPER NUMBER

INDEX BY AUTHOR

INDEX BY AUTHOR

INDEX BY AUTHOR

INDEX BY AUTHOR

INDEX BY TITLE

INDEX BY TITLE

INDEX BY TITLE

INDEX BY TITLE

PC Integration/Office Automation

Systems Manager/Data Communications

INDEX BY CATEGORY

Application Development

INDEX BY CATEGORY

Management Track

INDEX BY CATEGORY

The HP Employment Market - What Every Hiring Manager Should Know

Lynn A. Novo
Network Systems Company
98 South Turnpike Road
Wallingford, CT 06492

In the six years that I have been working as a personnel consultant exclusively for companies that have Hewlett Packard 3000 computers, I have seen that the managers who are most successful at attracting and hiring HP professionals are sensitive to the idiosyncracies of the HP employment market. What follows are some of their best ideas for identifying, interviewing, and hiring in this unique employment market. You will get a complete hiring process that will help you feel confident that once you do find someone who can fill your job requirements, you will have done everything possible to get an acceptance of your offer.

We will begin with an overview of the HP employment market and then we will concentrate on specific guidelines to utilize from the beginning to the end of your hiring process. We will discuss how to define your staffing need, how to identify candidates to interview, and how to conduct the interview itself. We will conclude with how to extend an offer in a way that it will most likely be accepted.

THE HP EMPLOYMENT MARKET

Let's begin our overview of the HP employment market by identifying it. Just who is it that looks for a new job? What are the reasons people give for leaving their current employer?

I am sure you have noticed that in some respects, every person who works in data processing is "in the market". We all have heard those words: "I'm always interested in hearing about new opportunities". I hope for your sake you don't frequently hear your programmers say those words into the telephone! But let's separate out the serious job changers from the "always looking" ones and see why they are in the market.

Whenever we at Network Systems Company talk with people who are seriously interested in changing jobs, we ask them to tell us the reason they would consider leaving the company they work for at the time. Remember that we only talk with people who work in HP 3000 environments. We log this information in our computer and find that all the reasons we have heard over the years fall into the main reasons shown below.

REASONS FOR CHANGING JOBS

Want more money, salary too low

Candidate relocating

Company or department moving

No career growth, want more responsibility

Company converting to another vendor system

Want a shorter commute

Position boring, want more diversity and projects

Management problems, unclear direction

Losing job due to termination or lay off

For the purposes of this illustration, we are grouping together the reasons sited by ALL levels and types of HP personnel. Please keep in mind that the bulk of the HP employment market consists of applications personnel such as programmers or programmer/analysts. These are the people who most often change jobs and for whom there is the greatest demand.

Which do you think is the reason we hear most often?

Surprisingly, it's not because of low salary or low raises. In fact, that is one of the reasons we hear LEAST often.

Most people who consider themselves "in the market" say the reason is:

" No career growth, want more responsibility "

Managers, please take note. Although this discussion is not about how to keep your current staff, retaining your staff is a challenge just as big as hiring is. I welcome you to use this list of why HP professionals change jobs as a way to begin a dialog with your staff on this important subject. If you can improve your relationship with your staff, you may never need to use these guidelines on how to hire!

As we look at the actual hiring process, it will be helpful to keep in mind the following keys for successful hiring in the HP market.

FLEXIBILITY * TIMELINESS * PREPARATION

We'll see how each of these plays an important role in our success at implementing a hiring process that will result in more "yesses" to our offers.

THE HIRING PROCESS

What are the essential components of the hiring process?

Identifying the need
Sourcing candidates
Interviewing to hire
Extending the offer

IDENTIFYING THE NEED

The successful hiring managers we talk to always begin their hiring process with a well thought out job description. They do not necessarily try to replace someone who leaves with the same set of skills. They take the opportunity to evaluate their current staff and ascertain where there might be in-house skills to fill the gap.

Remember our reasons for changing jobs? This is a good time to make sure that anyone who can be promoted or reassigned to a project gets the opportunity to do so. You'll eliminate a lot of resentment and a possible second resignation if you consider your current staff first, before you hire from the outside.

Take the time to write a job description. We find that one of the biggest complaints we get from candidates is that they are not told what they actually will be doing on a day to day basis.

If we remember that our keys to success in HP hiring, we can see that this first stage in the process requires a great deal of PREPARATION. Your time investment at this stage will be rewarded with successful hires after the interview.

Although I suggest that you have specific requirements for the job, I also recommend FLEXIBILILTY. In this highly competitive HP employment market, you must be willing to accept a combination of skills and total experience, not just specific requirements. Managers who fill their vacancies fastest are willing to interview candidates with backgrounds that come close to their needs. They are in the habit of screening in potential candidates, not screening them out because they do not match the rigid requirements.

SOURCING CANDIDATES

In this competitive market, as manager, you need to be creative in your approach to identify sources of candidates. It will not suffice to merely place an ad in the newspaper, or wait for your personnel department to forward you resumes. The successful hiring managers of the 1990's will be using a variety of sources and will network with as many people as possible on an ongoing basis to be certain that they are always tuned into the market.

Let's brainstorm the ways you can identify people to interview to fill your job openings. These are the ways hiring managers have told me they have sourced candidates:

Newspaper advertising
Radio or cable TV advertising
Job fairs
Employment agencies/recruiters
College placement office or computer science department
User group meetings/conventions
Personal contacts
Vendor referrals
Employee referrals (give bonus)

Our clients tell us that one of the consistently UNSUCCESSFUL ways to identify candidates is through newspaper advertising. I am sure many of you have had successful response to an ad at some point, but our clients tell us they cannot rely on the response. In this unique HP market, there is not a large enough pool of qualified readers to justify the expense in most newspaper geographic markets. The person you need to hire probably is not bothering to read the Sunday want ads anyway.

The college campus is an untapped resource in some areas. If you know that a college in your area is using the HP to teach computer science students, you should get to know the department head and placement office. Many of our clients take college students as interns and then hire them to work in full time positions after graduation.

In sourcing candidates to interview, it helps to be willing to try different methods. Timeliness and preparation count when it comes to the all-important networking at user group meetings, with your vendors, or with recruiters. You can't begin to make contacts in the marketplace <u>after</u> your lead programmer resigns. You need to have known what's out there and what sources you can tap before you actually need them.

Flexibility and timeliness are not attributes typically used to describe personnel departments. If your personnel department's idea of recruiting is only to put an ad in the paper, screen out resumes based on your ideal job requirements, then forward a resume to you two weeks after it was sent in, chances are the candidate has already started working at the HP shop down the road.

The managers who view themselves as successful in their HP hiring either have extraordinarily efficient and cooperative personnel departments who understand this unique market, or they are handling the hiring process almost entirely without personnel.

INTERVIEWING TO HIRE

When you have identified a gem of a candidate for your staff opening, it is time to prepare for the interview. There are some very simple things you can do to make sure that your gem doesn't get so aggravated with the interview process itself that he is turned off to your company. If enough of the following "little things" go wrong, your gem will not want to work for you regardless of how much you want him or offer him.

Have the personnel interview occur AFTER your department's interview, if you must have a personnel interview at all. Your gem doesn't care that his contribution to the medical plan is $14.10 a month. He won't be listening attentively enough to remember after the interview anyway. He wants to find out about you and the technial environment as quickly as possible.

If by chance you cannot persuade your personnel department to agree to this sequence, then at the very least make sure that they see your gem promptly and that they complete it as quickly as possible.

Don't bother with an application form. If an employment application must be completed, either let your gem take it and return it by mail or have him complete it after your interview. No use having your gem getting frustrated before your interview trying to remember if he started that job in February or March of 1980.

Don't keep your candidate waiting. Your gem needs to feel that your interview with him is important. When candidates must wait to begin their interview, that is the first thing they mention to us in the follow-up. "They made me wait 20 minutes". It's hard to dispel that resentment.

Try to be uninterrupted during the interview. Even if you don't mind if your gem hears your users screaming, he may get the impression that you don't value his time or have enough control of your time to be left undisturbed.

Spending time in preparation for the interview and following these pointers will help your gem be predisposed to accepting an offer.

Once you get your gem into your office (and it should be a private office, even if it isn't your own because yours is too messy) the most successful interviews have been well prepared to include the following four phases:

Rapport screening

Fact finding

Evaluation interview

Selling

RAPPORT SCREENING

In rapport screening, you set the stage for a productive interview. You want your gem to feel comfortable so that he will be open and honest with you. You should ask friendly, conversational questions about his hobbies and interests. These first couple of minutes are when your gem is going to be getting to know you as well. This is the time for establishing shared experiences. Often we hear things like, "We hit it off right from the start. It turns out that the manager likes to ski as much as I do."

FACT FINDING

After about 10 minutes of rapport screening, you begin to establish if your gem has the credentials to do the job. You will want to ask about what he does in his current position and probe into specific areas of his background to see how they relate to your needs.

As part of the fact finding session, many managers involve another staff member to ask technical questions. This is quite effective providing the other person doesn't duplicate the questions you have asked.

EVALUATION INTERVIEW

If there is one secret to successful hiring, this is it. I guarantee that if you implement this phase of interviewing and ask these three simple questions at this point in the interview, you will be rewarded with some very valuable information. You will be better able to determine if your gem should be made an offer; you will be able to use this information to extend an offer with a greater likelihood of acceptance; and you will be better able to retain your gem once he does come to work for you.

The evaluation interview is your time to get inside your gem's head. It will reveal more about his personality and work ethic than any other type of questions you can ask.

Because of the personal nature of these questions and the fact that you want your gem to be open and honest, it helps to take a break (get a cup of coffee, for example) to make the transition to this phase. I also recommend that you do not take notes during this phase, but rather write notes after the candidate leaves.

For the purposes of this paper, I will present the interpretation of some of the responses along with these questions. During the interview itself, however, I suggest that you keep your focus on the answers your gem is giving and reserve your interpretation of responses until after the interview.

EVALUATION INTERVIEW QUESTION 1

Let's begin the evaluation questions with a topic we have discussed already. Ask:

"Why would you consider leaving your current employer?"

We have already seen the range of possible answers to this question. Let's take a closer look at each of the reasons beginning with these tangible reasons when people are forced to find a new job:

Candidate Relocating
Company or department moving
Losing job due to termination or lay off

Most managers feel comfortable with any of these reasons and in fact find that candidates with these reasons are the most eager to accept offers. However, if your gem has lost or will lose his job, be sure to get the details of the termination. Good HP people are not usually let go, be certain your gem is the exception.

Continuing with other reasons for leaving:

Want a shorter commute

If your company is significantly closer to your candidate's home or if it will be an easier commute, you have one of the best motivators for changing jobs. In fact, even people who would not ordinarily listen to other opportunities become receptive to an opening that is closer to home. Of course we need not mention the benefits to you of having your staff members living close by.

Company converting to another system

If your gem is upset that his company is replacing their HP system with another vendor's system, this should be music to your ears. It demonstrates that your candidate knows and loves the HP and would rather switch companies than fight the new vendor. Incidentally, this kind of "vendor loyalty" is very common in the HP marketplace despite the popular notion that everyone would rather work on IBM.

Want more money, salary too low

This reason makes most managers nervous and indeed it should. The candidate who says money is his primary motivator for changing jobs is probably not the one you want to hire. If you hire this type, you will always wonder if you are giving him high enough raises (and our research shows that these people NEVER think they are being paid enough).

The reasons you will hear most often are the intangible ones that remain:

Position boring, want more diversity and projects
No career growth, want more responsibility
Management problems, unclear direction

The savvy hiring manager hears out the candidate's complaints in any of these areas. These refer to the way your candidate perceives his work environmnent. With these emotional issues, you can see that this information will help you shape your subsequent comments to your candidate in terms that will "feel" positive and that he will view as an improvement over his current situation. I'll show you exactly how when we get to the later phases of the interview process.

EVALUATION INTERVIEW QUESTION 2

"What do you want out of your work?"

This is where you will hear about your gem's plans for his future. Can he verbalize some long term goals? Does he know what he wants? It will be hard for you to keep him happy if he doesn't even know what it is that he wants. If he does tell you some plans that could fit into your department's future, we willl see how you can use this information later to "sell" the job to him.

EVALUATION INTERVIEW QUESTION 3

"What effort would you be willing to put forth to achieve what you want?"

Listen for a response that relates to commitment. Your gem should have some idea of what it takes to achieve the success he seeks. Often this response will be conveyed in specific time elements such as, "I would be willing to work 12 hour days to complete a project ahead of schedule so that I could be promoted to a Project Leader position."

EVALUATION INTERVIEW SUMMARY

Those three little questions will give you a great insight into your potential hire's level of commitment, his career goals, and his emotional ties to his workplace. We will use the responses to these questions again later in the hiring process but before we move on, please allow me to make one more suggestion on the evaluation interview.

Once you have hired your gem, I suggest you refer to your notes on this phase of the interview frequently. If you are continuing to provide an environment where your employee is getting what he wants out of work and is able to achieve his goals as given in responses to Questions 2 and 3, you will improve your odds of keeping him longer. An added benefit is that as your staff realizes that you care about their career development and that you are committed to helping them achieve their goals, you will gain a loyalty that is normally unheard of in the data processing profession.

Let's move now to finishing up your interview. We have come to the fourth and final phase of the interview:

SELLING

Many hiring managers have told me that this was the hardest phase of the interview to master, but the rewards in terms of successful hires are worth it.

You begin this phase by telling your gem about your job opening, emphasizing those aspects you garnered will be important to him based on the evaluation interview. You can see why it would be foolish to discuss the details of the job before you had this valuable information.

Here's an example:

You learn from the evaluation interview that your gem wants to leave his current employer because he feels bored and doesn't feel that he's involved in any projects. Even if your opening is for a programmer to maintain your canned financial package, you should include some mention of a project. Think of something on your "To Do" list. Is there some piece of it that an eager new hire could do? Probably, so mention it in terms of being a project:

> "You will be involved in a project to track user complaints and make recommendations for ways to streamline the problem-solving process."

Your gem will hear a specific thing about your job that he can distinguish as being better than his current job. He must be able to make this distinction in order to accept your offer.

In this selling phase, you must also be prepared to open up about yourself. Your gem will not work for you unless he feels comfortable with you. This is the place to discuss your own background including how long you have worked there, why you joined the company, and what you like about the company.

You might be surprised at how important this is. When we surveyed all the candidates we placed in new HP jobs over the past six years, they sited the <u>personality and skill level of the hiring manager</u> as a major motivation for accepting the job offer. If you're not letting your gem get to know <u>you</u> in the interview, you're missing a great "selling" opportunity.

Be enthusiastic about your company. You can't assume that your gem knows anything about your company. You must be able to state at least one clear reason why your gem should go to work for your company. Your enthusiasm will be contagious. He will recognize and begin to feel a pride in being able to work for your company.

EXTENDING THE OFFER

Let's move ahead now and say that your gem really proved to be a gem in the end. If you like him and think you might want to hire him, you must move toward making the offer immediately.

Remember that we said that successful hiring in the HP market required TIMELINESS. Here's a case of that. This is not a market where you have the luxury of spending three weeks interviewing and another two weeks deciding who to hire. The HP managers who are most successful in their hiring are willing to make an offer to the first candidate they interview. You should know enough about the skills and personality you need before the interview to make a decision quickly after the interview when someone is right.

In determining the appropriate salary to offer, you need to pay as much (or as little, depending on your perspective) as necessary to get your gem to accept. In the HP market, most managers tell me they would rather offer slightly higher than the minimum acceptable. That is a wise approach when you have the attitude that you want your gem not only to work for you, but to come to work happily. It has been our experience that your gem will be more eager to start work quickly and has a better chance of staying with you longer if he doesn't feel that you were frugal in your initial offer.

If you take the responsiblility for keeping in tune with the market, you should always have an idea of what the "going rate" for a particular position is in your geographic area. There will be times, however, when you will want to hire a person that has an unusual blend of technical skills that would make him extremely beneficial to your department. In that case, you must not hesitate to pay a higher salary than you had expected. That extra couple of thousand dollars will be recovered quickly in training costs and immediate productivity in your department. Determining salaries in this market requires FLEXIBILITY.

One final point about salaries. If you or your personnel departments are using any national data processing salary surveys for a reference in determining salaries in your HP department, throw them in the garbage. Our specialized market defies the national averages. If you can't get a salary survey based on the HP market itself, you can guage salary ranges by asking any candidates you interview for their salary history, not just current salary. From that you can see what other companies have paid for different levels of experience over the past years and you can compare it with your staff salaries.

Once you have determined the salary, you need to contact your gem to give him the good news. The managers who get most of their offers accepted either extend the offer themselves or have a well-trained third party do it.

The best offers are made when the call is made to your gem at home and parts of the selling phase of the interview are reiterated. This is also the place where you want to feed back to your gem that valuable information you gathered in the evaluation interview phase (which he will have forgotten that he told you). Here's an offer that will most likely be accepted:

> "We were most impressed with the experience you have in maintaining financial applications in COBOL. We feel that we could help you develop those skills further by involving you with our manufacturing systems and special projects like the one I discussed during the interview. Over the next couple of years, we expect to expand our MIS department and hope that you would be someone who could eventually take on additional responsibilities.
>
> "Would you be interested? [By asking this now, you save yourself the embarrassment of making an offer to someone who is no longer interested.]
>
> "Great! Then it is my pleasure to offer you the job at a salary of $30,000 plus the benefits we discussed. You will be eligible for a salary increase of up to 10% in 6 months. When can we look forward to you starting?"

Do you notice how we keep "selling" through the offer? And again, how the information from the evaluation interview gets reworded and used to paint a positive picture of what our gem wants? We present the biggest offer we can and be sure to mention benefits and raises. If you can make your offer like this as the culmination of a well-prepared hiring process, your efforts will be rewarded with an enthusiastic new staff member.

We have talked about how your success as a hiring manager in this highly competitive HP employment market requires that you use use flexibility, timeliness, and preparation in your hiring process. Using the guidelines for identifying your staffing need, sourcing HP candidates, interviewing to hire with the evaluation interview, and extending the offer will help make certain that the person you want to hire will want to work for you.

Symbolic Debugging: An Introduction

Timothy D. Chase
Corporate Computer Systems, Inc.
33 West Main Street Holmdel, New Jersey

Next to a trusty compiler it's a good symbolic debugger that tops my list of important software development tools. I would fear facing a day in my chosen programming profession without symbolic debug. It never fails to amaze me, though, how few people use symbolic debuggers let alone know what they are. In this article I will introduce you to symbolic debuggers, their technology and how they can be used to help programmers do their work. In addition, I will give you a list of basic features which should be looked for when selecting a symbolic debugger. No one debugger has all of the features I'll mention, but some do have an impressive subset.

There are as many different symbolic debuggers as there are compilers on different computers. For this reason, it is virtually impossible to talk specifically about any one debugger or language in detail. Instead, I will try to discuss symbolic debugging in general terms and leave you to find out the details of the symbolic debuggers available to you. I have tried to select a cross section of actual debuggers to use for examples. Those mentioned include Codeview for the IBM PC by Microsoft, HP's Toolset and CCS' TRAX both for the HP3000 and HP's DEBUG/1000 for the HP1000 RTE-A system.

Finally, if you do any serious programming or if you have to maintain the work of other programmers and you *don't* have access to some form of symbolic debugger, you should complain to someone. Loudly.

What is a symbolic debugger?

Before answering that, we need to briefly review the program development process. If you recall the *dark times* in computer programming then you remember assembly language. Assembly, or machine language, was the original way to program computers. Only one level removed from the 1's and 0's computers *really* understand, assembly language is very difficult for biological systems (people) to understand. So to save our sanity *high level* programming languages were designed. The idea behind high level languages is that the computer, itself, can be used to translate the high level language into the more basic machine language. This method of programming quickly became successful but its success brought about another problem — debugging.

The problem with debugging high level languages stems from two basic sources. First, operating systems are designed to support applications written in many different languages. In fact, on most computers, every language finally compiles (is translated) into a *relocatable* or *binary* module which is the same regardless of the original source language. This is called an *object module.* Object modules are then *linked* together by a system utility to form a finished application. The problem is that after the linking, the operating system has lost track of the original form of the language. When problems show up, the operating system only has information about the object form of the application and very little information about the original source.

The second problem comes from a mismatch in computational models. You, as a high level language programmer, think of the computer in an *artificial* way. If you program in COBOL, then you think of your computer as a COBOL machine regardless of the underlying hardware. This is one of the basic features of high level languages: you don't have to understand the *real* computer in order to do your work. The real computer, however, is usually quite different from the conceptual model created by the high level language. The problem is that the operating system relates bugs to you in terms of the real computer rather than the high level programming model. Faced with the error message like `SYSTEM STACK UNDERFLOW` the COBOL programmer is at a total loss. The COBOL model of programming does not contain system stacks or underflows. In order to deal with bugs of this sort, the high level programmer is forced to go to low levels in the system.

In a nutshell then, symbolic debugging is a method by which the programmer can receive information about bugs as well as look for bugs at the same level as the program was originally written. The successful symbolic debugger maintains the illusion of the computational model created by the high level language. By keeping the programmer at the source level during debugging, the symbolic debugger makes the same impact on the debugging process as the compiler made on the coding process.

How are bugs found without symbolic debug?

Without symbolic debugging there are three basic ways in which programmers debug their programs. They can analyze system output, they can put special debugging code in the source or they can resort to assembly language debuggers. Although widely used, each of these techniques has some serious problems.

Analyzing system output usually means pawing through miles of printer dump output. The application was running fine for a while and then it blew up. The dump comes out and then you try to reconstruct what happened. The problem, of course, is that it's difficult to find something wrong in the maze of maze of registers and memory locations. When you finally do find *something* you might actually be looking at second or third order effects. In other words the *real* bug caused something to happen which caused something else to happen which you saw (second order effect). In addition, you need to learn how to read dumps and, usually, you also have to learn a bit about assembly language in order to navigate the output. This type of debugging was the primary debug weapon in the arsenal of the batch programmer of the 60's

The next approach is to add special debug code. It's surprising how many programmers still use the "put in DISPLAY statements" method to finding bugs. This technique requires the programmer to insert little "hi, you got this far" messages in the program source code. The program is run and the output is then analyzed. This technique has several problems. First, you have to know where the bug is (roughly) in order to know where to insert the print statements. Second, inserting debugging statements alters the program and may even make the bug go away. Finally, the debugging statements are, by definition, never in the program when you need them. This means you have to put them back in, recompile, re-link and they try the program again. Also, unless the debug statements have testing associated with them, they can generate reams of output which must be carefully analyzed (shades of the output dump).

The final nonsymbolic approach is to use an assembly language debugger. These debuggers were developed by the assembly language programmers and are, in fact, pretty close to symbolic debuggers if you program in assembly language. The problem here, however, is that you must descend to the depths of assembly language in order to use them thus defeating the major reason for using high level programming languages. If, as many other programmers, you program in a *portable language* you may regularly work on several different computers. This means that you'll have to be adept at the assembler on several machines. Still, even with its problems, the assembly language debugger is usually the best alternative if full symbolic debugging is not be available on your system.

How is symbolic debugging different?

The approach used in symbolic debugging is different from the first two alternatives given above. It is also much simpler than the assembly language approach. The fundamental difference is that it is an interactive action. The programmer is dynamically interacting with the program being debugged. Other nonsymbolic approaches are static. You are analyzing a print out of what happened or you are watching the output generated by special debug code. With symbolic debugging you are encouraged to try experiments to prove or disprove theories about what might be causing a bug. If unexpected results are seen, you can quickly try a new course of action. In a large system without symbolic debugging it is often difficult even to discover which module the problem is in. With the right set of symbolic debug features this can border on simple.

As a maintenance tool for supporting a software system symbolic debugging is invaluable. Usually during this part of an application's life cycle it is supported by people who did not do the original development work on the project. Symbolic debugging techniques enable these people to watch the program execute. Its logic flow and operation become very real to the support people and their job is made simpler so their throughput is increased and the support costs are reduced.

How does symbolic debugging work?

The symbolic debugger is actually only one part of a series of cooperating programs. The symbolic debugger (depending on implementation techniques) requires information from both the compiler and the linker. Remember, during normal compiling, most of the source information is lost when the object modules are produced. In order to have symbolic debugging, the information usually discarded by the compiler must be passed through into the object modules. In addition, because object modules may be *relocated* in memory as a result of the linking process, the linker (or loader or segmenter) must output information to a special file called a *debug information file* or a *debug map file*. The result of the compile-link operation then is a completed application program along with a debug information file.

The basic idea behind a symbolic debugger is to be able to execute the application normally by issuing the appropriate start up commands to the host operating system. If, however, the program displays some aberrant behavior, the symbolic debugger may be invoked which then runs the program in a different mode making it available for debugging. During debugging there are two program executing: the application program and the debugging package itself. The programmer interacts with the debugging package while the debugging package interacts with the application program being tested.

There are, essentially, two classes of symbolic debuggers. These are *intrusive* and *nonintrusive*. An intrusive debugger is one in which the application is compiled in a special way which causes some amount of debug code to be placed in it. A program compiled with debug code inserted in it is said to be *instrumented*. A noninstusive debugger does not require any special debug code to be inserted into the application. Totally nonintrusive debuggers are rare and need a great deal of help from the host operating system. More than likely the symbolic debugger you will be using is an intrusive debugger.

It is important to determine the amount of intrusion which is required for debugging. Normally the compile process is altered in some way when the symbolic debugger is to be used. For example, in Microsoft's QuickC compiler, the user selects if the compile will result in a debuggable object module or not. Likewise in HP's TOOLSET COBOL debugger all source modules to be debugged must have the $CONTROL SYMDEBUG option. If the intrusion is small in terms of consumed program resources (memory space and execution time), then applications can be routinely compiled with the debug option. By doing this, especially on new applications, the debugger is always available when a problem is discovered. This removes the need to recompile and re-link the application when a bug shows up. In fact it is quite common for an address sensitive bug to vanish when a program is compiled in the debug mode. By always compiling in debug mode this can't happen.

What are symbolic debugger features?

Although source languages differ widely when it comes to features, symbolic debuggers are fairly similar in terms of the basic offering. Different debuggers are differentiated by how the features are implemented and the ease of their use.

The debugger model the programmer deals with is the original source file and the currently executing statement. Usually the source statements around the current statement are displayed on the screen along with a command area. The program under test is placed in a suspended state so that it is not executing and the debugger, itself, is waiting for commands from the programmer. Note that because of the required programmer *think time* a symbolic debugger is not normally useful for real time applications which must execute without interruption. If a program must read a sensor every 2 seconds and it takes you 4 seconds to type a command to begin execution, you're in trouble!

Most symbolic debuggers are command driven. This means that you type in commands to bring the debugger into action. DEBUG on the HP/1000 RTE-A is fairly typical. DEBUG's commands are one or two characters followed by one or more arguments delimited by various characters depending on the selected options. The arguments, however, reference objects normally found in the source language using at least some of the syntax of the original source language. These objects might be statement numbers or user identifiers (as opposed to octal memory addresses). PC's, having powerful screen management technology, can support debuggers which use a mouse.

The Microsoft Codeview debugger is a good example of a debugger which is both command and mouse driven. The mouse is a natural tool for the symbolic debugger. If you want to execute your program up to a given line, you just point at the line with the mouse and click a button. This highly tactile way of manipulating your program is both natural and easy to learn.

A good symbolic debugger should have as many of the following features as is possible:

Breakpoints

The basic operation of the symbolic debugger is to insert breakpoints into the program under test and then execute the program until it *hits a breakpoint.* When a breakpoint is hit, the program suspends execution and you can *look around.* For example, if you suspect that a problem is at line 100 (the program aborts from that location), you can set a breakpoint at that line and execute the program. When the program attempts to execute line 100, the breakpoint is hit and you can check the values of various data locations to make sure everything is as it should be.

The simple "stop when you hit it" breakpoint may be augmented by several other different breakpoint types. This may include *iterative* and *conditional* breakpoints. Iterative breakpoints usually have a count associated with them. The program is allowed to pass thought the breakpoint for a specified number of times (the count) and then the program stops. This is especially useful when you know that a problem occurs on a given line after a certain number of "events" happen. For example, your program blows up after reading 125 input records. You could place an iterative breakpoint with a count of 124 on the input read statement. The program will stop just before reading the 125th record.

The conditional breakpoint actually allows you to specify some type of test condition. When the breakpoint is struck, if the condition is met, the program will suspend. If the condition is not met, the program continues execution. For example, your program aborts when a value gets greater than 6742. You could set a conditional breakpoint with the condition "value greater than 6742". Each time the breakpoint is hit the debugger checks to see if the value is greater than 6742. If yes, the program under test is suspended for you. This is especially useful when a bug displays itself only when certain data values are being processed.

Single step

In addition to breakpoints another vital feature is the ability to single step source statements. This is usually accompanied with a representation of the current statement on the screen. This might be a marker to one side of the statement (CCS TRAX) or, more spectacularly, by changing the color of the current statement (Microsoft QuickC). Whatever the display technique, the single step operation should have these important features:

Simple command to repeat single stepping. When you are single stepping, you normally want to execute several lines one after the other. This should not require long command sequences. A softkey or a carriage return is all that should be required to execute the "next" statement.

Single step should come in two different "flavors". You should be able to step *into* subroutines or to step *over* subroutines. Stepping into subroutines means that you continue to single step when executing the subroutine. This gives you a look at the operation of the subroutine if you think the bug might be there. Stepping over a subroutine causes the subroutine to execute at full speed with the next single step operation at the source statement which immediately follows the source level call to the subroutine. This is important when you know that the subroutine is bug free, but you wish to trace the flow of the calling program.

Although not provided by all symbolic debuggers (multiple breakpoints accomplish the same thing) single step execution is a real time saver when trying to follow complex logic flow. For example, the outcome of a "go to depending on" statement is simple to figure out with single step, but quite a bit harder to figure out using breakpoints.

Variable display and modification

There are several different ways to implement this important feature. Some debuggers (HP DEBUG/1000, CCS CView, HP TOOLSET) have commands to display the current contents of program variables whenever you want. The display command uses the source level name for the data object as well as the source level syntax to access the object. This can be quite complex. For example, the C debugger CView from CCS allows you to enter a complete C expression which is then interpreted with the computed results printed out. A more common approach is to enable the programmer to print out simple data identifiers, perhaps indexed with constants.

A different approach is the one used by Microsoft's Codeview and to a lesser extent HP's TOOLSET. These debuggers allow you to define *watch expressions* which automatically display the contents of user identifiers whenever the identifier changes during the execution of the program. This feature coupled with single step can be very useful for determining when a data value gets incorrectly changed. The Microsoft implementation of this feature allows the programmer to open up a special watch window which contains the current values of specified variables.

The automatic display of variable changes as opposed to upon user request can take its toll in extra added code unless there is some assist from the hardware. The compiler must add special subroutine calls on every assignment statement in order to trap variable changes. The overhead may become prohibitive.

Another, less costly way of achieving much the same effect is to be able to *attach commands* to breakpoints. The commands are usually display commands. The idea is that the commands will be held in ready waiting for the breakpoint to be hit. When the breakpoint is hit, the attached commands are executed. Doing this, for example, would allow a programmer to attach a number of display commands to a breakpoint followed a continue execution command. Whenever the breakpoint is hit, the display commands execute followed by the continue command. The net effect is that the data objects are printed will little overhead.

A final aspect of data display is the format of the output. It is a useful feature if the debugger is capable of outputting data in several different user selectable ways. Sometimes, for example, you may wish to view a decimal number as a decimal number, but other times you may want to see the same value in octal or hex. Most symbolic debuggers accommodate this.

The flip side of variable display is variable alteration. Most symbolic debuggers enable the programmer to change the contents of variables. Again, the original source names for the variables are used along with the source syntax. All debuggers support changing the values of simple scaler variables (non array type). Some allow you to change the values of an array variable with a single command.

Being able to change the value of a variable is a surprisingly useful feature. It often allows you to change a good value to a bad value just to see how the program will react. Or it allows you to patch an incorrect value in order to continue looking for a different bug without re-linking. On the HP/3000, for large COBOL programs, the time required to re-segment can be large. Being able to find *another* bug without recompiling and segmenting is an important feature.

Flow control

Most debuggers offer several commands which effect the execution flow of the program under test. There are usually two different types of flow control. The first is the proceed command. Using a proceed will cause the program you are debugging to begin execution under the control of the symbolic debugger. The program will continue execution until it either terminates, is terminated by the host operating system or hits a breakpoint. There are several variations on this theme which are available.

For example, the DEBUG/1000 package gives the programmer the option of keeping the program under test *alive* after a system termination. This means that if the program under test executed an illegal instruction or violated the memory protection scheme, rather than terminating the program, control is passed back to the debugger so that the programmer can examine the program after the crash. DEBUG/1000 even allows you to re-execute the program after the halt (presumably after changing data values).

The proceed commands found in symbolic debuggers usually have several different options associated with them. The CCS TRAX debugger, for example, provides for a *temporary* breakpoint which may be set with the proceed command. The proceed command can optionally indicate that the program under test should execute up to a given point and then stop. Although the same effect may be achieved with a breakpoint, combining the breakpoint with the proceed makes life a little easier.

Another important (but dangerous) flow control command is the *go to* command. This allow the programmer to redirect control to another part of the source program. This is often used to re-execute statements after a variable has been changed to check the new outcome. The problem with this command is that its power often lets you abuse the source language by going where the compiler never intended you to. In COBOL, for example, you might circumvent a proper exit from a PERFORM statement by using a debugger go command. This might ultimately introduce a (non-

Separating Data and Processing
or
Building Databases for Systems Yet to Come

Matt Ohmes
Cognos Corporation
2301 East Lamar Blvd. #416
Arlington, Texas 76006

Introduction

This paper had a long and difficult birth.

Like most programmers during the last 10 years, I had been told of the glories of "normalized" data structures. But also, like most programmers, I did not really know WHY it was important to normalize. It was simply accepted as a theoretical maxim that "good" database designs were normalized. It was also accepted as common sense that "real programmers" didn't normalize; real programmers optimized!

There did not seem to be a practical advantage to normalization. It made it more difficult to program, maintain, and fine tune applications. On the other hand, if you just made a few adjustments to the database, here and there; things got so much simpler.

Since I considered myself a "real programmer", I never paid much attention to all this theoretical talk. After a while though, I began to notice some problems with my database designs. Oh, they worked all right for the original application, but the designs caused nothing but problems for anything new.

Several years and a few minor disasters later, I decided to re-analyze my approach. Why was the database that was so "perfect" for one application, "perfectly awful" for the next? Finally, the light dawned; "normalization"!

I had fine tuned and "tweaked" myself into my a corner for the last time. From that point on, all my data bases would be perfect paragons of normalized form!

Several more years and minor disasters later, I realized that normalization alone, was also not the answer. It was a technique. A set of rules that tended to yield a desired result. The result is easier to state than achieve: The "ideal" database is one that allows any number of applications to effectively and efficiently access and manipulate it. To achieve this result, the designers and programmers must, as much as is possible and practical, separate the database from the processes that acts upon it.

This paper addresses many of the pitfalls that I have encountered while learning the ins and outs of database design. It also lists some rules of thumb that I use in my data structures. Hopefully it will help you sidestep some of my "minor disasters".

What is Normalization?

Since this paper is not intended to be a formal discussion of normalization, I won't spend much time on the normalization process or theory. But, I will give a general description of 1st, 2nd, and 3rd "normal form" structures.

The initial requirement for any level of normalization states that all records (or "tuples" if your relational) must be uniquely identified. This unique identifier is commonly called the "primary key" and can be either a single field or combination of fields. Note that Image Detail sets cannot have a unique key item, but each record can still be different from others in the set; through a combination of fields.

First Normal Form stipulates no repeating groups. Simply stated, this means getting rid of all arrays. This applies to both explicit arrays (eg. Item MONTH occurs 12) and implicit arrays (eg. Item JAN, Item FEB, etc.).

Actually, First Normal Form prohibits a variable number of repeating fields in a record, but this restriction is generally expanded to include any arrays. Arrays are usually eliminated by creating a new file in which each record would take the place of a single, occupied array occurrence.

Second Normal Form must satisfy First Normal Form, plus all items must be functionally dependent on the primary key alone. In other words, eliminate data redundancy. "Customer Address" should only be in the Customer file. It should not also appear in the Invoice file just because Invoices are sent to Customers. The Customer must be related to the Invoice file in some manner, but that relationship should be via the primary key of the Customer record. The goal is minimum redundancy.

Third Normal Form requires Second Normal Form, plus the elimination all transitive dependencies. Simply stated again, this means don't put vital information where all references to it might be eliminated by normal processing. For example, if all Invoice records are deleted from your system after they're paid, make sure that the Invoice file is not the ONLY place you store your customer's addresses. If a customer pays his bills promptly, you certainly don't want to lose track of him!

There is also a Fourth and Fifth Normal Form in relational theory. In practice, however, Third Normal Form is the generally accepted standard for most data processing shops. Third Normal Form is what I refer to as a "normalized" database in this paper. If you want a more thorough discussion of the normalization theory or process, there are many papers available. Three I have referenced in the writing of this paper are:

A SIMPLE GUIDE TO FIVE NORMAL FORMS IN RELATIONAL DATABASE THEORY
William Kent
Communications of the ACM
February 1983

and

SIX STEPS TO A NORMALIZED DATABASE
Paul Bass
Supergroup Magazine
May/June 1985

and

HOW TO DESIGN FOR THE FOURTH GENERATION
Leigh Solland
Baltimore/Washington RUG - Interex 1985
Paper 3013

There are literally hundreds of others available. Any that explain the process clearly are worth reading.

Why aren't most data bases normalized?

Let us assume that the average programmer or analyst can understand enough relational theory to describe a normalized data structure. Why then, aren't more data bases normalized?

The primary reason is, there is no perceived advantage to normalization. Most traditional programming languages do not readily lend themselves to normalized data structures. A Third Normal Form database does not necessarily make for <u>more</u> work for the average programmer, but certainly <u>does not</u> make for less!

Normalized data bases tend to have more files with each file having shorter records. Those files all must have separate open, read, check for end of file, and close routines. It is simply easier for the average programmer to have fewer files. Fewer files mean a lot less code in most programming languages.

It is also easier to build on familiar techniques. Even if those techniques aren't necessarily as relevant for present day computers. For example, many systems have been designed to minimize reads and writes at all costs, even though almost all systems can utilize Multi-record reads and disc caching with little or no programmer intervention. This doesn't mean that disc I/O should not be minimized. But it is not the hobgoblin it once was.

It should be noted however, that normalization does tend to penalize record retrieval. Data that might be on one file in a traditional structure might have to be read from two or more files in a normalized structure.

Why should data bases be normalized?

If it is so much trouble to use normalized data bases, then why on earth should we even bother? The answer is simple; normalization yields data structures that are stable, accessible, and flexible. The design minimizes data redundancy and maximizes consistency. If the database is designed properly, any subsequent application should be able to access and manipulate the data as well as the first.

This is the major problem with most "unfriendly" data bases out there today. Most were designed around a single application. Their structure is build to get the most out of that application. Unfortunately, most designers don't realize that the fundamental data entities and relationships for a company change very little over time. The systems that process that data, however, and the specific procedures involved, change more frequently.

This point must be clearly understood, before the advantages of normalization can be realized. A normalized database is inherently independent of its applications.

The "Danger Zones"

If you are in the process of designing a database or building an application, you may want to run down a mental check list here. There are several "warning signs" of a non-normalized database. If these signs are present, some "rules" have been violated. That does not mean the database design is "bad". It is simply not perfectly normalized. As we'll see later, there are sometimes perfectly valid reasons for breaking the rules.

Arrays

Anyone who ever took a computer programming class in school learned to manipulate arrays. They are perfect for teaching looping and control break processing without actually getting into messy concepts like files and data management systems. They are also easy to understand and control, so most programmers quickly become quite dexterous with them. Programmers are people, and people like to stick with what they know, so arrays tend to crop up in a lot of data bases.

There is nothing wrong with arrays. In fact, they are very handy for certain things. Arrays are great for "summary" type records. If I have an individual sales record for each of my 100 products for each week of the year; at the end of the year, it would be nice to archive that data on 100 records. Each record would have a 52 occurrence array, and each occurrence would have the sales for that product in that week. That seems a sensible idea, so why wait until the end of the year? Why not use the array records for daily processing?

The most obvious problem is actually the most minor; the problem of unused occurrences. If we used our example array for daily processing, over half the array would be wasted space until the middle of each year. However, unless each occurrence was quite large, the actual space "wasted" would not significant.

The major difficulties with arrays arise when we change our processing requirements. What if I decide to record daily sales figures for the 10 most active products, but only want monthly sales of the bottom 50? My array made assumptions about my processing that had nothing to do with the basic data. The data includes product, total sales, and the period of time involved. The array forced my period of time to be 7 days. Would your boss allow you design a database that would permit only 10 products or only 50 invoices? The principle is the same.

Arrays force processing assumptions on data structures that inevitably must be revised, and arrays are notoriously resistant to revision. They also tend to be exceptionally tenacious. Programmers seem willing to go to almost any length to keep their arrays (eg. "Well, when the array overflows, we add another record with the same key, except we set a flag on the second record indicating it's a duplicate. Then we increment a counter on the first record showing how many duplicate records we have ...).

A normalized data structure may force more records to be read for a given report, but it is relatively easy to restructure, and it is very processing independent.

The "Everything" Record

The next "warning sign" is what I call the "Everything Record". It is revealed by the presence of very large records in relatively few files. The general idea seems to suggest that if you put all your data into a single record, you can save a lot of time and effort opening, closing, and reading files. The chief benefits are simpler file handling routines and fewer records to read. You don't have to go elsewhere to get any information. Normalized databases tend to have more files, but each file has shorter records.

Very large data records are rarely a conscious decision on the part of the database designer. Most "everything" records just seem to grow as applications evolve. It is a lot easier to add a new field to an existing record than to consider the implications of a new file in the overall design.

"Everything records" often suffer from a "merging" of data entities. If several fields have been added to a record to indicate a number of different client statuses, for example; perhaps a new "client-status" file should be created. That way new status codes could be created and old ones deleted without a structural change to the primary file. The resulting design would also be easier to comprehend.

It is difficult for programmers to understand very large record structures. It is hard to grasp the individual meaning and relative importance of 300 different data items in a single record. Normalization clarifies data relationships and anything that clarifies the structure, simplifies development and maintenance.

The Multi-Record Type File

The next "warning sign" is another example of "merged" entities. That is the multi-record type file. (eg. Header Records, Detail Records, Trailer records). There are few advantages to multi-record type files. The only one I can think of is a reduction in file opens. There are a lot of disadvantages!

Multi-record type files complicate programs. The programmer must check for file type, save record locations for later updates, check for different data types in overlapping fields, and a number of other irritations. Any record structure change is major problem. You can't just add a field, you must check every other record type in the file and make adjustments. Even minor changes cause major ripples.

Terrible performance problems can also result. At one site I visited, there was one Image data set, containing over 1 million records, that had 17 different record types described for it. Three record types comprised over

99 percent of file, two types had less than 100 records each, one had 5. To report those 5 records, over 1 million had to be read! The user base had little idea which record type was which. They were forced to decide if they should scrap an fairly satisfactory system or live with it, as it was, forever.

Multi-record type files perpetuate old, batch processing concepts. They come from an era when sequential files were the cutting edge of data management technology. There is little advantage in retaining this technique.

Calculated Items

Calculated fields are also considered a "warning sign" in data design. Although I have found no specific prohibition against stored, calculated values in relational theory, they are generally rejected in normalized designs. If calculated values are stored in the database, their accuracy is always in question. This restriction is usually rather loose, however. If accuracy of data is vitally important and processing time is relatively unimportant, calculated fields should be avoided. If processing time is of paramount importance, calculated fields are well worth considering.

All the "warning signs" listed above fall into a broad category I call "language specific designs". They result from applying application programming techniques to data structure design. When programmers learn how to "code around" unusual data designs, they typically add those techniques to their "bag of tricks". Unfortunately, they also tend to add the unusual data designs to that mental bag too. This perpetuates "unfriendly" designs. It's like the old saying: "When the only tool you have is a hammer, all your problems start to look like nails." Don't repeat a questionable design just because you know how to program around it.

File Specific Application Designs

Another broad category of "warning signs" is "file specific application designs". This is the opposite side of the design coin. In this case it is the application that suffers because the developer overuses certain file features. For example, if an entire application is built around the generic retrieval capability of KSAM, future systems may suffer.

A more common but less obvious example of a "file specific application design" involves item level locking in Image. Item level locking has widely been described as an important component in "strong" application locking. While Item level locking may be useful in a single application, it is an invitation to misfortune in the long term. Applications that rely on item level locking to achieve acceptable performance, are extremely fragile. If a subsequent application does not follow exactly the same locking rules as the first, Image resolves the conflict by using the broadest locking level requested. This could easily cause an existing application to suddenly become unacceptably slow.

For a more complete discussion of locking strategies and pitfalls, I would suggest reading Chapter 15 of the Image/3000 Handbook, "Picking the Lock". A general rule can be applied; don't design your database around the way you code, and don't code around the "neat tricks" in your file management system.

When should you "break the rules"?

As I mentioned earlier, the "warning signs" indicate some "rules" have been violated. But, there are valid reasons for breaking those rules. If performance concerns become overwhelming or a change will greatly simplify the overall processing, then bend the rules just enough to get by. Don't assume that all normalization should be thrown out the window just because one report runs slow. If possible, wait until the user base has a chance to exercise the system. They will show real bottlenecks, as opposed to predicted ones.

I know of at least three examples, from personal experience, where violations of the normalization rules dramatically enhanced the existing system, without compromising the overall design.

One site I visited did not have <u>any</u> calculated or summary fields, even though they had hundreds of thousands of transactions. Even the simplest of calculated reports took hours. Since day old data was accurate enough for management reports, several summary files were constructed. In each summary file, thousands of records were compressed into a few summary records each night. The reports then ran almost instantly against these shorter files the next day.

Another site had a system that was almost totally table driven. They had over 30 different types of "lookup" files in which the records all had the same format; "code-value", followed by "description". They were all put into a single multi-record type file to create a "table of tables". This kept the overall structure cleaner and made adding new table types much simpler.

The third site (a well known software vendor, very familiar to me) had a database in which each Customer was related to one or more CPU records. The number of CPU records per Customer was not originally keep on the Customer record. It was soon discovered that every Sales Rep wanted to instantly see the number of CPUs each Customer had. Rather than force that number to be counted on each inquiry, it proved to be more effective to count CPUs when they were added, store that count on the Customer record, then have a batch process re-count periodically for "insurance".

In all three instances, the data designs were "de-normalized" to a limited degree to aid in processing. But, none of the sites had to corrupt their basically sound designs; and all of the sites have added new systems on top of their original databases with little or no problems.

Conclusion

In the final analysis, subsequent systems are the measure of a database design. If the second application runs as well against your database as the first, then you have a strong, stable, flexible design.

Data (the database) and processing (the applications that use that database) are and should be separate. Processing is based on the needs of the user at the moment. Data reflects the information needs of a company over time. With accessible data and flexible data structures, MIS can plan for the needs of systems yet to come and keep the users satisfied. And that is, after all, our job.

USING INFORM, PROTOS, AND QUIZ
A User's Experiences

Richard Decker
Computer Task Group
Suite 644
World Trade Center
Baltimore Maryland 21202

This paper shares some of my experiences in developing sysems with COBOL and fourth generation languages. The fourth generation languages to be highlighted will be INFORM/3000, PROTOS and QUIZ.

INFORM/3000 is the report writing product of Hewlett-Packard's (HP) RAPID/3000. PROTOS is a product developed by PROTOS SOFTWARE COMPANY of AUSTIN, TEXAS that generates COBOL source code. QUIZ is the report writing section of POWERHOUSE products developed by COGNOS of CANADA.

A good working knowledge of the Hewlett-Packard database language IMAGE and screen formatter VPLUS must be mastered before the user can acquire all the potential that the fourth generation language PROTOS can offer. The user of QUIZ and INFORM does not need a knowledge of the database language or screen formatter. The difference is that PROTOS is a productivity tool designed to be used mostly by programmers, while QUIZ and INFORM packages are tools used by end users.

There is a learning curve to be considered when first beginning to use the new languages. I have learned all of the languages on my own without having the opportuinity of attending a formalized class on any product. Manuals supplied with all products are excellent, well written and organized. A person could become proficient in the language with about 40 hours of concentrated study and practice for PROTOS, about 20 hours of effort for QUIZ, and about 10 hours for INFORM.

OBSERVED PROBLEMS

Problem 1:

I have observed that when a fourth generation language is installed at a site a strong tendancy prevails to let the user play a predominate role in developing reports. After the database has been built, and the data loaded into the system, the screens are developed and a user can access data on-line. The user is usually very pleased with the quickness of the on-line screens, but in time, they always want a hardcopy report that shows everything. Hardcopy reports that show everything, sorted two or three different ways are not things that process very fast.

Possible solution to problem 1:

During system analysis all anticipated reports should be analyzed and an agreement reached with the user and analyst as to content before the analyst or programmer is transferred to another project. Reports that might take a long time to process might be faster in execution time if the programs creating them were written in COBOL. If the report accesses more than one file and various types of cross-checking and validation are involved within the report then the use of COBOL is recommended. Reports such as these are best addressed during development.

Problem 2:

On the other hand are 'ad-hoc' reports. These reports are those quick, short reports that answer 'questions of the moment'. Fourth generation languages are usually excellent for these types of reports. These quick reports are usually answering questions of data content. The data content is either of a certain field of data, or during a particular time period. There is usually a selection verb involved and one or more parameters associated with it. A problem arises if the programmers are busy with other things and the end user knows nothing other than the report writers of the fourth generation language. The problem is that production reports are written with the report writer, and some of the reports are made quite extensive in scope of data reported. A user can become very attached to a package and try to see just how intricate he can make a report before the entire system is brought to its knees. This type of crippling affect is not necessary.

Possible solution to problem 2:

More thought and on-going support should be given the user by the programing staff. Complicated production reports should be written in COBOL, or PROTOS, and some restraints should be placed on the user abusing the power of the report writers for 'ad-hoc' reports. This type of mal-practice causes lack of system control, and may very well cause the buying of more and more disk drives, and more powerful CPU's, all in the name of 'giving the user what they want'.

How is the programming staff suppossed to accomplish this on-going support when they are already backlogged with requests. Here are a few suggestions that may help. Situations differ, of course, from site to site. The reasons for backlogs are surely not all the same.

Suggestion number one: More uniformity of programming style.

If all programs were to look alike, maintenance people would know where to look for a problem. If a change were to be made to programs, the change would be predictably placed, and a better estimate of time involved could be given. Uniformity of style also includes uniformity of code. Copylibs should be used wherever possible. Skeleton programs should be the basis for all types of programs. Screen program development, report programs, and maintenance type programs all should have a well thought out pre-defined basis for their development. The cries of programmer creativity should be drowned out. If a person has a better way of doing a routine, or program, then let the idea be shared for the benefit of all. Having more than one style of code in a shop, or worse within the same system, is a nightmare to maintain. Besides, when you execute fourth generation languages, you are trading individual creativity for speed. The speed comes from the predictable way of expressing the syntax of the development language.

Suggestion number two: More interaction between users and the DP staff.

If a user is becoming obsessed with a package, and begins to be abusive with its capabilities, do not let it continue until the entire CPU is openly crying for help. Fourth generation report writers are ideal for certain types of reports. Determination of best use is always needed.

Suggestion number three: Normalize the database whenever possible.

Fourth generation languages work best in databases that are normalized. Normalization is linking of data with the use of keys and related information. A process of delimiting the data to a smaller and smaller number of records to scan is quicker than looking at all records in a file. Random access can be used with keys. Sequential search is needed if no keys are available.

Review the present structure of the database and see if benefits could be realized with some structured modifications. This suggestion could become very expensive to incorporate on existing systems. It may not be practical or possible on vendor systems. When developing systems in-house it is a primary concern.

MAINTENANCE

Different products take different approaches to their operation. PROTOS, because it will generate COBOL code as the output utilizes its own database to organize itself. QUIZ will utilize either DICTIONARY/3000 or a compiled schema, called QSCHEMAC, to organize and operate. INFORM utilizes DICTIONARY from which it extracts data.

Schema description

A schema is a flat file describing each data element of a system in its most primary form. For example last name is a alpha-numeric element of twenty characters, and item cost is a numeric element with two decimal positions. The schema then goes on to describe where elements are found within a file. In addition to file layouts, redefinitions of a element may be found. For example the element of telephone number may be redefined as composed of an area code, prefix or exchange, and the station number or last four digits. Editing information can also be found in some schemas. This information is usually identified after a verb in the schema, such as the verb TYPE. TYPE Dollar would indicate that all elements following should be formatted with two decimal points when shown to a user. Other TYPE editing would be Date in MDY or YMD format, or a picture clause such as Type 999.9999, a number having four decimal places.

Maintenance of the schema

If the QSCHEMA method is employed then a schema file is maintained using an editor subsystem. This editor file will contain the names of IMAGE database and formfiles. In addition the names of flat and keyed files to be used are also listed. The identification of each element used, size, type, and record layout of each file is also listed in this editor file. After the editor file is created, use of a utility program creates the QSCHEMAC file.

Each data element's size and type must be entered into the system by a user at least once. Each file name and layout must be entered into the system by a user at least once. These two statements are to be done whether DICTIONARY/3000 or an editor file is used with the products. So why use DICTIONARY/3000? Because more vendor products interact with and thru DICTIONARY/3000 than an editor file.

Maintenance of the dictionary

When using QUIZ the choice of using DICTIONARY/3000 or QSCHEMA is left to the customer. DICTIONARY is probably a better choice. DICTIONARY/3000 is available on the HP-3000 whether or not a fourth generation language is in use. The dictionary could be used as a stand alone reference source. Program DICTDBM.PUB.SYS (Dict database manager) maintains the dictionary. DICTDBM is not as easy to use as the QDDR product to input information into DICTIONARY/3000. The QDDR package also has some other nice features associated with it that are helpful in the maintenance or development phases of projects. There is an option that automatically generates COBOL copybooks for all the divisions of a COBOL program. In addition a user can view all information concerning a data element, dataset, database or file residing in DICTIONARY/3000. When DICTIONARY/3000 is used there is another product made by COGNOS called QDDR that uses menus and screens to update the DICTIONARY/3000. This is important if many flat files, or KSAM files are used in the system. The record layout of flat files and KSAM files are not automatically copied into DICTIONARY/3000. Image database rootfiles (showing database schema information) and FORMSPEC, containing formatted screen information, are automatically copied into DICTIONARY/3000 using a few simple utility programs.

Maintenance of standards for programs - Copybooks

While thinking of ways to increase productivity and quality of software used in today's systems, let's not overlook some of the more common and less expensive methods available to programmers.

The communication area (COMAREA) of the buffer for the screen handler VPLUS/3000 is not simple. Use of a copybook member for the COMAREA standardizes and thus simplifies the use of VPLUS/3000. The copybook member for the IMAGE area contains all the redundant parameters the IMAGE calls require, database name field, dataset name field, modes, buffer name field, etc. Using copybooks insure that those data elements appearing in many programs will always be named the same thing. This is appreciated by those persons doing maintenance on programs after the original author has departed. It is also appreciated by the original author when trying to get a program finished by a deadline.

Another use of copybooks would be to store the format to be used within the Identification division of a COBOL program. Within this format is an expanded Remarks section. Expansion would include all pertinent information about the program and how it relates to other programs and files within the system. This can prove to be an invaluable asset during maintenance of the system.

Type of systems developed

Normally my work is in a manufacturing environment. Users of my systems include plant managers, production supervisors, operators, office clerks, and secretaries. In general, managers require summarial information, operators need simple data entry screens which require minimal training to use, and secretaries need simple and flexible capabilities with their screens. Usually the systems developed are a combination of on-line programs involving data entry or inquire screens, and batch programs. In developing the screen programs VPLUS/3000, or FORMSPEC, was the screen generator used.

Thoughts on choosing a development language

If only one of the languages were utilized to develop a system the following drawbacks might be encountered. COBOL programs take time to develop and the services of a programmer. PROTOS would require a knowledgeable programmer or user. INFORM and QUIZ are limited when it comes to doing intricate edit checking.

Factors to consider when mixing the languages

COBOL only

If COBOL is the only language used there is usually a backlog of user requests. These requests can be for new systems, modifications to systems, corrections to programs, updates to 'hard-coded' tables, or need for more programs. This delay results from a variety of factors; the quality of the programming staff, the size of the programming staff, the tenure of the staff. The DP staff's inexperience in programming, or their understanding of a system under consideration, greatly affects the speed at which requests are satisfied.

COBOL and INFORM

When COBOL and INFORM are used there is an assumption that data somehow found its way into a database, or file and a user is extracting data almost in a reference type mode. INFORM can be used to create links between various datasets in a database, but in its simplest mode it reports data elements within a single dataset. During development work this can be a great asset. The product itself is very menu/panel driven which is good for novice user, however the progression of panels gets tiring quickly. A more experienced user is given the option of stringing out his request for a panel. The more experienced user can enter a string of "1,5,3" and will get to the data element screen within the database they want, enabling them to do what they want to do. Normalization of any database is strongly suggested when using a fourth generation report writer. Normalization will affect the retrieval time for requested information. Searching for keys within the data uses a short string of data called an index to locate the desired data. If the user requests data that is not indexed then all the data must be read to match the 'string of data' requested.

COBOL and QUIZ
When COBOL and QUIZ are used a more dynamic environment can be more readily established. By using the ACCESS statment in the QUIZ language linkage to various datasets is quickly and easily accomplished. The SELECT statement is used to retrieve a subset of information. The SORT statement allows a more readable report.

Steps necessary before and during use of fourth generation languages
First the dictionary has to be created. To accomplish this run the program DICTINIT.PUB.SYS. This will establish the DICTIONARY/3000 database. After the dictionary database is created it must be filled with information. To fill the dictionary database a utility program is ran. The name of the program to fill the dictionary is DICTDBD.PUB.SYS. Filling the dictionary with information about databases and formfiles is an easy matter. With the command LOAD and the name of the database or formfile the program goes to the rootfile or formfile and gets all information of data elements, datasets, and forms. At this point the user can access the dictionary and find information concerning dataset names, and what data elements are in the datasets. To allow uers to access information located in more than one dataset at the same time, relational linkages or groups must be set up in the dictionary.

Once the groups are established a user can more fully utilize the dictionary to extract information in a more logial form.

INFORM - Run the program INFORM.PUB.SYS. A series of panels is presented to navigate the user thru the program to extract data.

QUIZ - The dictionary again has to be established prior to using QUIZ. The dictionary can be generated by using the method discussed with INFORM, or another method can be used. QUIZ is a COGNOS product, and the purchase of QDDR could also be made in additon to QUIZ. QDDR is a series of formatted screens which allow data entry into DICTIONARY/3000. Once the dictionary is made QUIZ uses its own language to extract and report data. Statements such as ACCESS, SORT, REPORT, GO, LINK and others are used to manipulate the data and datasets or files. QUIZ is powerful in that with the ACCESS and LINK statements many datasets and/or files can be logically tied together to extract needed information. No previous links need to be established, such as in using INFORM. The use of temporary subsets of information is also a useful idea. As a QUIZ jobstream executes the user can direct that subsets of information be saved either permanently or just for the duration of the jobstream. This allows for easier generation of rather intricate reports. Much more detailed reports can be generated using QUIZ than INFORM with the same amount of effort involved.

PROTOS - The dictionary involved with this product is not the DICTIONARY/3000. Rather PROTOS uses its own dictionary generated from a schema that this product recognizes. There are more steps and planning involved using this product in comparison to QUIZ or INFORM. But we are talking apples and oranges here. PROTOS is a COBOL generating language, and is more powerful in what it can do than the other choices being compared in this paper.

In planning for PROTOS a PROSCHEMA must be made. A PROSCHEMA involves naming the databases and formfiles to be accessed. Declarations of data element's type can be made. These declarations involve date formatting, currency formatting, decimal alignment and redefines of an item into sub-items. Files are declared and their associated record layouts are given. PROTOS, when defining datasets and files, will give a prefix to the items associated with them. In the COBOL code these internally generated prefixes appear. An option is available where the prefixes are predefined by the user.

Several User Defined Commands (UDC's) are then used to initialize and ready the PROTOS product for use. The three main UDS's are PROINIT, PROBUILD, and PROCOPY.

PROINIT will build the PROTOS database dictionary and initialize it for use. The dataset capacities are pre-defined, but can be overridden.

PROBUILD will read the PROSCHEMA and fill the dictionary with information as directed by the PROSCHEMA.

PROCOPY will build a copylib. In the copylib will be all the file layouts, and buffer areas that will be needed by future programs. In addition all IMAGE and VPLUS buffer areas will be in the copylib.

At this point the user has a dictionary, a linkage defining database, and a copylib. Programs written in PROTOS syntax can now be written. The PROTOS language is easy to learn and not exceptionally complicated. It is very helpful to understand how IMAGE and VPLUS work when writing a program. PROTOS is a language of few words. A program that will do quite a lot can be written with very few lines of code. This is where the increase in productivity can best be observed. However because the language is so powerful, a thorough understanding of what can be accomplished by a single command is necessary.

After the PROTOS source code is written it must be converted to COBOL code. The UDC to accomplish this is the PROWRITE udc. After the COBOL code has been generated the normal compiling,and preping must be done prior to saving the object and COBOL source code. Some installations save both the PROTOS and COBOL code, while others discard the COBOL code after the executable code has been saved. It is advisable to make any future modifications to the PROTOS source and not the COBOL source code. Within the generated code any calls to IMAGE or VPLUS are performed using the SL provided by PROTOS. Being that all error checking is done within the SL, user edit checking and exception handling must be thoughtfully done.

Factors involved when choosing a language

All three vendors have excellent reputations, and all three give very good customer support. Let us approach this question of choosing a product from what the design of the system demands.

Design Considerations

If the new system is in its budding stages the analyst must converse with the user to determine what are their expectations of the system. Will the system have many on-line screens. Will the screens be very intricate or will they display information found pretty much in a logical area of data. A logical area of data is something like basic personnel information, component makeup of a product, current payroll information, etc.

If the data elements are somehow logically linked together then those items might be contained within a single dataset. If a screen is to show many items upon a single request from the user the screen becomes more intricate. A user may like 'busy' screens that show just as much information as can possibly be displayed in eighty columns and twenty-four rows. Another user will like menu driven systems that take them to a certain area of information and display only requested items of information.

Data entry screens usually look simple enough but internally many many checks have to be done to determine proper linkage to master datasets. I am speaking of a user defined conditions that make data entry more valid and accurate. Does a purchase order have a vendor? Is the vendor valid? Cross-checking and validation of data can become as involved as the user, money, time and expertise of people involved allow.

FORMSPEC can be used to develop the screens. The programs will be written in COBOL with or without the help of PROTOS. If PROTOS is to be used there are some considerations as to how screens are made. The naming conventions are most prominent. Data element names within the PROSCHEMA will only be fifteen characters long. This limitation of field size allows PROTOS to interact with the FORMSPEC product. PROTOS allows a very convenient technique to be applied to screens working with detail sets. The technique involves filling up the screen with as many detail records as the screen will allow. The screen is defined in the PROSCHEMA using the REPEAT verb and naming of the screen. When PROTOS operates it treats the area of the screen as a matrix, and automatically tracks data fields within the matrix. With this technique inquiry, and updating of many detail records is possible with a screen full of data.

If the system is to be fairly static any of the languages will do nicely. Static means once the information is loaded into a database the user will access the system to retrieve information in much the same way as a reference library is used. The data may be updated daily, weekly or less often but at the moment the user is requesting data, the system is fairly static. Personnel, payroll, and some inventory applications are in this area of a reference system. If the system will stay within one area of a plant, then this too becomes a reference type system showing a status of the area.

When systems cross departments, or location boundaries the systems become more interactive and more complicated. More cross-checking and validation is needed to insure accuracy and smoothness of operation between the various areas of the system. In these situations a language other than a report writer is needed. COBOL programs, or programs generated by using PROTOS, are needed to build the system. If many reports are needed PROTOS is very well suited to generate complex reports with minimal effort on the part of the programmer. Reports written in PROTOS are much easier to write and modify than a straight COBOL program. The reason for this is the use of the FORMAT verb in PROTOS. The program shows the 'format' or report layout of the report in the program. The fields are then defined by just naming the data elements to be shown, and control breaks are identified by naming data elements after the BREAK verb.

QUIZ is a very powerful report writer. It is easier to use than PROTOS. If the report is a listing of data element values it is very easy to use. The user accesses the dataset or file, and gives the REPORT verb naming the data elements desired. Then the GO verb is used and the report is formatted, column headers included, and generated to printer or other device. If subsets of information are needed, the SELECT verb makes quick work of extracting the needed data. A SORT verb is also easy to use and declare. Again, if the report is a set or subset of information QUIZ is excellent to use. The way that the database is organized can greatly affect the performance, but that is the problem of the analyst or database designer, not the language. It is when the data to be extracted is not a simple subset of information that QUIZ is more difficult to use. Extraction becomes difficult if the conditions to select data are based on data elements not defined within the dataset itself.

Summary

When using fourth generation languages organization of data and preparation for use of the tool is very important. Various types of dictionaries and various dictionary utilities have been reviewed in this paper. Data structure has been shown to be the foundation for the fourth generation languages. Some dictionary maintenance products have been mentioned and briefly reviewed. In reviewing these products emphasis has been placed on how the product is set-up for operation. In general, the complexity of a program will dictate what type of fourth generation product to use for an application. The more complex a program the greater the need for COBOL. Once the dictionary is built and loaded with data of the system, extraction of information has been greatly simplified by these languages. It is this simplification of extraction that can affect system performance. Reports should be scrutinized to determine if the power of the languages is 'being used effectively. System administrators should be aware of how a user is accessing the data to determine efficiency and "bottle-necks" caused by abuse.

The Information System Lifecycle

It's Tough
When There's No Can Opener

By Mark L. Symonds
Innovative Information Systems, Inc.
63 Nahatan Street
Norwood, Massachusetts 02062

In years past I had been an economist. I heard a lot of jokes about that profession in those days. The most memorable of which involved three people stranded on that old familiar desert island. One was a chemist, one was a hockey player and the other was an economist. They had nothing but one large can containing food and supplies.

While pondering their dilemma, the chemist suggested they let the container soak in seawater until it rusted through. That way they could get at the food. The others scoffed at him pointing out that they would waste away from hunger long before getting their first meal. The hockey player insisted they should heave the can against the palm tree to break it open. The economist heaved a condescending sigh and said "There's an obvious solution to this problem." The others quickly leaned forward, listening intently for the answer. The economist began:
"First, assume we have a can opener ..."

Information systems development poses similar problems. If one assumes we have huge budgets, a large, experienced staff, well-defined requirements and flexible deadlines then anyone can put together beautiful systems ... in theory. When we assume we don't have a can opener, things get interesting.

Despite what some software vendors say, systems do not come in cans. It is not just a matter of "open and serve." We all know that implementing systems requires careful planning, hard work and diligent execution. Indeed, defining, developing and maintaining information systems is a complex undertaking that requires many different kinds of skills and management techniques.

My purpose here is to outline the steps, issues and challenges in each phase of the information systems lifecycle. I will also illustrate some practical techniques for coping with these challenges and for avoiding some of the pitfalls inherent in each phase of work.

There are a great many temptations, blind alleys and wrong turns on the road to a successful systems implementation. There is always a scarecrow pointing down the road of custom programming saying "there aren't any packages to fit our business." There are the packages that seem to have six bedrooms and 4 bathrooms as well as a pool and tennis court, but turn out to be only a nice facade. It is important, then, to have a good road map and to plan the trip. It is equally important to be a diligent and careful driver.

Information Systems Lifecycle

The Information Systems Lifecycle is a conceptual framework that will provide the basis for effective and efficient systems development projects. There are four major phases in the lifecycle:

I. Information Planning
II. Information Design
III. Systems Implementation
IV. Support and Maintenance

Each phase has its own set of challenges, pitfalls and rewards. Each in turn requires different skills on the part of MIS management and staff as well as that of top management. Before exploring those details let's look at what is usually entailed in each of the four phases of the cycle.

Information Planning

In general terms, information planning is the process by which an organization determines what information it will need over the next three to five years to remain competitive in its industry or to gain a comparative edge. It also looks at how this information should be gathered, stored and distributed so as to make most efficient use of the data and other resources such as hardware and people.

The product of the information planning process should be a set of goals and a detailed plan for achieving those objectives. These goals are set only after careful study of how the organization operates now, how it should operate and of what competitors might be doing. The action plan will define various distinct projects to be undertaken, prioritize those projects and estimate the costs, benefits and impact of each.

Information Design

As the projects identified in the information plan are undertaken, they enter the Information Design phase. Here, the detailed requirements of information content and flow are defined. Analysts assist users in determining the types of data needed, the sources of the information and the required timing and presentation.

In most cases a software package will meet a high percentage of the requirements for an application. It is during this phase that analysts evaluate alternative packages against their list of requirements to find the one that is the best "fit." It is here also that any required modifications to the system are identified and designed. If a package does not make sense, all the details of a custom system are designed at this time.

Other important steps in this phase are planning the conversion of data to the new system and sizing the hardware needed to support the application.

Systems Implementation

This is where proper planning and design pays off. Lack of it shows up very clearly as well. Tasks include detailed design and programming of package modifications or custom programs, user training, procedures development, comprehensive testing and conversion of data.

It is at this stage of the systems lifecycle that decisions and details can no longer be put off. A well-planned project will require a minimum of redesign and unforeseen effort during this phase. The potential for problems is well-known. One of Murphy's Laws is that the first half of a project takes ninety percent of the money and time allotted. The other half takes the other ninety percent.

Maintenance and Support

User requirements are not static. New ways of looking at the data emerge; new data need to be captured; new technology allows more efficient processing.

This phase involves ongoing support of users for questions and problems. It also entails bug diagnosis and correction and development of enhancements. This work is very different from the other phases because it is ongoing and sometimes repetitive.

Eventually it comes time to reevaluate the existing information architecture. The current systems are mature and have been modified and enhanced for some time. Often, an individual system can be studied and replaced without affecting the others. Periodically however, it is necessary to take a fresh look at the overall approach to information processing. A new Strategic Information Plan may confirm some approaches and highlight bottlenecks and inadequacies in other areas. So the process begins again to ensure that the organization has the complete, accurate and timely information it needs to maintain its competitive edge.

CHALLENGES AND TECHNIQUES

I. Information Planning

The strategic planning phase often poses the most difficult problems for MIS management. The concept of a Chief Information Officer has only recently gained a foothold in the Fortune 500. It is a very rare phenomenon for small to medium sized organizations to recognize the competitive and strategic importance of information planning.

A. Selling the Idea to Top Management

I've seen many projects fail due to infighting and lack of cooperation among departments. Commitment from senior management is essential to ensure that each user area do what it can to help attain the timely and successful completion of the project.

The key to getting the top people "on board" is speaking their language. Top management is accustomed to getting formal reports from other areas such as finance and operations.

Formal, informative reporting in terms they understand will earn respect and confidence. To the extent possible, estimates of cost savings, both tangible and intangible, should be provided. Over time, this practice will bring more attention to the importance of information processing.

B. Making it Pervasive

There is a great temptation to study carefully the areas that are most interesting. There is a tendency to gloss over ones that may yield large benefits despite the lack of glamour. Voice response order entry may get a lot of attention even though the system cannot allocate inventory properly and loses backorders.

It takes discipline to do a thorough job of information planning. It is essential, though, to touch every box on the organization chart.

C. Keeping Focused

An information may be well-planned and designed to study all important areas of the organization and still run into problems. It's a challenge to remain focused on the scope of the project and not to get distracted by fun details that should be studied later.

II. Information Design

Since the products of the design phase are less tangible it often does not get the attention it deserves. It is the thought of many that the design phase ends when the budget runs out. The major issues here are completeness and organization.

A. Complete Design Before Coding

The biggest temptation here is to start coding and seeing some results before the whole system is scoped out. Management and users are eager to see results and the staff is excited and anxious to get to "the fun part."

The fun part becomes a disaster, though, when the house is half-built and the bathroom and kitchen have to get moved to the other end.

B. Get Good User Input

Don't skimp on user interaction with the design staff. After the initial requirements definition users must remain available for questions and clarification. This is more easily accommodated when analysts batch their questions instead of interrupting users constantly. They typically have full-time responsibilities in addition to the development project.

Analysts should understand the business functions under review. A good analyst will lead the user through the universe of features and functions and not just take notes. An experienced business systems analyst can make this a two-way conversation drawing upon experience at other businesses or other areas. S/he can help the user define which ones are essential for doing his/her job effectively.

C. Prioritize

User requirement lists must not be taken as gospel. A knowledgeable user will include wish list items that would make his or her job easier and provide better information. The analyst should help the user rank each item as 1) Required, 2) Very Helpful or 3) Nice to Have. The nice-to-haves are often difficult and expensive to implement... especially in comparison to their benefits.

It is often the case that the users are not prepared to take full advantage of all the features on day one of the new system. The core pieces can be put in place first as long as appropriate "hooks" are provided to accommodate future functionality. In any case, generalized hooks should be built in to reduce the impact of unforeseen future requirements.

D. Be Creative

Too often we are constrained by past standards and perceived limitations. Creativity is the mother (or at least the mother-in-law) of greater efficiency and productivity.

III. Systems Implementation

Whereas the products of the design phase can be somewhat nebulous, the implementation segment has very concrete deliverables. It is here that poor execution of the previous phases really shows up.

If the planning and design are carried out diligently, the implementation will be much smoother, but there are still pitfalls.

A. Supervision and Resource Management

There is a paradox in most management situations. You want the most qualified person for a given task but you also want the individual to grow and learn. A good manager must balance these opposing ideals.

The key here is to delegate and then supervise carefully. The individual should have to reach to accomplish the tasks but not be so lost as to get discouraged. Proper training is an important element. Some specialized education will give a staffperson a sense of worth as well as additional tools to perform the job.

Making deadlines and expectations clear must be coupled with providing the means of accomplishing them. In this way project personnel will have more control over the outcome of their piece of the job.

B. Testing

Testing is not given its due often enough. The testing process should begin in the detailed design phase. It is there that the programmer/analyst defines the conditional logic and should identify test cycles and conditions. Rigorous unit testing will make the all-important integration test go more smoothly.

Package acceptance testing is also an oft neglected procedure. Few packages are 100% bug-free. I have also known there to be occasional errors when installing such software. Diligent acceptance testing not only helps shake out any problems but gives the in-house support staff a much better understanding of the programs and data structures.

C. Change Control

How often is it that users sign off on the design and are not heard from again? Changes to design during this phase cannot be made willy-nilly. The pressures of an implementation project usually result in such changes being thrown in without adequate forethought.

Some larger shops will not process any modification requests until after the system, as designed, is implemented. Of course, the tighter the design the easier it is to enforce such a rule. Again, proper design should reduce the call for last-minute changes.

IV. Ongoing Support

Maintenance programming is often thought of as the mailroom of MIS: not much excitement and not much growth potential. It doesn't have to be that way. One can distinguish himself here by providing consistency, creativity and organization.

A. Organization

Users need an effective means of reporting their problems or enhancement requests. There is room for creativity in defining the methods to be used.

A good approach will involve meaningful change request or problem report forms as well as meetings with analysts if more explanation is needed. These are complemented well by "user group" meetings which serve as an open forum to discuss system use and and proposed enhancements.

B. Feedback

Another valuable aspect of user group meetings is to recap and explain the status of bug reports and new modifications to the system. My former boss once told me: "If you're giving someone something make sure they know about it." Users have to be aware of the changes made for them.

C. Prioritization

Communication is a two-way street. Users must also make their priorities clear. When there is a difference in opinion management must be prepared to step in to moderate. Establishing and reevaluating relative priorities helps assure that scarce resources are put to the best use.

The information systems lifecycle is fraught with perils, pitfalls and temptations. Many can be avoided or at least minimized with diligence, proper planning and discipline.

The potential rewards of well-defined and carefully installed systems are enormous. A sound information architecture will provide a solid foundation for creating or maintaining a company's competitive edge.

Not even canned systems come with can openers and simplifying assumptions won't fly in the boardroom. Organization, hard work and resourcefulness are the next best bets.

TITLE: Minimizing Coding, Maximizing Production

AUTHOR: Karl Smith

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 0012

ELECTRONIC FORMS:
ANOTHER STEP ON THE ROAD TO THE AUTOMATED OFFICE

By Richard J. Armitage and William C. Tuminaro
Business Systems International
Canoga Park, California

THE CURSE OF THE PREPRINTED FORM

Forms are a good tool. They allow information to be categorized for ease of interpretation. They standardize and simplify the communication of complex corporate and legal data. A properly designed form can highlight information that would otherwise be lost in a sea of numbers.

Forms provide value, and that is why we use so many in business. Like any good tool, forms -- at least the traditional preprinted forms -- cost money:

* money to design
* money to print
* money to store

These are the visible costs associated with using preprinted forms, but what about the other costs -- the ones I refer to as "The Curse of the Preprinted Form." Let's briefly review some of these less obvious costs.

First, there is the cost of time spent finding the right form and then mounting and aligning it for printing. There is also the potential of using the wrong form. Here the cost is wasted time to re-run a job, or if no one detects the error, embarrassment or even a lawsuit.

Next, there are costs associated with managing preprinted forms. These include time and energy spent controlling access to forms, monitoring inventory levels and reordering forms, and making sure that only the current version of a form is being used.

The cost of obsolescence is a major consideration in using preprinted forms. As requirements change, forms must change; and the old forms aren't worth the paper they are printed on. Murphy's Law suggests that the obsolescence of a form will occur just after you have ordered a year's supply. (Many companies avoid the cost of disposing of obsolete forms by not throwing their obsolete forms away. Instead they hide them in dark places in the hope that the waste will remain undiscovered. I cannot help but wonder how much of America's office space is being used to store obsolete forms.)

Perhaps the most serious problem associated with using preprinted forms is the cost of keeping inefficient forms. These are forms which should be redesigned, but aren't because of the cost of creating and printing a new version.

Today, with the technology of laser printing and the sophistication of available software, there is no reason for a company to incur the excessive cost of buying preprinted forms or to continue to operate under the curse of the preprinted form. Electronic forms are here, and they represent a significant step on the road to the automated office.

In this presentation, we will discuss the electronic form as an alternative to the preprinted form. We plan to cover the capabilities of the laser printer which make electronic forms a viable alternative to preprinted forms in the typical office. We will then review the procedures involved in developing electronic forms, merging data with electronic forms from both mini and micro computer based applications, and finally we will review some actual cases involving the implementation of electronic forms. Throughout the presentation we will attempt to point out the advantages in cost savings and improved productivity resulting from the switch to this new technology.

First, what is an electronic form? Viewed very basically an electronic form is nothing more than a file containing instructions which allow a laser printer to print an image of the form. The instructions are stored in a host computer and sent to the laser printer's memory when needed.

Let's briefly review some of the advantages which electronic forms offer to the modern business office. We can begin with the obvious costs in time and money spent on designing and printing forms. With a preprinted form it can takes weeks to design the form, do the artwork, set the type, review proofs, make changes, and start the process all over again. Once the form is finalized, there is the actual cost of printing, shipping, and storing the preprinted form. With an electronic form you can create and begin using a new form in a matter of hours. You can fine tune the design without incurring additional costs other than your own time. There is no need to order and pay for a year's supply, you print the form only when it is needed. Because the electronic form is stored in the computer, it is always ready for use in each and every company location. You have eliminated the problem of wasted time spent looking for a form or inadvertently using the wrong form. And when the form is printed, it is done with laser printer quality and perfect alignment of data every time.

We spoke about the effort spent in managing preprinted forms. With electronic forms this effort is reduced significantly. Since forms are stored in the computer, you no longer have to worry about physical access control. You can restrict access to your electronic form files just as you restrict access to your computer programs. Certain forms can be restricted to individuals or departments, while others are universally available. You do not have to be concerned with inventory levels or lead times for reordering forms. Most important, there is only one version of the electronic form available -- the current version. You can forget about tracking down supplies of obsolete forms spread throughout the company.

Since forms are only printed when needed, there are no large inventories on hand when you decide to change a form. The redesign and implementation of a new electronic form is easy and straightforward. The day the new form becomes effective, it is the only form available for downloading to the laser printers.

One more benefit before we move on. With an electronic form it is easy to print the output on demand, and in the user's department. You do not have to batch the output until you have enough to justify the time to mount a new form, nor do you have to print all your forms centrally because your users cannot mount and align forms on their own department printers.

LASER PRINTERS -- THE HIDDEN CAPABILITIES

According to industry sources, laser printer sales for calendar year 1987 were approximately 500,000 units. Most of these units are being used to produce high quality word processed documents. But as those of you who have ventured into desktop publishing are probably aware, the laser printer offers some unique capabilities. We would like to discuss these first, and then discuss how they have opened the door for electronic forms. Our discussion of capabilities will focus on the Hewlett-Packard LaserJet Plus family of printers. These include the LaserJet Plus, the LaserJet 500 Plus, the LaserJet Series II, and the LaserJet 2000.

1. Print Quality

The laser printer with its 300 x 300 dots per inch resolution creates documents which come very close in quality to typeset documents. It handles both graphic images and text on a single page, and allows up to 16 different fonts to be used on a page.

2. Fonts

The laser printer offers an almost unlimited variety of fonts, with both fixed and proportional spacing. These can be mixed to replicate most preprinted forms. The LaserJet Series II has 6 internal fonts available plus two font cartridge slots. In addition, it can store up to 32 soft (downloadable) fonts with sizes ranging from 6 to 30 point.

Cartridge fonts (which are also referred to as hard fonts) plug into font cartridge slots in the printer. The fonts contained in the cartridge become available once the cartridge is plugged in. They do not have to be downloaded and they do not take up any of the printer's memory.

Soft fonts are supplied on diskettes or tape and must be transferred (downloaded) into the printer's RAM memory. Soft fonts generally cost less than cartridge fonts and provide more versatility in the combinations you can use and the size of fonts available.

Fonts are identified and may be selected by their individual characteristics. These include orientation, symbol set, spacing, pitch, height, style, stroke weight, and typeface. The printer maintains a table which contains the values of all of the currently specified characteristics. These values are described in an "escape sequence". We will discuss escape sequences later. Generally, electronic forms software will allow you to identify fonts with a number or short name which is cross referenced to the table of font characteristics.

3. Memory

The laser printer has its own memory which is used to hold fonts and commands for forms and graphic images. The memory is also used by the printer for buffering input and for building a page image. The LaserJet Series II comes with 512K of memory and can be upgraded with an additional 4 megabyte memory board. The LaserJet 2000 comes with 1.5 megabytes of memory and can be upgraded to a total of 5.5 megabytes.

4. Programming

Laser printers provide a control language which can be used to communicate from the system to the printer in order to access the printer features. Hewlett-Packard's printer language is the "Printer Command Language" or PCL. PCL provides four levels of printer features ranging from level 1 which handles basic character printing and spacing, to level 4 which handles page formatting. The original LaserJet uses level 3 (the office word processing feature set) while the LaserJet Plus family uses level 4. Each level is a proper super-set of the previous level, providing upward compatibility.

PCL commands are also referred to as escape sequences. These commands are used to control cursor movement, font selection, line drawing, job control, definition and placement of graphic images, and to access other printer features.

5. Storage of form files and graphic images

The laser printer's memory is not limited to storage of fonts. It can also store macro files which contain commands which direct the printer. These files are used to define electronic forms or other graphic images. The advantage is that once the files are stored in the laser printer's memory they are available for use throughout the day. Thus, in most situations, only data is transmitted to the printer from the computer. The electronic form is already present. The LaserJet Plus and the Series II can store up to 32 form files. The LaserJet 2000 has a capacity of over 32,000 forms.

These are some to the laser printer capabilities which make electronic forms a reality. Next, we will discuss the process of defining an electronic form.

ELECTRONIC FORMS DEFINED WITH PCL

If you analyze most business forms you will notice that they typically contain lines, boxes, shaded areas, text, and graphic images. Each of these components can be defined with PCL commands. With enough experience, anyone can design even a complex form using PCL commands and scanned images. For example, to draw a horizontal line the escape sequence has to specify the starting cursor position, the line thickness, the length, and the dot pattern. The PCL command to draw a line would look like this:

```
{ESC}*p300x400Y{ESC}*c90a150bP
```

It is not our intention to teach you how to write escape sequences. There is an abundance of forms design software which will create these PCL commands for you. It is, however, important for you to understand which components of a form can be drawn with PCL commands and which need to be scanned images, and the difference between storing a form in raster format or vector format.

First, let's review the components of a form and identify those specified with PCL commands and those which must be scanned images.

* Horizontal Lines - PCL
* Vertical Lines - PCL
* Boxes - PCL using horizontal and vertical lines
* Shaded Areas - PCL
* Text - PCL
* Logos - scanned image
* Special Characters - some with PCL, others must be scanned
* Mixed Orientation Text - scanned
* Reverse Text - scanned

At this point some of you may be wondering why not scan an entire form and forget about PCL commands. In order to answer that question, you should understand the significance of defining and storing a form in raster versus vector format.

Let's begin with an explanation of raster. Raster refers to images which are composed of groups of dots. When a laser printer prints an image it creates the image by laying down a pattern of dots (90,000 for each square inch).

These dots are actually specified by individual bits of data, each bit being either on or off. These patterns of dots can be communicated from the host computer to the printer in either their dot format (raster) or as vector commands which the printer converts to dots at the time the image is printed.

In raster each individual dot (or bit) is transmitted from the host computer to the printer. It requires a large amount of information (in byte format) to describe even a small raster image. For example, a one square inch box would require 90,000 bits (300 x 300) or 11,250 bytes of data. A complete 8.5 by 11 inch form would require over one million bytes of data.

In vector an image is communicated to the printer in a command format similar to the line draw command previously described. The printer converts the commands into dot patterns just prior to printing.

While raster provides ultimate versatility and flexibility in defining laser output, its use should be limited to those situations where vector commands are not available. Using vector commands to define an electronic form saves transmission time from the host computer to the printer, and requires less printer memory to store the form. It also saves disk space on the host computer.

Now that you understand that, we can get into how electronic forms are designed. There are a number of electronic form design methodologies available. We will briefly describe the most common, and discuss some pros and cons of each approach.

1. Scanning Forms

We have already touched upon this approach when we discussed raster images. In scanning, you use a scanning device to create a raster image of an entire form (lines, text, logos, etc.). Scanning a form appears to be quick and precise. But, actually the scanned form requires extensive editing to meet even a minimum standard of acceptability. Even with editing, a scanned form will rarely match the printers potential quality. Of course we have already discussed the size of a scanned form file and the excessive time to transmit the form file from the host computer to the printer. Another problem with scanned forms is the difficulty in registering the form for data.

2. Forms Designed with Commands

Many forms design packages allow you to draw a form by entering commands. For example, to draw lines and boxes you enter commands for the starting and ending positions and the thickness of the lines. This approach allows extreme precision and is especially useful for forms which will be filled in with computer data. No special equipment is required. A disadvantage is that the form cannot be viewed on the computer terminal. It must be printed. Actually for complex forms, the image displayed on a graphics terminal usually is not in sufficient detail to be of much value to the designer, so printing the form at each major step in the development is usually required anyway.

When and how the printer is prepared for data merging depends on your application. We will outline several alternatives which fall under the two broad categories, which we refer to as, Application Control and User Control.

Under the Application Control method, one approach is to download soft fonts and the form required by a particular application each time data is sent for printing. The sequence would be to send the soft fonts, then send the form, reposition the cursor by sending a PCL command, and finally sending the data. This approach should only be used for low volume forms that are printed infrequently. Its main advantage is reliability, that is, all required elements are guaranteed to be available when needed. The main disadvantage is the overhead involved in sending font and form files to the printer for each page of data.

A slightly modified version of this approach is to download the soft fonts and the form required by a particular application at the beginning of the application job stream. This eliminates the repetitive downloading of fonts and form for each page of output, while still maintaining a high degree of reliability that the fonts and form will be available when the data arrives at the printer.

A superior approach for handling printer preparation, in situations where forms are used on a routine basis, is to store all of the soft fonts and the electronic form files required by each of your applications in the printer's memory so that they are available whenever needed. This is the User Control approach. As we discussed previously, the LaserJet Series II can store up to 32 macro files in its memory. (Forms are stored as macro files.) Under this approach, you establish host computer job streams to download the fonts and forms required by each laser printer. The jobs are run each time the printers are powered up. Forms and fonts remain resident in a printer's memory as long as it remains powered up. Since more than a single form can be in the printer's memory, the printer must be told which form is to be overlaid with the data from your application. The printer has a feature called auto-macro overlay, which allows a previously loaded macro (form) file to be merged with data from an application. The signal to start auto-macro overlay (which is referred to as activating a form) is an escape sequence sent by your host application as the first characters in your output data string. Once a form is activated it will be overlaid with each page of output until another form is activated or until the printer is reset.

APPLICATION OF ELECTRONIC FORMS ON THE HP3000

Next we will discuss some of the considerations in using electronic forms on the HP3000. Typically, you will be integrating the electronic form with an existing application which is already printing output on a preprinted form with an impact printer. Your main concern should be how easily can the data merging requirements of the electronic form be integrated into the application.

Let's start with procedures for downloading fonts and forms to the laser printer. You need a program to download soft fonts and forms from the host computer to the laser printer. They cannot be merely copied with a utility program. The actual program is typically executed in one of three ways. The first method is on-line with the particular font and form files to be downloaded selected by the user or operator. The other two methods were described under data merging. The program can be set-up as a batch job which can be run by itself or as part of an application job stream, or as a subprogram called by the application program which generates the data.

One advantage in some software packages is the ability for the fonts required by a form to be automatically downloaded with the form. This saves time and eliminates the potential for error. Of course, the same fonts should not be repeatedly downloaded when you specify several forms, instead the software should identify the common fonts and only sends them once. Another valuable feature is the ability to group forms so that only a group name need be specified in order to download all of the forms in the group.

We spoke of activating the auto-macro overlay feature for a form stored in the printer. Usually this is accomplished by including the escape sequence as the first characters in the output data string. If a second form is required, its escape sequence is included in the next string of data. This procedure of sending an escape sequence, data, another escape sequence, and more data is referred to as dynamic form switching. After the last record of data is sent, the printer must be reset in order to terminate the auto-macro overlay. Usually this is handled by the spooler, however your program can also send its own PCL reset command.

APPLICATION OF ELECTRONIC FORMS ON THE PC

Whereas, replacing preprinted forms with electronic forms on the HP3000 usually involves an existing application program which is processing and printing the data, on the PC you frequently find electronic forms being implemented to replace preprinted forms filled out by hand or on typewriters. Thus, the first concern in dealing with electronic forms on the PC is determining the application to be used to fill out the form. Typically the application is a word processing program, a spread sheet program, or a data base management program. There are also specialized data entry programs which can be used to display a data entry screen that looks like the form.

The procedures for downloading forms and fonts from a PC to the printer are the same as on the HP3000. Likewise, the basic requirement for activating the auto-macro overlay feature is the same. However, implementation of the auto-macro overlay can be complicated with some PC packages which restrict the inclusion of escape sequences within the data string. In these cases the escape sequence to activate the form must be sent outside the data string, and if the program sends a printer reset prior to data, the reset must be disabled.

The bottom line is that there is no standard approach for handling electronic forms on the PC because each package provides different approaches for interfacing with the printer. At BSI, we routinely use electronic forms with dBASE, R-BASE, Lotus 1-2-3, Word Star 2000, Word Perfect, and Advance Write.

One important consideration in using electronic forms on the PC is control over the master version of each form. When you have individual PC users, PC network users, and HP3000 users all using electronic forms, it is essential that a central library of forms be established. Otherwise, you will soon fine various versions of the same form spread throughout the company. At BSI, we maintain the library of master electronic forms on the HP3000. Both our HP3000 users and PC users access this library for the current version of electronic forms.

IMPLEMENTATION OF ELECTRONIC FORMS - CASE STUDIES

In this section, we will cover two situations where electronic forms have been implemented with great success. The first situation, which was described in an article in the November, 1987 issue of The HP Chronicle, involves the Central Coast Computing Authority (CCCA) in Santa Barbara, California. CCCA is a public agency with data processing responsibility for the Santa Barbara Community College District, as well as two school districts comprised of 24 high schools, junior high schools and elementary schools. They service approximately 130 users, most of whom are at remote locations.

CCCA's initial application of electronic forms was for printing student transcripts. Under the old method, a student or parent requested a transcript from one of the member schools. The request was then transmitted on-line to CCCA, where it was held until off hours for printing. At that point it was necessary for the computer operator to mount the preprinted transcript forms and then print the small number of requested transcripts (usually from 1 to 20).

After printing, the transcripts were sorted by requesting school, and sent to the requesting schools either via mail or courier. Considerable time delays occurred routinely, and a number of people were involved in the process.

The implementation of electronic forms for this application was simple and straightforward. Most of the schools already had laser printers on site so the actual printing of the requested transcripts was shifted from the central facility to the individual requesting schools. Here is how the system works. The file containing the electronic transcript form is maintained in the central computer. Access to the form and the font files is via an on-line request screen which allows each school to request that these be downloaded to its laser printer. When a transcript is requested, the job is submitted on-line from the remote location. The job is processed on the HP3000 and the data is sent to the laser printer at the school where it is overlaid onto the form and printed.

The application was easy to implement, but the benefits were significant. Now instead of days, the turnaround time from request to printing is just minutes. Also, most of the man hours have been eliminated from the process since the computer operator and the mail room are no longer involved.

CCCA is a typical example of electronic forms replacing preprinted forms. The process was simple and no modification to the existing program was required. Another example of electronic forms is an application involving the TDY travel order form for the U.S. Navy. This application involves the use of both an electronic form and electronic signatures.

The Naval Ship Weapons Systems Engineering Station at Port Hueneme implemented a new on-line financial management system (referred to as STAFS) which included a number of very sophisticated on-line approval processes. One of these involves the generation of TDY travel orders for Department of Defense personnel. The TDY travel order allows government personnel to receive advances for business travel expenses. The system allows for the requesting, approving and authenticating of travel orders to be done entirely on-line. However, once the actual travel order is printed, each of the officials involved is required to sign the form.

At Port Hueneme, over 50 travel order requests are processed and printed each day. Prior to electronic forms, the approved travel orders from the STAFS system were printed in the travel office on preprinted forms, and then someone would carry them around the base getting the appropriate officials to sign the orders (which they had already electronically approved on the new computer system). The implementation of electronic forms and electronic signatures was designed to eliminate this duplicate effort, and to streamline the process of generating travel orders.

The system operates on a Hewlett-Packard Vectra computer which is located in the travel office and connected to the STAFS computer. Printing of the travel orders is done on a LaserJet Series II. Approved travel orders which are to be printed are transmitted from the STAFS computer to a signature merge program operating on the Vectra. The names of the requesting and approving officials on the travel order record are used to search for the appropriate electronic signature images, which are stored in encrypted form on the Vectra's hard disk. The travel order data, the signature images, and the electronic form are merged and printed by the laser printer.

The system saves time and money, but more importantly it allows the Navy to take advantage of the control and approval functions built into the STAFS system. The electronic signature image on the physical document is proof that the travel order was in fact electronically approved in the STAFS system.

CONCLUSION

We have covered a lot of ground in our discussion of laser printers and electronic forms. Hopefully we have succeeded in demonstrating that the laser printer has the potential to be an invaluable tool capable of assisting you in achieving tremendous cost savings and efficiencies within your office. At this point, we would like to recap some of the key advantages to be gained in using electronic forms in place of preprinted forms.

Reduced cost is one of the primary benefits to be gained with electronic forms. The direct costs associated with using preprinted forms are high and they continues to go up. The indirect costs of storage space and wasted time are also significant. All of these costs can be eliminated or drastically reduced with electronic forms.

Electronic forms provide you with almost complete control, something that is impossible with preprinted forms. Access to any particular form can be restricted to certain people or departments, and the current version of an electronic form is the only one available for use throughout the entire company.

Printing of forms can take place in the user departments without worrying about inexperienced people having to mount and align preprinted forms.

Finally, you never run out of electronic forms.

The bottom line is that the laser printer has opened the door to a new office technology -- electronic forms. With it, you can make your forms the efficient tools they were intended to be.

How To Keep Your Auditor Happy

Robert A. Karlin
Karlins' Korner
7628 Van Noord Ave.
N. Hollywood Ca.
91605

'Twas the night before audit
and all through the shop,
not a creature was stirring,
not the tiniest flip flop.

'The listings were placed
'neath the window with care,
in hopes that the Auditor,
wouldn't look there.

'And I with my coffee cup
clutched in my fist,
searching around for
what could have been missed.

'And as I await there
sweeping the floor,
I suddenly hear
a loud knock on the door.

'I rush to the door,
and what there attends,
but a three piece blue suit,
and eight tiny red pens.

'"On Debits, On Credits,
"On Dashes, and X's,
"Is the Hold Account right,
"Have we paid enough Taxes?."

'He moved through the room
searching for listings,
to find for himself
what controls we were missing.

'And when he was through
and his notations abed..,
he turned to my boss,
with a shake of his head.

'"All of these programs
"must be rewritten.

"Documentation is bad,
"and the Data's not hidden.

'And as he strode out,
you could hear, loud and clear,
"If you think this was bad,
"just wait 'til next year!"'

with apologies to Clement C. Moore

IN THE BEGINNING

In most cases, it is difficult to involve your auditor in a project from its inception. Usually, he will wait until most of the design is complete before descending from his office upon the project team. A few fairly simple design considerations will go a long way in providing him with comfort and peace of mind, and this can be very important in keeping the project on course and under budget.

COST JUSTIFICATION.

Our first, and primary concern in providing auditability for our systems is cost justification. If we are spending twenty thousand dollars to design and implement a system, we should not spend two hundred thousand to insure its auditability, if the system does not require it. If you are doing bulk mailings, It may not matter if you loose ten percent of your master file, if you can still select enough different names to fill your mailing requirements. On the other hand, if you run a bank, a listing for the IRS of all customers that have earned more than six hundred dollars had better balance to the penny. During the design of the system, the actual worth of the end product (worth, not cost - a three hundred dollar program may be worth millions to the corporation) must be weighed against the cost of any auditability design enhancements.

BACKUP AND RECOVERY

One of the more important areas that an auditor will want to examine is the Backup and Recovery scheme for your project. He will not be concerned with the daily and weekly backups for your overall system (though the scheduling of these jobs in relationship to you system is important), but specifically with the application dependent backups that should be designed into your system. If you nightly processing takes almost your full offshift time, ten minute backups at convenient times during the night could save hours of recovery time. Sometimes, copying one key file to tape could be the key to a timely recovery. If your daytime online operation is critical to the corporation, a lunchtime backup could be very cost effective. Closing and copying your transaction log file at reasonable intervals can also save time and money in the long run. If you chose this route, be certain that you are predictable. If you close down at noon every day for a half hour, don't close down at eleven fifty five one day and twelve fifteen the next. 'When the operator gets hungry' is not a particularly effective lunchtime backup scheme. Keep a reasonable number of generations of your backup, and keep some of them offsite. This is an important part of the original system design, and including the backup procedures will show thoroughness that auditors like. Note that copies of your software should also be kept offsite.

TESTING AND TURNOVER

A second area that an auditor will be interested in is your testing and turnover procedures. Procedures for both problem resolution and new software releases should be considered as part of the initial system design. Backout procedures should be part of the turnover of any changes to a system. Application system changes should be scheduled at non peak times, and the users should be forewarned that the change is being implemented, so that they may prepare for the possibility of catastrophic failure. All changes should be tested in an environment as close as possible to the production environment in which they will run, and the test results should be part of the change documentation. All changes must be reflected in the system documentation, and no system should be accepted that does not have sufficient documentation. The auditor will probably require the user to sign off to any changes to the application system, whether or not they will be visible to him.

DOCUMENTATION

System documentation is probably the first thing any auditor will ask for, and woe to the project manager that does not have it. The documentation should be written during the project development as an integral part of the system design package, and should be kept current as programming continues. The original program specs should be complete enough to write the program without the need for additional information, though I have yet to see a system where this could be done. At the least, the program specs should be annotated by the programmer when further information becomes available. File contents and usage should be kept in some form of data dictionary, and the dictionary should be kept current as the project continues. File layouts should be in copybooks, generated, if possible from the data dictionary, and all program should use the same copybooks. Keeping the documentation current is as important as the completeness of its content. System documentation should consist of, at the very least, input and output descriptions, the data transformation and formulae used in the programs, restart and rerun procedures, primary users, other users, system dependencies, balancing instructions, and program and job control listings. Copies of the documentation (either in machine readable form or on microfiche), should be kept offsite. You should also develop some form of change control documentation consisting of a description of the change, the reason for the change, copies of any changed system documentation, and backout procedures in case of a change failure.

SECURITY

Of all areas of system control, security is probably the most misunderstood. Most people consider security to consist of preventing unauthorized access to production data. This, in truth, is part of security, but is hardly the entire subject. Security should be considered as the protection of the operating environment from compromise or damage WITHOUT SERIOUSLY IMPACTING USERS AND PREVENTING THEM FROM DOING THEIR JOB. This includes providing the operations personnel responsible for the maintenance and monitoring of the system with the tools necessary to their work. Any good programmer, with the time to work at it, can access or change data surreptitiously in a production environment. It is better to spend time and money in ensuring that all accesses to production data are properly logged, and that unauthorized access can be spotted easily, than to try to prevent all access to the data. Tools such as QUERY should be controlled, as opposed to eliminated. If security is made so tight that programmers and users cannot do their jobs properly, the security system will be circumvented at the first opportunity. The balance user friendly and tight security is a fine one, and should be based on the actual worth of the data and system involved, not on a 'global' security mandate.

DATA INTEGRITY

The last area that we sill discuss from the auditor's checklist is the controls needed to ensure the veracity of the production data. At each step in processing, checks should be established to ensure that the data at this step is proper. All programs should, at a minimum, provide a record count of input and output records. Programs that do serial reads should also provide some form of balancing total that can be used to trace data as it moves through the system. A transaction log, containing dates, times, users and change data, should be updated for every change to the data base. Control totals from this log should be checked against the data base at appropriate intervals. Programs to insure the integrity of the database should be part of the original system design, along with maintenance programs that update the transaction log while updating the database. For image databases, DICTDBA, HOWMESSY, DBSTAT (or any of the other programs that do forward and backward reads of all the chains on a database), can be run on a scheduled basis, though a program to do the same function can be written and tailored to the application and would provide a better test of the database. In general, audit trails and balancing controls should be kept as simple as possible, and should be easily located for crosschecking against each other. If they are not, they will be ignored.

AS THE WORLD TURNS

One of the major problems that plague ongoing projects is that the personnel that finish a project may not be the ones that started it. The job of the auditor is made that much harder by constantly changing procedures and styles of coding and documentation. An auditor that does not understand your system is one that will criticize it heavily. A few important dos and don'ts may save hours of headaches writing responses to your auditors reports.

CONSISTENCY

If you feel the need to change standards in mid stream (and there may be very good reasons to do so), take the time to retrofit previous coding and documentation to match the current standard. But it works, I hear you say. If it works that well, then maybe your standards don't really need changing, at least as far as this project is concerned. On the other hand, if there is such a pressing need for revision, then there is a need for the recoding effort. If you don't schedule it as part of the change, it will never ever get done. It is far worse to attempt the maintenance of a system that does not follow a consistent standard than to attempt the maintenance of a system that follows a bad one.

INFORMATION AS IT HAPPENS

Copy your auditor on every design change memo, the minutes of every design meeting, and copies of all user memos. It may seen that by deluging him with what you might think to be trivia, you will be actually hurting your cause, but this is not so. If your auditor can see the development of your system as it happens, you will not have to explain as many of the design decisions you make to him down the road, when those reasons are clouded by time and subsequent problems. Also, by keeping track of the evolution of your system, your auditor will develop a sense of continuity within which your decisions will be more explicable.

AND THE MOMENT OF TRUTH

In the final analysis, whether or not your auditor is happy with your project will depend as much on how the project is accepted by its users as on anyone else's opinion. Your auditor will usually look in those places that have been pointed out to first. If your users are will informed, and happy with the results of the project, the auditor will begin his audit on a positive note. If your users are plagued by an unreliable system, and are not kept informed of changes, both planned and unplanned, the auditor will have half of his report written long before he enters the data processing department. It is difficult to keep all users happy all of the time, but by getting user signoffs on all changes, and by making the user feel as if he had a hand in designing the system that will ultimately determine how well he does his job, you can keep him happy most of the time. And this will certainly increase your chances of receiving a successful audit of your system.

AND SO ...

In summary, most points that your auditor sill bring to your attention are common sense approaches to the design of your system. Your auditor is most concerned with the protection of the production environment, and the ability to prove the results of the system under consideration. He will want to be able to pick up any report, pick a figure, and trace it back to its original source document. To do this, he will need good system documentation, a good audit trail of transaction history, the confidence that the report he holds coincides with the database at a particular time, and the knowledge that the programs that led to the production of the report have been fully tested and integrated into the system as a whole. In general, these points are the points that you should be concerned with as well. In some respects, your auditor is your conscience, keeping your from cutting corners that you know you shouldn't. Remember this, and you will survive even the most stringent audit.

User Friendly Security

Robert A. Karlin
Karlins' Korner
7628 Van Noord Ave.
N. Hollywood Ca.
91605

SECURITY: ... 2. something that gives or assures safety, tranquillity, certainty, etc.; protection; safeguard ... (Webster's New World Dictionary of the English Language Second College Edition: 1976)

From the beginning of the online business data processing environment, the question of protecting that environment has been asked many times. The answers have run the gamut from complete indifference to extreme paranoia. The propounders of these answers have collected numerous arguments, most of which have little or nothing to do with the basic question. In order to adequately explore the question of safeguarding the business data processing environment we will use the following definition of security:

SECURITY: The protection of the business data processing environment from damage WITHOUT SERIOUSLY IMPACTING THE ABILITY OF THE DATA PROCESSING USER TO CONDUCT HIS BUSINESS.

With the emergence of large DP staffs, many professionals have forgotten that the business data processing environment is not an end in itself but a service provided to enhance the efficiency of a particular business. Most security is installed and administered by staff members far removed from the user. Especially in these cases, security officers must balance the need for protection against the impact on the user's operation. In this paper we will explore the kind of considerations that must be taken into account when establishing or maintaining a secure environment.

SECURITY AND RISK ANALYSIS

In order to properly implement security in a data processing environment, one must know what one is protecting, and what one is protecting it from. There are no global rules on how much security is necessary, or what form it should take. Each particular installation must analyze its own environment, identifying the perils to be protected against, and assessing the cost of protection against the worth of what is being protected. We can group the perils in the data processing field into the following categories: Unauthorized Access to sensitive information, Accidental Damage, and Malicious Destruction. For the purpose of this discussion we will ignore the problem of Unauthorized Use (that is, use of hardware or software without either damage or the compromising of sensitive data) because with the exception of response time considerations, the problems of unauthorized use (such as the productivity of an employee who spends his day playing pong) should be handled in other areas (such as personnel) without making security either the scapegoat or the cure-all. Employees who cannot behave in a responsible manner will turn to other forms of diversion, while security restrictions to prevent this type of problem will do nothing more than lower the morale of those staff who are responsible employees. In addition to these considerations, it is impossible to assign a meaningful value to losses suffered due to the unauthorized use of hardware or software, and without a value the imposition of security is a meaningless gesture.

Each different type of peril has different formulae for calculating the loss that could be suffered. Some losses can be more political than actual, and proper security for these cases may consist primarily of education. Other losses, while real, may be incalculable. In these cases, one must be more concerned with the recoverability of the environment than in its protection. True disaster recovery will be part of any efficient security system, and if a disaster recovery plan is in place at the outset, the job of instituting effective security is made that much easier.

Full scale risk analysis for an existing shop can be both costly and time consuming. Most companies opt for ignoring the problem of designing a viable security plan altogether and therefore end up instituting security measures on a haphazard basis. This creates havoc for users and programmers alike, costs much in time and manpower, and usually does not protect the environment from many of the perils besetting it. To make the burden of risk analysis easier to bear, a surface analysis of the environment can establish the types of applications within the environment, and recommend the appropriate security for each type of application based on the both the value of the applications within each type, and the ease in tailoring the security tools available to that application type. New applications standardly can include a risk analysis, while old applications can be retrofitted as time allows, and the risks warrant. In this manner, all systems can be brought to a standard level of protection with far less impact on the current users of the systems.

When performing risk analysis it is easy to focus on the programmer as the main security risk. Aside from the obvious morale problems in a shop where each employee is treated as either a hardened criminal just waiting to sabotage the system, or a fumble fingered oaf unable to read a file without deleting it, a security system designed to keep programmers in small boxes can double development time, and prevent timely problem resolution. While it is appropriate to place some restraints on the programming staff, it is necessary to allow enough freedom to the staff to do its job. In general, one must be realized that one's programming staff works for the same company, and is not 'the enemy'.

TYPES OF SECURITY

Security comes in a number of forms, and not all forms are appropriate for a particular task. Each application must be examined to determine the most cost effective form of protection. Ease and cost of implementation, susceptibility to damage, and sensitivity of the data must all be taken into consideration.

The first and simplest form of security is the assignment of unique user identification. This security measure is the foundation on which many of the more sophisticated measures are based. Until Until unique ids are assigned, no responsibility can be assigned, and no audit trail can exist. Common user ids for departments or applications remove one of the cornerstones of a good security implementation.

Once unique user ids have been assigned, each user can be restricted to a single on-line session at a time. This prevents the usurption of a user id while the user is logged on, and reinforces good security habits by forcing the user to log off when he leaves his terminal. (If you don't think this is the case, watch a user try to sign on to show his boss a problem).

Password protection of the user id is the next level of security. Passwords should be assigned by the user himself, and changed at reasonable intervals. Requiring passwords to be changed too often, or assigning meaningless gibberish will result in passwords that have been written down and placed in desk drawers or taped to the terminal. Once the user id has been protected, additional passwords should be used sparingly if at all, since additional passwords will also encourage passwords to be written and stored instead of remembered.

Once the user id is secure, files and applications can be write-restricted to certain users. This helps preserve the integrity of the files. Data bases and other master files should be protected from any unauthorized write access. Transaction files should have a broader access, encouraging the correction of erroneous data by transactions (which are auditable) instead of direct manipulation of the data base (which is not).

Read-protecting data should be limited to instances in which the data itself is sensitive. Many applications overlap, and the integration of data and elimination of redundancy can be greatly hindered by a common policy of read restrictions.

For extremely sensitive material, encryption is superior to read-protection, but costlier to implement. The combination of the two is very secure.

Nonvolatile files can be protected by restricting access to batch programs only. This allows a complete audit trail for all activity against these files, and an extremely easy recovery path in the event of file corruption.

Restricting access to certain users ids, programs or files, can also be accomplished by terminal location. This form of protection must be applied carefully, since certain minor disasters that affect the location of the acceptable terminals can have major consequences. Also, late night and weekend remote problem resolution can be hampered. Restricting terminals to certain users by user id can pose the similar problems.

Any form of restriction carries with it certain costs not associated with implementation. Restrictions that hamper problem resolution can have serious cost consequences. Restrictions that force users to alter their way of business unnecessarily can also increase operating budgets tremendously. These factors must be taken into account in any effective security implementation.

UNAUTHORIZED ACCESS

Probably the most misunderstood area of security is unauthorized access. Unauthorized access involves the compromising of data or programs, as opposed to Unauthorized use, which is the use of hardware and software without the compromising of data or programs. It is particularly easy to apply the strictest security measures to a whole installation where only small portions of the data within the installation are significant. On the whole, there is usually very little sensitive data in an installation. This data divides easily into three types: economically sensitive data, legally sensitive data and morale sensitive data.

Economically sensitive data consists of information that could cause monetary loss if divulged, such as proprietary software, sales commission rates, future market studies, etc. Security in these areas is more to make the user feel secure, than to actually protect, since it is generally more important to the user himself than to any competitor, and since most of it is available on the corporate rumor mill anyway. In the user community, availability on a need to know basis is appropriate; for the programming staff, nondisclosure agreements are usually sufficient to protect the installation. On those areas that are truly sensitive, including merger plans, inside information, etc., nondisclosure agreements can sometimes be the only protection, since the data will be available to a programmer at the first application program failure.

Legally sensitive data presents a greater problem. Bank account balances, credit information, personnel records, etc. can all be the basis for costly suits if divulged. This data should be protected from any unauthorized user, and most of the programming staff. Only those whose responsibilities directly include production problem resolution should have access to this data, and again, nondisclosure agreements are a necessity.

Morale sensitive data can be the most difficult security access problem. The largest area of morale sensitive data is payroll information. Any company that does its own payroll is asking for trouble. If there is no way out of it, responsibility for payroll problem resolution should be relegated to one's most trustworthy staff, and no one who handles payroll data should be drastically underpaid.

Another area of morale sensitive data can be online interoffice memos. Most mail packages do not encrypt these missives, and many use data bases that are accessible to all users.
In these cases, staff must be cautioned from using online mail for any message that could be inappropriate for general release.

The general rule of thumb for unauthorized access is to assume that staff members are responsible people, and will, in general, behave in an appropriate manner. Truly sensitive material should be protected to prevent temptation, but minimal security measures are sufficient for most purposes.

ACCIDENTAL DAMAGE

The most prevalent security problem is accidental damage to software or data. All shops have experienced some form of accidental damage, and we include under this heading anything from the program bug to purging the wrong file. Each of us has experienced the joy of attempting a coherent application repair at 3 a.m., only to find in the morning that our fix has gone awry. Yet completely eliminating access to the production data can be a cure worse than the disease, as production program inconsistencies wreak havoc in our data bases, while we sit helpless to correct the problem.

Our first goal in handling accidental damage is to admit not only the possibility but the probability of error. For each application area, we must assess the extent of damage that is, if not acceptable, at least tolerable. How late in the day can the users be allowed access to their system before a true crisis sets in? How far back can the user recreate his input? Can the user survive if the nightly batch jobs were not run? Until these questions are answered, we cannot truly assess our risk, and apply the correct amount of security.

Our second step is to codify the types of damage that can occur accidentally. These usually break down into the following categories: Loss or corruption of production program, Loss of input data, Corruption of input data, Corruption of Database, and Loss of Database.

Most security problems relating to production programs occur while attempting to correct other types of problems. The best solution to this type of occurrence is the establishment of a separate operations group responsible for production turnover. This group could also double as documentation librarian and production problem support, and is an excellent way of apprenticing new programmers. Production turnover procedures must be rigorous, with write access to the production program group denied to all but the staff responsible. If the shop is too small to warrant a staff for this purpose, the system manager should take on this responsibility. Even in a shop with one or two programmers, the underlying security should be put in place as if a production turnover staff existed. Moving production programs (as well as JCL, standard copybooks, etc.) should be accomplished by preexisting job streams, parametrically modified to the task at hand and capable of creating an archive version of both source and executable program files. An audit trail of some sort, either a listing that is filed, or a file that is extended, can also be created by this stream. Proper program security includes the ability to easily back out any program change, in addition to identifying what was changed. Proper security also includes being able to easily identify the current program version, through compile date and/or version number prominently displayed each time the program is executed.

Preventing loss of input files, on the other hand, can be a nightmare that no amount of security will prevent. Identification of input is essential in safeguarding a production environment. All programs that create data that is input to other programs must create an audit count of at least the number of records passed. In addition, some other form of hash or logical count should be instituted. These counts can be displayed on control reports that are filed, or added to an audit file along with a time/date/program stamp. Online input should be logged somewhere, with an easily accessible way of determining if the data logged has been applied to the production environment in case of system or program failure. Just supplying a user with a transaction log in the event of a crash can save hours of manual labor trying to recover a day's input. If the integrity of the system warrants it, transaction backups can be taken anywhere from hourly to daily. Here again, we must weigh the cost of the loss against the amount of security to apply.

Corruption of data bases or input data is usually (though not always) accomplished by a program bug. A perfect program of more than a hundred lines has yet to be written, and no test procedure can effectively test all possible occurrences. All systems should have a method of correcting bad data within the system using builtin safeguards and audit trails, but all programs should also be perfect. The time will come when the input data must be tweaked, and good security must allow it. It is far more important that changes to production data by other than production programs be identified than be prevented. If it is too difficult to change the data within the security system, it will be changed outside of it, and any audit trail will be irretrievably lost. The minimum audit trail must be to log the access to the production data. The minimum security must require the bad data be backed up prior to being tweaked, for even bad data is better than none at all. If successful, the change should be signed off by someone other than the implementor, preferably the user.

Loss of data base through hardware or software can not be prevented. The only possible security for this type of problem is regular backups and transaction logging. Logging can be implemented through programming on files not managed by a Data Base Management System, and all systems that use such files should be evaluated for the necessity of such measures.

Loss of a data base through human error can usually be prevented through the use of appropriate access security. Privileged capabilities should be restricted to those for whom it is necessary. Even these users should have both privileged and non privileged access, and use the former only when necessary. Privileged capabilities should never be restricted to a single user id shared amongst those who need it, since this eliminates a major audit trail. Each user who may need privileged access should have his or her own user id, the use of which may be audited easily.

Our last step is the logging and analysis of all accidental damage to the production environment. Only by examining the pattern of past errors can we improve our security. We must restrain ourselves, however, from implementing random security measures in response to any particular event. Security must be established as a coherent structure of policies and procedures, not a haphazard collection of unrelated actions.

MALICIOUS DESTRUCTION

Rex Stout once commented that it was impossible to prevent a determined murderer. It is not difficult, however, to make certain he is caught. The same logic of course applies to most business data processing environments. One cannot secure one's system from those whose responsibility it is to ensure timely and accurate service to one's users. One cannot secure one's system from the user who must update it. Any attempt to place severe enough restrictions to actually protect may result in a morale problem deep enough to precipitate the very acts one is trying to protect against. The two major efforts in this area must be: first, to establish a secure enough environment to enable reasonable detection of sabatoge; and second, to isolate the perpetrator.

Securing the environment involves many of the steps outlined for accidental damage. A separate staff responsible for turnovers, sufficient backup procedures, including offsite retention of files back far enough to cover most contingencies, and separate production libraries are all good measures to help secure the environment. Other common sense items include removing access to hardware and software before an employee is informed of his termination, giving two weeks pay in lieu of notice IN ADDITION to any severance due, and enforcing password changes at reasonable intervals (two months is adequate). Creating an atmosphere where the majority of employees feel they have been reasonably well treated is probably the best safeguard. Allowing employees to feel that they are professionals, even when terminating them, will also help foster professional conduct.

Isolation of the perpetrator can be enhanced immeasurably by assigning sole responsibility for each area to different staff members. All senior and intermediate staff should have an area of total responsibility, and should be held accountable for knowing the current state of their area, including recent problems and changes. This provides a single area to audit on employee termination, either by the corporation, or by the employee's own decision.

In general, though, one cannot run one's business in the fear of sabatoge by disgruntled employees. Though stories of computer crime fill the newspapers, very very few employees actually resort to such tactics. Most staff are professional, and even if they are not, the effect on one's career of being discovered is enough to discourage even the most foolhardy. Only in anger will these employees attempt to damage your installation, and proper management is more important in preventing this form of sabatoge than any security implementation.

INSTALLING SECURITY

After delineating the perils that would affect your installation, it's time to analyze the extent of the security necessary for your installation.

First, isolate global perils, those elements that could bring your complete operation to a halt. The responsibility for these elements reside with your system manager and technical staff. Securing the system from these people is extremely counterproductive and probably impossible anyway. Establish instead a monitoring procedure to protect the system, regular backups to recover with, and 'walk throughs' of all system changes amongst the responsible parties. New releases of vendor software should be publicized prior to installation, and copies of the prior release should be available to return to. Never let a vendor, including HP, IBM, etc. install software without first, explaining the new release and installation procedures to you; second, explaining the backout procedures to you; third, allowing you to do a complete backup of your current system; and fourth, allowing you to talk to another installed site.

Second, identify your essential time critical applications, that is, those applications that MUST (not should) be completed or online at a particular time, or your operation goes down the tubes. These can include daily payrolls, order picking tickets, point of sale applications, etc. Determine the types of perils that these applications are subject to. Establish the minimum time it would take to recover from each type of peril and determine if a manual backup can be designed for complete disasters. Determine the minimum security necessary to protect against the majority of these perils and implement it on an application specific basis. This may involve programming the security into the application itself. Remember, these applications MUST be available at a particular time, and so you have no real choice in whether or not to implement security here.

Third, identify those essential applications that are not time specific, that is, they must be done but you have some leeway in recovery. Security here can be looser than in the previous categories, but these are still essential systems. With this category, you must determine the maximum time that you can live without each application, that is, how many hours or days have you to fix any problem. Application considerations for this category should be geared toward the auditability and recovery of data, as opposed to stringent internal application security.

Fourth, group those applications that should be run, but are non-essential or can be produced at a later time. Many user reports fall into this category. Some complete applications may fall here. Usually, standard system security is sufficient to safeguard these applications.

And if you have any applications in the fifth category, that is those applications that do not need to run at all, why are you still running them?

ARE WE HAVING FUN YET?

In conclusion, by applying security at the application level, in response to the perceived need of the application itself, you will find that you will need less security than if you try to apply security to your system as a whole. I will leave you with the following olio of security guidelines.

* When possible, allow users to assign their own passwords.

* If you are assigning passwords, do not make them overly complex, or someone will tape them to the terminal.

* Try not to require numerous different passwords. Use the user id to ascertain access.

* It is more important to log changes to your environment than to prevent them.

* It is more important to provide a tool to the user, than it is to protect him from the consequences of that tool. This does not relieve you of the responsibility of explaining those consequences to him, but he's a big boy, and should be allowed to make up his own mind.

* When in doubt, back it up.

* When in doubt, log it, count it, apply it ... but back it up first.

* Security should not be painful. If it is, you're doing it wrong.

* And finally, your employees' morale is the best security in any environment.

AI-The Three Toed Sloth

Robert A. Karlin
Karlins' Korner
7628 Van Noord Ave.
N. Hollywood Ca.
91605

The three toed sloth, the ai, is a large slow beast that lives in the South American jungles, where he hangs upside down from the trees, feeding on fruits and vegetables. Though slow and normally docile, the ai is a powerful animal, and can be dangerous when aroused.

Artificial Intelligence, that is AI, is also a large slow beast, living far from the ken of normal programmers. Close observation of AI by programmers more used to the mundane could easily lead them to believe that the AI has been programmed upside down, or at least by creatures that live in trees, feeding on fruits and vegetables. And AI, though slow and normally docile, can indeed be very powerful, and also very dangerous if misused.

INTRODUCTION

Since the beginning of the computer age, man has wondered at the idea of a thinking machine. From the Dybbuk of eastern European myth to the clockwork figures of seventeenth and eighteenth century fiction, thinking machines were usually envisioned as human in shape. In 1923, Karel Capek coined the word Robot in his play, *R.U.R.* , and since then, the word has been synonymous with mechanical intelligence. It wasn't until the creation of Eniac and Univac, the first commercial computing machines, in the mid 1950s that the actual mechanism by which intelligence could be imparted to nonliving structures took shape. Since that time, the search for artificial intelligence, has been ceaseless.

In general, AI has not touched the business market place. AI is still too new a field of study to have produced much fruit. However, this is changing. This paper is a look at AI from the businessman's point of view, examining what has gone before, and what is still to come, and how this will affect the business data processing department of the near future.

INTELLIGENCE

Before we enter into a discussion of what constitutes artificial intelligence, we should first see if we can define what it is that we mean by intelligence.

Webster's defines intelligence as 'the ability to learn or understand from experience; the ability to acquire and retain knowledge; the ability to respond quickly and successfully to a new situation'. What part of this definition is applicable to intelligence of the artificial kind?

By far the most striking difference between computer programs and human beings is the ability of humans to alter their behavior pattern based on experience. But it is possible to create programs that learn as well. The simplest example of a program that learns is the game program ANIMAL, a computerized version of twenty questions. To initiate the game, the program says 'THINK OF AN ANIMAL'. After the player responds with a carraige return, the program inquires 'DOES THIS ANIMAL HAVE FOUR FEET?'. If you respond in the affirmative, the program will say 'ARE YOU THINKING OF A CAT?'. If you respond that you are not thinking of a cat, the program will ask you what animal you are thinking of, and when you respond 'elephant', the program will ask for an indicative feature of an elephant. The next time the game is played, if the answer to the first question is affirmative, the program will ask the question saved from the first game to differentiate a cat from an elephant.

If the program again guesses wrong, it will store the information acquired this play. In a surprisingly short time, the program can accurately guess thousands of separate animals, usually with less than ten questions. This form of trial by error can be very effective for small problems, but almost worthless for anything larger. To illustrate, imagine a program that plays chess. After each loss, the software stores the last move prior to the mate, and eliminates it from its possible moves. To develop any coherent play in this manner would take years, even with our fastest machines, and the lookup time during play would be prohibitive. Current research in the area of software learning is concentrating on methods for deriving underlying rules of thumb, as opposed to specific courses of action.

It may seem that acquisition and retention of knowledge is the easiest segment of intelligence to emulate in machinery. After all, this is what we believe computers do best, storing data. And yet, just storing data does not really fit what Webster was driving at. Data must be stored in some usable form, indexed appropriately for later retrieval, and summarized into coherent structures. And this is where we run into trouble. We can store data much more easily than we can produce general purpose rules to identify and classify that data. We have trouble developing rules to allow a program the ability to distinguish between a photo of a beach ball and a photo of the sun. We also have trouble eliminating "noise" data, data that does not belong, from data that just does not fit. A human can easily look into a basket of objects and retrieve a particular size and color block, yet there is no program yet that can perform that task with one hundred percent accuracy.

The ability to respond to new situations based on prior experience may seem to be the most difficult section of our definition to emulate, yet is actually one of the success stories of current AI research. Rule-based 'expert systems' have been developed that can generalize from insufficient data and, responding to new situations with its area of expertise, produce fairly reliable results, but the key to a successful expert system seems to be as much in the artistry of the design analyst as in the developmental technique. We have yet to produce an expert system able to design other expert systems, though this project is certainly being pursued.

EXPERT SYSTEMS

The biggest success story of artificial intelligence research is knowledge-based or 'expert' systems. The first program that could be called knowledge-based was called DENDRAL, and was developed at Stanford University in the mid 1960s in order to help chemists identify compounds by spectrometric analysis. Since that time, expert systems have been designed for applications as diverse as oil prospecting and medicine. In addition to complete applications, expert system 'shells' are now available, allowing customers to tailor the system to their own needs.

All expert systems contain at least two basic parts. First, obviously is the 'knowledge base' itself and second, some form of knowledge interpreter to input and retrieve data from the knowledge base.

In order to represent a field of knowledge, it must be codified in a way that will make it both accessible and understandable. Some of the different ways of coding are logical coding, procedural representation, semantic nets, production systems, and frames.

Logical coding consists of formal logical statements. For example, if we take the statement 'all cows have four legs', then we could express this in formal logical expression as 'For any object x, if x is a cow, then x has four legs'. The advantage of logical coding is that the rules by which expressions are evaluated is based on centuries of philosophical research, and is known and well understood.

Procedural representation consists of small well defined procedures that process individual portions of the problem. For example, if we were building a natural language parsing program, nouns, verbs, adverbs, etc. each would have their own small procedure that determines what actions should be taken. The major disadvantage to procedural based systems is their complexity, creating problems in understanding the entire interaction of the base procedures and making debugging difficult.

Semantic nets most resemble the database of the business community. Objects, concepts and events are stored as 'nodes', and the interrelation of these nodes are stored as 'links'. A simple net might be:

```
   COW
    |
    | has-part
    |
    V
FOUR LEGS
```

The major problem with nets is that they are not innately valid, that is, the interpretation of what a link means is entirely in the hands of the software, and, unlike the logical representation, the data itself is no guarantee of its meaning.

Productions systems, also known as rule-based systems, store information as a set of rules, called productions, usually in the form of 'if condition, then action'. An example may be, 'If it starts raining, and you are outside, then open umbrella'. Because of their innate understandability, production systems have been useful for large application systems. DENDRAL, mentioned above, and PROSPECTOR, a geological program, are among the better known of these.

The last representational scheme we will discuss is the 'frame', or object oriented representation. Unlike rule based systems, where each 'unit' is procedural, and nets in which each unit is declarative, each frame includes both a declarative and procedural portion. This allows the frame itself to determine what action it should take to any action against the frame. Object oriented coding is becoming popular outside of the AI research centers, and object oriented PASCAL compilers are even now becoming available.

In addition to our knowledge base, we must have some form of program to analyze our queries and convert them into some form of database access. Each different form of data storage has its own type of 'inference engine' to form the bridge between the user and the data and to represent the methodology of the original expert. Much of the research in this area centers around the ability to control and predict how the system will function under a broad range of circumstances. These techniques are slowly working their way into the business community to produce management reporting and long range data analysis and prediction.

Expert systems have been successful, when they are, due to the narrowness of the scope chosen for each project. Knowledge based systems are massive undertakings, involving skilled 'knowledge engineers', who are responsible for interpreting the methods of each expert chosen for a system model, and converting this knowledge and methodology into a program and database. As any analyst knows, even the expert may not know what he is doing when he exercises his talent, so separating the substance from the window dressing can be quite an undertaking. Even after the system is complete, improved technology and current information must still be added continuously to keep the system from becoming obsolete. And yet, expert systems can pay for themselves in a single use, capturing an expertise that would be lost forever.

Even though we may seem to have produced the embryo of machine intelligence, we are beginning to realize that humans think differently than the models we have built. Let us look at how researchers have tried to provide machine intelligence with the ability to solve problems.

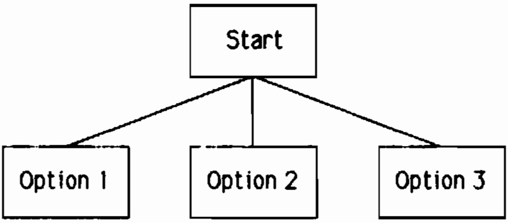
PROBLEM SOLVING

The area of game playing has been one of the most successful areas of research in AI. Most of the leaders of AI were fascinated with games research, due not just to the enjoyment of game playing, but to the limited domain that exist in a game environment, allowing a game simulation to act as a proxy for more complicated real world problems. One of the most popular game simulations has been, of course, chess. The first paper on the subject, 'Programming a Computer for Playing Chess', was published in 1950 by Claude Shannon, and many of the techniques described are used by today's chess playing machines. Chess is a good example of how problem solving techniques have developed.

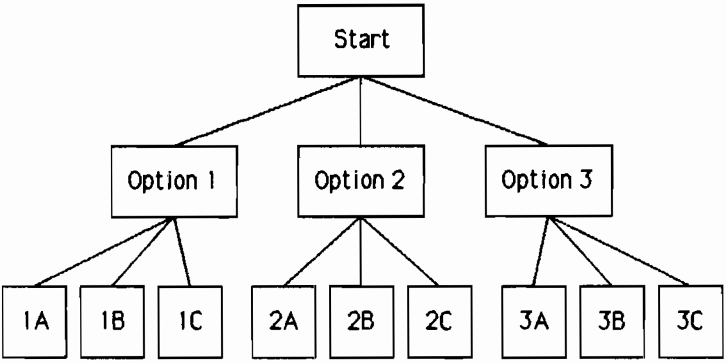
In the beginning, most scholars tended to believe that all that would be needed to develop a good chess program was enough storage and speed to examine all possible moves that could be played for, say, the next ten turns. Most chess experts, by the way, look ahead about six moves. However, if we say that, as an average, we can move at least ten pieces in any one move, and our opponents can move the same, we would need to evaluate 10^{20} moves to look ten moves ahead. Even if we could process a million moves per second, it would take about $3x10^{10}$ hours, or about 3 million years. Obviously, this is not quite acceptable for an afternoon game of chess. Some way must be found to shorten our search. This area of research has been one of the most important areas of AI, and is basic to almost all other AI areas.

In order to examine search techniques, we will represent our problems in the form of a 'search tree'. We start at the top of our tree as so:

From this point, let us say, we have three possible options. We would represent them as so:



Again, let us say that each of these options can have three possible options of their own:



And so forth.

There are a number of ways we can search this tree. The first, and easiest, type of search is called the 'depth-first' search. This, as its name implies, involves searching each branch of the tree to its bottom most leaf, that is, from start to option 1, to 1A, to 1B, to 1C, to option 2, to 2A, etc. Strictly at random, this type of search will, on an average, hit half of the nodes to be searched before finding a 'hit'. The second type of search is known as the 'breadth-first' search. This would involve searching each level, from start to option 1, to option 2, to option 3, to 1A, to 1B, to 1C, to 2A, etc. This search type has the disadvantage of needing a very large intermediate storage area to store the results of searching each level, and, in a random search, will probably hit a greater number of nodes. On the other hand, breadth-first searches are useful in many game situations, since they allow partial evaluation of each major limb of the tree before continuing the search, and this, in turn would allow the program to pick the limb most likely to contain the solution.

Another technique that helps shorten search time is called 'bidirectional reasoning'. In this type of search, not only do you start a breadth-first search from the top of the tree, you can start a breadth-first search from the goal. This is useful for types of manipulations, such as mazes, in which the goal and its precursors are clearly defined, and we are looking for a relationship between the start and the goal.

Another important problem solving tool is called 'means-ends analysis'. This method involves determining, for each node in the tree, what is required to accomplish the execution of the node. These are then analyzed for each complete path from the top of the tree to the required solution, and an optimum strategy for traversing the tree is evolved.

Most problems in AI involve using a combination of techniques to achieve a solution. The dynamic selection of search algorithms based on the type of problem presented is another continuing area of AI research, and is an area that can be of direct benefit to the business community. As the size and complexity of our data base technologies increase, we will need to include many of the problem solving methods of AI just to keep the cost of our inquiries manageable.

NATURAL LANGUAGE SYSTEMS

The ability to understand and use language has always been a yardstick for determining intelligence. Even after the discovery that other species used language, this fact was used as evidence of their intelligence, and closeness to man on the evolutionary scale. Even the intelligence of dogs and cats is not usually considered to be their innate ability to solve complex problems in opening doors and proceeding through mazes, but their ability to understand human speech. Is it any wonder that the use of a natural language, such as English, French, or Swahili as opposed to COBOL, PASCAL, or LISP, is considered prime evidence of machine intelligence?

The problems besetting the analysis of natural language can be easily illustrated in the following examples. First, examine the following sentences:

Time flies like an arrow.
Fruit flies like a banana.

In the first sentence, 'flies' is a verb and 'like' is a preposition. In the second, 'flies' is a noun and 'like' is a verb. This type of uncertainty can give software fits. A second example might be: The ball was hit by the boy with the hat. Did the boy hit the ball using a hat, or is the hat just the means by which we should identify the boy? Change 'hat' to 'bat' and answer the question again.

WHERE ARE WE NOW

We have looked at a bit of AI technology, and we have discussed a few of the applications now under study. We have completely ignored the areas of robotics; visual, aural and tactile perception; self programming and self enhancing systems; etc. The purpose of this paper was to give an overview of the field of AI, with emphasis on those areas that may become applicable to the business community of today.

In general, we have seen that AI is usually very costly, both to design and to execute. Many 'commercial' AI systems were originally developed in traditional AI languages, such as LISP, and PROLOG, and later converted to PASCAL, FORTRAN and even COBOL to achieve the speed necessary to compete in the commercial marketplace. As AI moves from a hit and miss field of study to a fully engineered science, more applications will find their way into the market place. Beware, however, the vendor who is trying to sell you an 'expert' system for $495.90. The most charitable point that may be made would be that an overzealous sales force tacked the appellation to the product without looking the words up. On the other hand, this could be another method of separating the unwary from their money. It will be quite a few years before even a reasonable subset could be found at computer boutique prices.

And yet, certain subsets of expert systems and natural language systems are beginning to appear economical. In not to many years, it will be common to find query languages that seem to be English, because the subset of English implemented will be fairly large. You should expect these programs to be very literal in their interpretation. Asking 'Can you get me the sales figures from October?" may get you an answer of 'YES'. These programs will also tend to get confused often enough for users to comment on the need to desk check the output before treating it as gospel.

We should also see improved database access techniques, with algorithms to optimize inquiries based on prior experience. Our concept of data storage will tend to include concepts such as 'self-defining' databases, whose complete structure, including source, responsibility, editing criteria and report format, will be stored as part of the database itself. We should begin to see object oriented extensions to our current languages. These are extremely useful in rapidly changing businesses, such as life insurance and commodities trading, and would ease the problems of implementing new products or modifying existing ones.

Though we may not see Karel Capek's robots in our offices next week, AI is here to stay, and will be a large part of the office of tomorrow.

BIBLIOGRAPHY

I have found the following books to be extremely interesting while researching this paper. I would highly recommend them to anyone who is interested in the subject of Artificial Intelligence.

THINKING MACHINES The Search for Artificial Intelligence by Igor Aleksander and Piers Burnett. 1987 (Alfred A. Knopf, Inc). This book is an exceedingly clear book describing the area of AI.

The HANDBOOK of ARTIFICIAL INTELLIGENCE volumes 1, 2, and 3 by Avron Barr and Edward A. Feigenbaum. 1981 (William Kaufmann, Inc). The definitive text on artificial intelligence. No study of the field could be complete without it. Again, this book is quite easy to read, and even the most difficult examples are presented with grace and style.

AI in the 1980s and BEYOND an MIT Survey, edited by W. Eric L. Grimson and Ramesh S. Patil. 1987 (Massachusetts Institute of Technology Press). A provocative look at AI present and future.

GODEL, ESCHER, BACH: an Eternal Golden Braid Douglas R. Hofstadter. 1979 (Basic Books, Inc). One of the unique books of our time, GEB is a marvelous blend of treatise and nonsense, exploring the world of artificial intelligence in a manner akin to Lewis Carrol, creating a world of Zen AI. This book is not a quick read, but it is well worth your while to explore it.

CONTROLLING THE DATACOM MONSTER: ONE COMPANY'S APPROACH

Jeffrey D. Van Brunt
Ireco Incorporated
11th Floor Crossroads Tower
Salt Lake City, Utah 84144

Introduction

In 1983 when I started working for Ireco, a Utah based international explosives company, our datacomm consisted of several local terminals and one 9.6 point-to-point leased line. This line connected our Salt Lake City office and our West Jordan plant, 20 miles away. In June 1985 Ireco acquired the Explosives and Nitrogen Products division of Hercules Inc., an acquisition tripling the size of the company overnight. In the area of datacomm, this meant we added five major manufacturing plants and one distribution facility with which we had to establish nationwide communications. We were very much in the dark as to what the needs of these sites would be. Adding to the complexity, we were just making a recommendation to replace our current hardware with Hewlett-Packard equipment. We knew we had to act quickly, so we took our best guess, ordered data lines and began implementation. Four months after the network was installed we realized what we had in place was totally inadequate. We also realized that we did not have the expertise to design a network that would handle our needs. There were questions concerning response time, redundancy, and reliability. We weren't sure how to solve these and many other questions. What were we to do? After a great deal of thought, we came up with a plan as to what we thought we should do.

About The Paper

This paper will discuss the approach taken to tackle the datacomm monster before us: one company's approach to achieving results in the world of data communications. I need to emphasize at this point that in no way do I feel that our approach is the only approach, or the best approach. I'm not even sure there is such a thing as the best approach. What we have done is develop a method that works for Ireco in it's particular situation. I hope that by reading this paper you can gather some ideas and generate some questions in your own mind as to how to control the datacomm monster in your company. The remainder of this paper will be a chronology of the events and steps taken at Ireco over the last three years in the area of datacomm.

A Word About Ireco

Ireco Incorporated is an international explosives manufacturer and distributor headquartered in Salt Lake City, Utah. It has operating facilities in 26 countries located on every continent in the world. Ireco is owned by Dyno Industrier A.S. of Oslo, Norway. Dyno is one of Norway's oldest and largest companies with it's history dating back to 1865, when it was producing dynamite based on Alfred Nobel's patents. Ireco consists of three manufacturing divisions: Industrial Explosives, Nitrogen Products and Defense Products. Ireco has four marketing divisions: Western U.S., Central U.S., Eastern U.S. and International Sales. Ireco also has wholly owned subsidiaries in Canada and Chile. Ireco is considered to be a full-line explosives manufacturer spanning all explosives markets.

Early Datacomm Network

Factors Involved

As mentioned before, Ireco was a very unsophisticated datacomm user when the Hercules acquisition took place. During this time we were a very small shop consisting of five people, none of which were "datacomm experts." What faced us seemed like a monumental task. We now had a total of six manufacturing plants and one distribution facility with which we had to establish data communications. We had been told of the pending acquisition in April of 1985 but were allowed no contact with the new sites until the final papers were signed. Once the papers were signed an agreement was made with Hercules to continue to allow the plant sites to use their existing applications for one year. We all breathed a sigh of relief since we thought we had some time to plan our strategy. The one task we immmediately needed to worry about was establishing a connection from Salt Lake to Wilmington, Delaware (Hercules' computer center) in order to allow corporate office personnel access to the existing Hercules computer systems. This was accomplished without too much headache. We sat back and started to plot our strategy for handling these new needs. We thought we had time! Before we knew it we were into September and were told that as of January 1, 1986 the manufacturing plants had to be online with Salt Lake. At this point we still had not been able to gather an adequate amount of information on the data processing needs of these plants. What we did have was a list of applications and equipment used to access the Hercules system. We also found out that it took 45-60 working days to get a leased circuit installed. Realize that while all of this was going on, there was the problem of finding adequate software packages to handle the increased needs of the company, and we were also in the process of converting to HP hardware. WHEW! It gives me a headache just thinking about it.

Why Our Decisions

Based on what little we did know, and with advice from a couple of vendors, decisions were made and the lines, modems and multiplexers were ordered. We were feeling pretty good and were excited about what was to come. Boy were we naive.

The design consisted simply of two multidrop 9600 baud leased lines with three drops each. At the remote sites, 9600 baud modems were to be installed to act as slave units as well as multiplexers capable of handling eight devices each. The one site that we had been communicating with was to be left as is. In Salt Lake there would be two master modems and a major multiplexer node capable of handling all the remote devices. We had at the time two HP3000's and it was decided that we would use DS/3000 with the DS pt-to-pt link for system to system communications. (see illustrations 1 & 2)

There are a few key points about the decisions that need to be emphasized here. First, the design was a best guess based on the information gathered. Second, the modems were purchased from the only company we had a history with. Third, the multiplexers were purchased from the same company that our existing equipment had been purchased from in order to protect what we thought at the time was an already substantial investment. This was done even though the local HP support people had had no experience with this company's equipment communicating with the HP3000.

Implementation and Problems

I would like to say at this point that the implementation of this network went smoothly, but I would be lying. The datacomm lines were supposedly all installed around the first part of November and the modems and multiplexers were also arriving. Again we thought, "no problem, we have almost two months to get ready." A trip was organized and I was sent out to install the equipment at three of the sites in early November. In addition there was cable to be strung in the offices. Well, to make a very long story short, there were problems at all three sites. Some of the problems were: the telephone companies not installing an RJ41C terminator, or if it was installed, not in the right place; the lines that were supposed to have been tested, were not working; not knowing how to properly configure the modems; bad cards in the multiplexer equipment; missing reels of cable for connecting terminals and printers; and countless other minor problems. After a two week trip I did manage to arrive back in Salt Lake with things semi-working. Over the next month+ we continued to try and iron out the problems and get things working by January 1. After many sleepless nights we finally did. The second week of January we were off and running.

The Next Step

Underlying Major Concerns

The implementation of our first network taught us one important thing: "We didn't know very much about data communications." It was decided that at this point some education was essential. A class was found and a couple of key people were sent to school. Some questioning and re-evaluation of our network was an important result of this education. We were already seeing some problems. There seemed to be alot of down time in the phone company circuits which caused lost work time. The network was already slowing down and all we were running was Payroll. The sites had too much effect on each other. This raised major concerns about performance and response time, reliability of the network, and contingencies in the case of a circuit being down for a long period of time. We were really worried since we were just beginning to add the load of applications that we were to eventually reach, and the sites were going to become increasingly dependent on the computer. We knew we needed a better network. We didn't panic and decided that we would take the time to do it right. For this we sought some outside help in the guise of network consulting.

Steps Taken in Design of Network

Once a consultant, or I should say consultants, were selected and fees negotiated, the first step was to interview some of the key users and management personnel. This was done for two reasons: first, to determine what some of the complaints about the way the network was functioning now were; and second, to make a determination of how long they could afford to not have access to the computer. The overwhelming answers to the first part were the system is just too slow and the lines go down too often. The concensus answer to the second surprised us. Users felt that anything more than a couple of hours downtime during the day would cost the company money. From this it was decided that two key criteria for the new network design would be to dramatically increase the performance (response time) and to provide for redundancy to avoid excessive downtime. Another criteria added was that the network had to be flexible enough to easily add sites. This was for both terminals and CPU's. The next step was to learn as much as possible about the traffic that this network would be expected to handle. Information was gathered from each site concerning applications used, volume of transactions/day, number of pages printed/day and peak processing times daily, weekly and monthly. This was an exhaustive list as even an application that was only accessed on a monthly basis was considered. Next we looked at the applications themselves. A program was written to run on a datascope that would measure the number of characters being communicated between the CPU and terminal while running an application. To me, some of these numbers were quite amazing. For instance in our order management system it takes approximately 10,000 characters to enter an order header and one line item. Added to this data was the area code and telephone prefix of each of our plants. We also decided at this time that we were going to install another HP3000, this one being in our Port Ewen, New York plant. One thing that made the design of our network difficult was that our plants are not isolated to any given CPU. They need to be able to access any CPU quickly at any given time.

Armed with all the information they could carry, the consultants went back to their office and hashed out a design. Much of the data gathered was input into a software package that analyzes traffic to suggest line speeds and helps in determining the most cost effective routing of datacomm lines. After a period of time, a design proposal was presented to Ireco. This design was discussed and refined several times before a final design was achieved.

The Design

The design in its final form is shown in illustrations 3 & 4. As can easily be seen, the new network was drastically different from the old network and much more complex. Following are some of the key parts of the design:

1. There are no multidrop circuits involved. It was determined that the line traffic was too great for multidrop circuits to be feasible.
2. The speed of many of the circuits was boosted to a speed of 19.2 in order to handle traffic demands.
3. There are in reality two paths between Port Ewen, New York and Salt Lake City: one running effectively at 9.6 and one running at 19.2. This provides for redundancy and the capability to keep running if one of the circuits fails.
4. Several modems have been added that have split streaming capability. This was done to allow the splitting of a 19.2 line into two 9.6 chunks.
5. Through the use of either dual dial restoral or re-routing of modems, every plant has the capability of continuing to function if their main circuit fails.
6. Louisiana, Missouri became a central hub for communcations.
7. DS X.25 link was chosen to do system to system communications. This was chosen because in conjunction with the multiplexer each system is only

one jump from another. It was decided, the system to system would be mostly limited to program to program communications, spool file transfers and network file transfers. The plan has also been to supply a PDN type link to our parent company in Oslo, Norway.

8. Even local terminals in Salt Lake and Port Ewen would be connected to the multiplexer in order to allow one-step access to any system. The multiplexers act as port selectors and the first menu a user sees is from the multiplexer.
9. It was determined that a network management system was needed to monitor the datacomm lines and control all of the modems from one central place. This provides a distinct advantage when dealing with AT&T. When a line is having a problem we can tell exactly what the problem is and relay this information to AT&T. This has helped us solve bad line problems quickly.

Test Phase

The biggest question mark in the new design was the multiplexing equipment. As mentioned earlier, the vendor was an unknown in the HP3000 world and we had experienced some problems with the multiplexors that had taken awhile to resolve. We knew there was other equipment available that was proven with the HP3000 but again we wanted to protect our investment and our jobs. So, in order to answer this question, a test plan was developed in which all possible types of connections and all possible applications were to be tested. A complete copy of the test plan is contained in Appendix A. An agreement was reached with the multiplexer vendor for enough equipment to conduct this test. HP was also involved in supplying modems, test sets and software for the HP3000. This test was conducted afterhours over a series of nights so as not to cause down time to users. There were a couple of significant factors that came out of this test:

1. It was determined that we would need 25 pin ATP ports on the 3000 instead of 3 pin. The reason was that if the phone line was dropped, or the session on the multiplexer was dropped, it left the session on the HP3000 and anyone could grab it. This was a security risk that we could not live with. A special cable was developed to solve this problem thanks to the efforts of engineers from both vendors.
2. We also found that we would have to leave the packet size in X.25 on the HP3000 at 128K because we were telling the 3000 that it was talking to a PDN. This would be a drawback, but we felt we could live with it.

After completing all of the tests, it was determined that the multiplexers would function adequately for our needs and orders were placed.

How Did We Implement?

At this point one thing was for sure; this implementation was going to be extremely tricky. First, we were adding two more CPU's to the network. Second, we already had a network in place and a good deal of the existing equipment was to be used in the new network. Third, we had to keep downtime to an absolute minimum. We thought, "If we can pull this one off it will be a miracle." The whole implementation was a carefully thought out phased plan, and it worked.

Phase I: Install the HP3000/52 in Salt Lake and add the users to it as well as those local users that would be going through the network to the multiplexer. Once the 52 had been installed, the setup of the multiplexer and rewiring of the computer room was completed over a three-day weekend. This also involved the establishment of X.25 communications between the systems, and the capability to move spool files and data files from system to system.

Phase II: This involved the installation of equipment at two of our plant sites; Louisiana, Missouri and Donora, Pennsylvannia. These were the two sites in addition to Port Ewen that were to have more than one modem. In these situations the modems would be back to back and required that the configurations be exactly right, which was nowhere close to default, and a user contact be trained to make the cutover.

Phase III: This would be the most difficult and complicated phase. This involved the installation of the HP3000/58 and extensive datacomm equipment in Port Ewen. It would also require additional equipment to be installed in Salt Lake and the cutover of the whole company to the new network. Again, this phase was done on a three-day weekend plus one working day down time. The 58 and the datacomm equipment was installed and readied over a period of 1 1/2 weeks. On Friday a company wide coordinated effort was made and the phone line cutover was completed. Over the next three days the cabling was re-done and the network tested. By Tuesday morning the whole network, with the exception of one site due to an out of specification line, was up and running. Even the site with the bad line was brought up with the DDR option.

The experiences we had gained from our first network installation helped us greatly in preparing for this one, and contributed to it being a success.

Strengths and Weaknesses

Now that we have been functioning on this network for a little over a year, there are some strengths and weaknesses that should be pointed out.

Strengths:

1. The strongest point of this network is probably the performance (response time) for terminal users. There is not much difference between a local (SLC) terminal and a remote site terminal, even at peak use times.
2. The ability for any terminal user to access any CPU directly is definitely a positive. This allows us to save X.25 for file transfers and the moving of spool files.
3. Although we don't use it very much, the ability to resume communications even when a phone circuit is down has helped us on occasion. We do limit the use of the DDR feature due to the cost of long distance phone calls.

Weaknesses:

1. The X.25 traffic should not be sharing the same multiplexer as the terminal traffic. It tends to cause performance problems for the file transfers and moving of spoolfiles. This problem is partially due to the before mentioned packet size limitation, and also to the fact that our projected load has increased.
2. We have had trouble with the quality of some of the 19.2 circuits. These have taken time to resolve, even with the help of our network management system. We have also had trouble with one of our 9.6 circuits and getting that fixed by a RBOC (Regional Bell Operating Company).
3. There remains a problem with performing file transfers using a PC connected through the network. This appears to be a problem with the way the multiplexer handles ENQ/ACK flow control.

All in all the network has done its job and performed according to design.

Where Are We Now

What Has Happened Since

One thing that I have been able to count on in the 4 1/2 years that I've been at Ireco is that things are going to change. There have been further acquisitions, the company's marketing organization has been re-structured, and our Canadian subsidiary wants to come online. This has all placed new demands and requirements on our datacomm network. We have once again been faced with the question of "What are we going to do?" Although we have learned alot in our experiences, we are nowhere near experts. So, once again, we have called on the consultants to help us meet the new needs that are before us.

The New Design

What we are doing with the new design at this point is to take our existing network and add to it. (see illustrations 5 & 6) We are adding tail circuits to accommodate our Olympia, WA. office and our Montreal, Canada office. We are also adding a subnet that will accommodate our central marketing region. As a part of this we are hoping to add another CPU in Salt Lake. We have added a PC LAN to the Salt Lake office, and we are going to LAN connections for system to system communications in the Salt Lake office. We're also pulling the remaining X.25 communications out of the multiplexers. Not shown in the illustrations but of equal importance is that connectivity within the manufacturing plant sites has become extremely important. We are currently investigating different ideas for accomplishing this. The implementation has not begun yet as the plans have just been finalized, but it should once again be challenging.

Future Considerations

In the future we hope to merge the new subnet into the older network to take advantage of hubbing and cheaper phone line costs. We will have an office in Atlanta, GA. that we will need to bring online, and we still hope to establish communications with our parent company. Eventually we would like to include the 58 in Port Ewen into the system LAN that will be present in Salt Lake. The main thing for us is to remain as flexible as possible so that further changes within the company will not continue to cause us headaches and sleepless nights, and so that we can react quickly to these changes.

Conclusion

I hope that sharing our experiences at Ireco will help others in controlling their own datacomm monster. With the speed at which equipment and standards keep changing, things can get out of control in a hurry. In closing I would like to reiterate some of the things that have been key to us.

1. Get outside help. There is no way we could have done it without the consultants. We are not datacomm experts. With the speed at which things

are progressing, and with new technologies, there is no way we can keep up. Besides, it's the consultant's job to keep up; that's what they're paid for.

2. Plan ahead and test the equipment before implementing. Salesmen can tell you anything. The sky is the limit for them. Make sure what they tell you will really work.
3. Get a network management system. Our network really isn't that big, but we could not manage without it. This has allowed us to have one person manage the entire network.
4. Remain flexible. You never know when things are going to change. There will always be new requirements and increased demands that cannot be anticipated.

Last but not least, it is also very important to have good people. As my boss likes to say, "You can have the best system in the world, but without good people you don't have squat."

IRECO US DATACOMM NETWORK

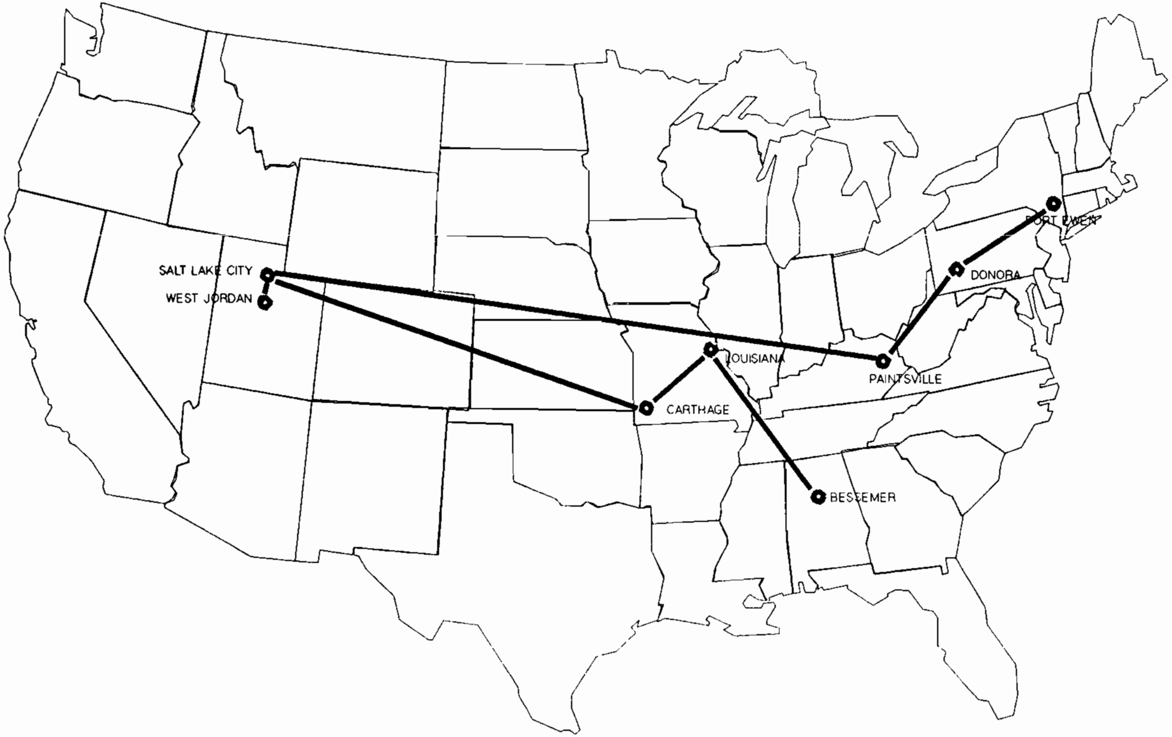


Illustration 1

ORIGINAL IRECO DATACOMM NETWORK

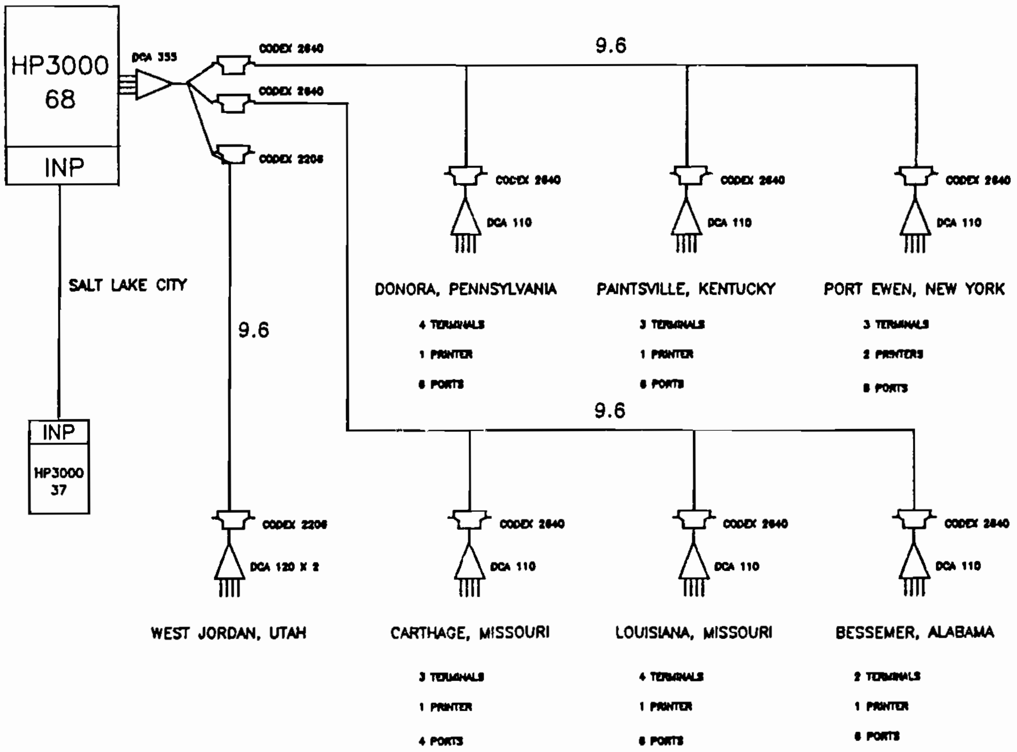


Illustration 2

IRECO US DATACOMM NETWORK

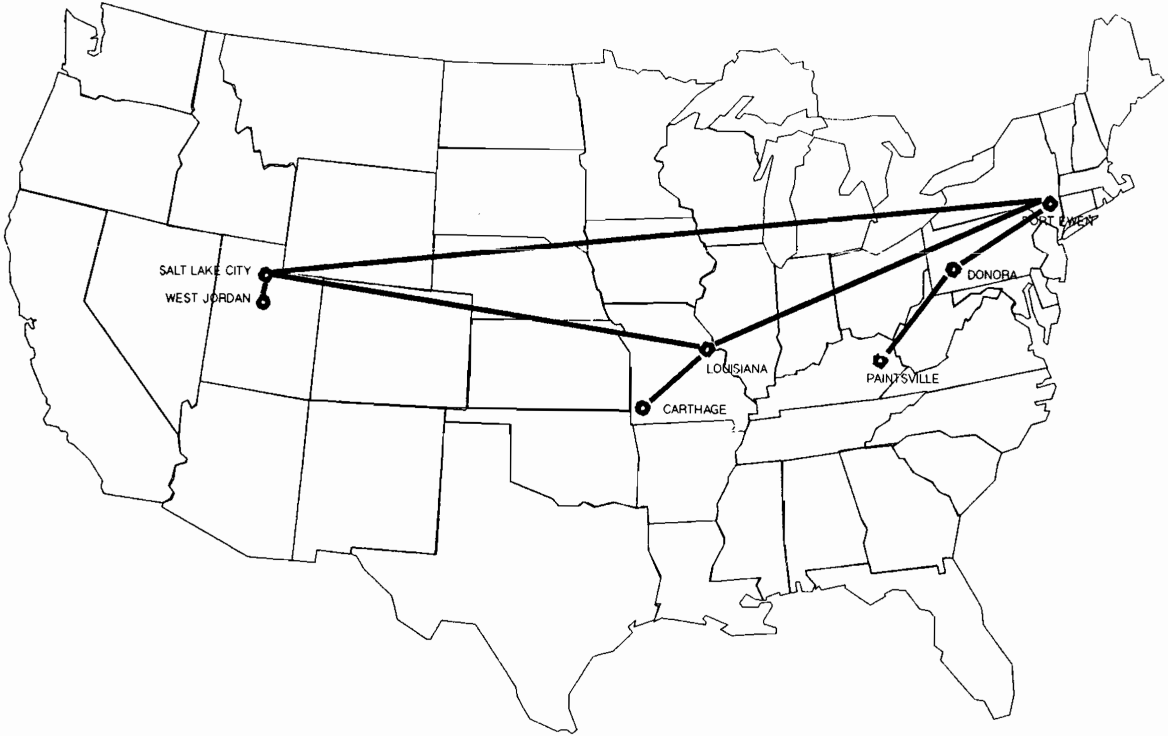


Illustration 3

IRECO

Data Communications Network

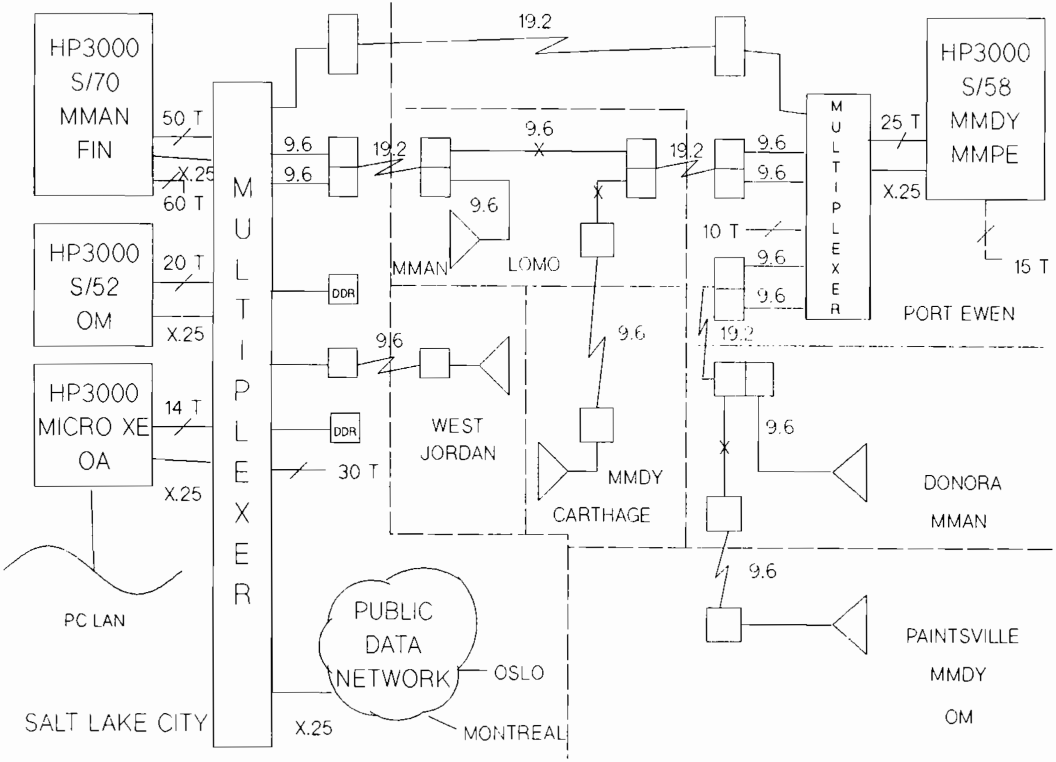


Illustration 4

IRECO US DATACOMM NETWORK

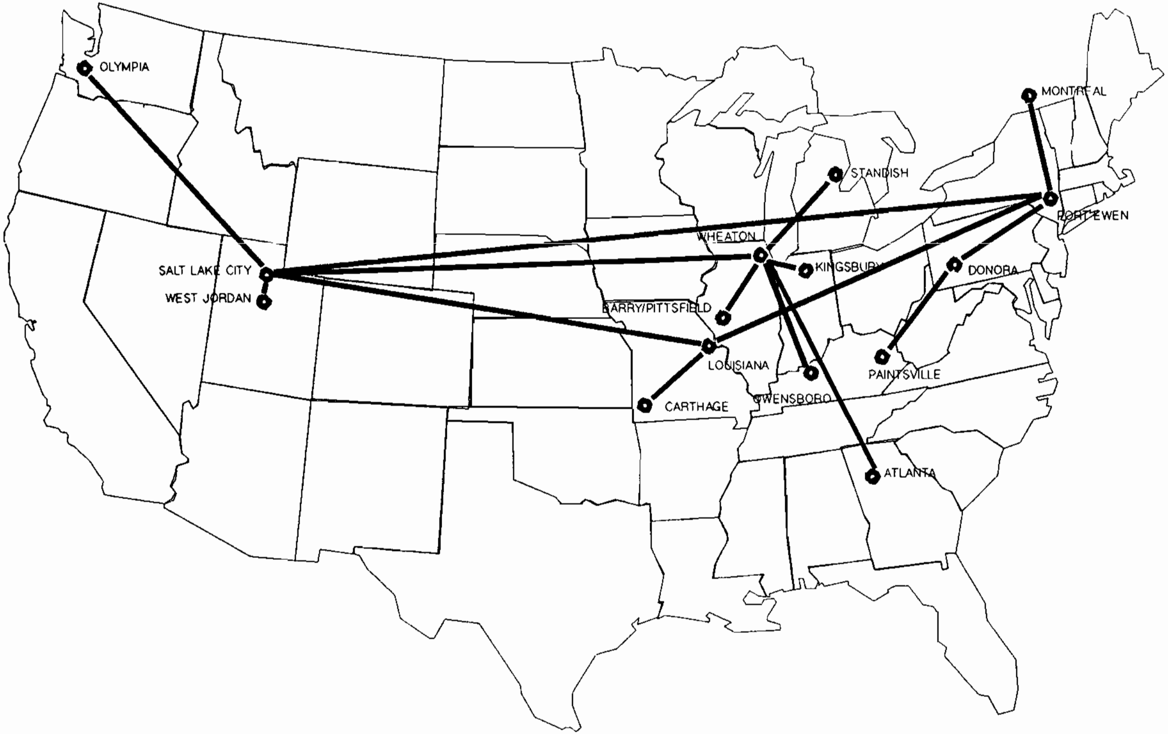


Illustration 5

IRECO Incorporated Communications Network

Processing and Switching Nodes

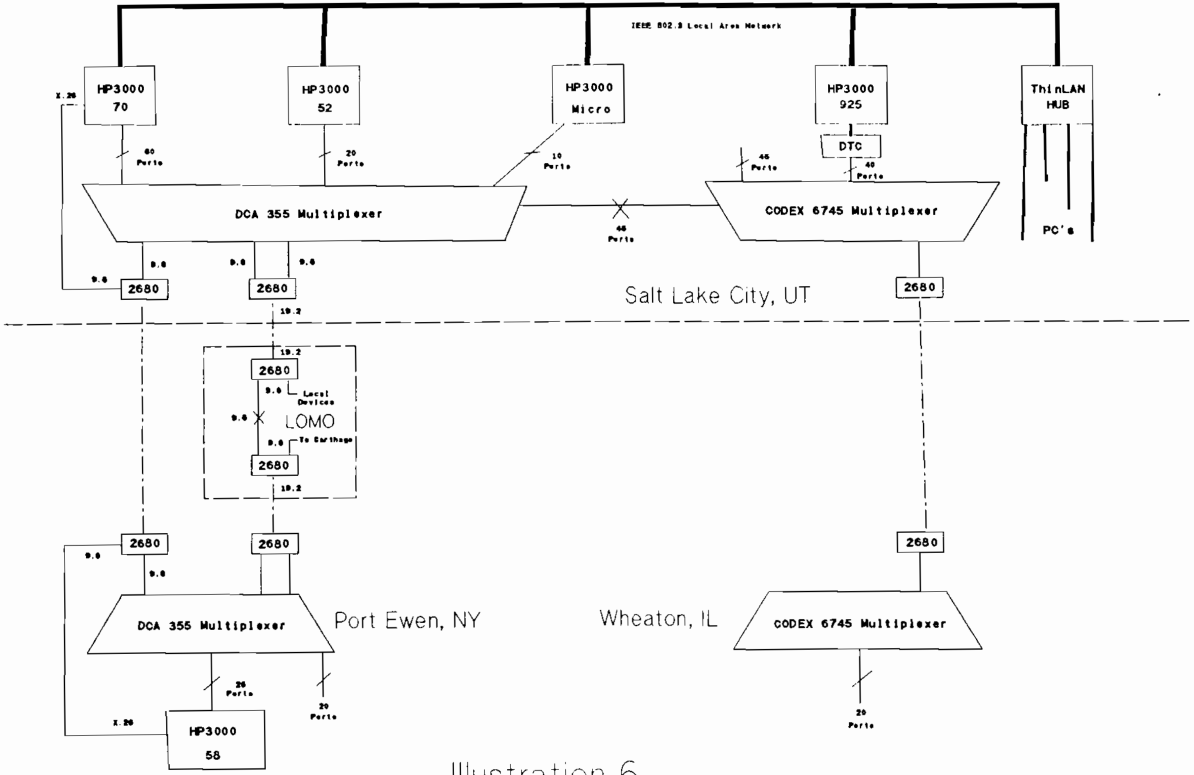


Illustration 6

04/29/88 RWS

IRECO MULTIPLEXER TEST PLAN

OBJECTIVE 1 VERIFY DCA355, DCA110 PERFORMANCE WITH HP PRODUCT SET FOR TERMINAL - CPU COMMUNICATIONS

Terminal-Oriented Product

PRIORITY:

(1) HPWORD V
(2) HPWORD 150
(3) HPDRAW
(4) ADVANCELINK 2392 (File Transfer)
(5) OM
(6) MM
(7) DELUXE VISICALC
(8) EASYCHART
(9) REMOTE SPOOLED PRINTING
(10) ADAGER MODEL II
(11) SECURITY/3000
(12) MPEX
(13) LISTKEEPER
(14) INFORM
(15) REPORT
(16) PAYROLL
(17) ACCOUNTS PAYABLE
(18) GENERAL LEDGER

TOPOLOGIES:

(1) Single DCA 355 (SLC) used as Data Switch, Terminals to S/68 and S/37
(2) Two DCA 355, Point-To-Point, Terminals to S/68
(3) DCA 110 to DCA 355, Point-To-Point, Terminals to S/68
(4) DCA 110 to DCA 355 to DCA 355, Point-To-Point, Terminals to S/68
(5) DCA 110 to DCA 355 to DCA 355, MultiDrop 110 and Point-To-Point DCA 355's, Terminals to S/68
(6) DCA 120 to DCA 355, Point-To-Point, Terminals to S/68

HARDWARE REQ'D:

DCA	1 EA	DCA 355 with	HP Option Ports (4 Ports) 2 Composite Ports
	2 EA	DCA 355	Composite Port Boards for existing DCA 355

HP	4 EA	37230A	9.6KBPS Shorthauls	(Bellevue)
	2 EA	37210T	4.8KBPS Modems	(Bellevue)
IRECO	1 EA	HP150		
		Misc Cabling		
	1 EA	2392, 2934, 2686		

ACCEPTABILITY CRITERIA:

(1) DCA as Network Vendor -
No errors in using the following products:
(1) MM
(2) PM
(3) OM
(4) PCM
(5) DESKMANAGER

(2) DCA as Conditionally Acceptable Network -
No errors in using the following products:
(1) HPWORD V
(2) HPWORD 150
(3) HPDRAW
(4) AdvanceLINK 2392 (File Transfer)
(5) Deluxe Visicalc
(6) EasyCHART
(7) HPACCESS

Errors using a product will eliminate the use of that product at a site. Therefore, IRECO will decide whether to restrict the use of the product to hard-wired terminals or eliminate DCA as network vendor.

METHODOLOGY:

(1) HPWORD V
(a) Run HPWORD - Observe and time download.
Observe LDEV
(b) Edit existing document
(c) Print existing document to:
(1) Spooled Printers (2932 & 2686)
(2) Attached Printers (2932 & 2686)
(d) Exit HPWORD
(e) LOG OFF
(f) LOG ON w/2nd terminal
(g) LOG ON w/HP150
(1) Make sure LDEV is different from step (A) above
(h) Complete step A above. Download should take less than 10 seconds

(2) HPWORD 150

(a) Transfer HPWORD 150 document from HP150 to HP3000. Observe and time file transfer

(b) Use HPWORD to convert display and print document

(c) Evaluate printed document for errors

(d) Transfer HPWORD document from HP3000 to HP150. Observe and time file transfer

(e) Use HP150 to display and print document

(f) Evaluate printed document for errors

(3) HPDRAW

(a) Use HPDRAW to display and edit an existing drawing

(b) Plot drawing using system plotter

(c) Plot drawing using eavesdrop mode

(d) Observe drawings for errors

(4) ADVANCELINK 2392

(a) Transfer ASCII file of approximately 100 records from HP150 to HP3000. Verify correct transmission

(b) Transfer binary file of approximately 1000 records from HP150 to HP3000. Verify correct transmission

(c) Transfer ASCII file of approximately 1000 record from HP3000 to HP150. Verify correct transmission

(d) Transfer binary file of approximately 1000 records from HP3000 to HP150. Verify correct transmission

(e) Compare time required for above transfers to time required when HP150 is connected directly to HP3000 @ 9600 BPS

(5) ORDER MANAGEMENT/ACCOUNT RECEIVABLE

(a) Get into the main menu and enter the OMS subsystem

(b) Add a sales order

(c) Inquire into an existing order

(d) Print order acknowledgements online

(e) Print shipping papers on a remote slave printer

(f) Enter into OMR subsystem and run a couple of reports

(g) Enter the AR subsystem and enter a daily cash entry

(h) Perform a customer inquiry

(i) Enter into the GM subsystem and enter a new customer

(j) Enter into the GMUTIL subsystem and perform at least one function
(k) Run a TREG to test the interface to MM

(6) MM
(a) Have the software start a terminal
(b) Use the "start" command to bring up a terminal
(c) Start a terminal as a logged in terminal
(d) Perform the following functions on each of the above terminals:
1. Transfer between each of the four subsystems
2. Transfer between copies of MM
3. Add a part number
4. Add a work order
5. Issue to the work order
6. Receive against a work order
7. Add a purchase order
8. Receive against the PO
9. Review activity and other things online
10. Transfer to the SAI terminal and back
11. Follow a menu tree down from top to bottom
(e) Terminate the MM session
(f) Submit a job to the SAI

(7) DELUXE VISICALC
(a) Create 50 row x 12 column spreadsheet
(b) Fill with data (50% entry, 50% calculated)
(c) Print spreadsheet to spooled and stand alone printers (use compressed format)
(d) Save spreadsheet
(e) Retrieve spreadsheet
(f) Examine all cells for correct content
(g) Compare timing with direct connected terminal @9600 BPS

(8) EASYCHART
(a) Use HPEASYCHART to display and edit an existing chart
(b) Plot chart using eavesdrop mode
(c) Observe chart for errors
(d) Plot chart to screen

(9) REMOTE SPOOLED PRINTING
(a) Create 10 copies of an approximately 132 line spoolfile with monotonically increasing line lengths

(b) Place properly configured HP4951 between system and multiplexer port

(c) Release spoolfile for printer. Observe proper printing. Look for:
   (1) Proper handling of status request (Esc ? DC1)
   (2) Proper handling of XOFF's (DC3 followed by status request properly answered followed by XON [DC1])

(d) Interrupt printer operation by
   (1) Taking printer offline, then placing online
   (2) Causing paper out and correcting paper out
   (3) Pressing reset

(e) Power off printer. Observe proper console message

(f) Verify (a) - (e) above at 1200, 2400, 4800 and 9600 BPS

Notes: Channel must be 7 bit odd parity or 8 bit no parity

Channel must have Flow Control disabled

Printer must be Type 32, SubType 14, Term-Type 19

(10) ADAGER MODEL II

(a) Change the capacities in at least three datasets of two separate databases and verify integrity

(b) Run Detpack on at least three datasets of two separate databases and verify integrity

(c) Move at least three datasets of two separate databases from one ldev to another and verify integrity

(11) SECURITY/3000

(a) Login to a user that is set up in security with a menu and verify no problems getting in

(b) Use three or four of the menu functions to verify that they work

(c) Verify that the logoff utility works, time inactivity period

(d) Verify that timeout on the DCA and on the HP3000 are in sync

(e) Stream a job using Streamx and an asked for parameter to verify that it works properly

(12) MPEX

(a) Verify that MPEX can be entered with all capabilities set properly.
(b) Use the extended LISTF command options
(c) Use the Altfile command to change the attributes of at least two files
(d) Use the extended fileset capabilities

(13) LISTKEEPER

(a) Get into Listkeeper and edit a list
(b) Create a new list from an old list
(c) Print a list to the system line printer
(d) Print list to a slaved printer

(14) INFORM

(a) Enter Inform and select into several groups
(b) Modify (or create) a report and display to terminal
(c) Print a report to the system line printer

(15) REPORT

(a) Run a compiled report and print to display
(b) Run a compiled report and print to printer

(16) PAYROLL

(a) Enter the payroll system and input some time transactions
(b) Run the time transaction edit
(c) Perform maintenance on several employee records
(d) Run a report using the report writer subsystem

(17) ACCOUNTS PAYABLE

(a) Get into the AP system and enter the Batch Input Processor. Use the functions to add, change, delete, and copy a batch
(b) Enter the Online Services. Use the functions for log procedures, vendor entry, voucher edit, master file inquiry, and control file inquiry
(c) Enter the standard jobstreams and use the functions to do control file maintenance, daily processing, and period end processing
(d) Bring up online and then bring it down

(18) GENERAL LEDGER

(a) Get into the GL system and enter the Batch Input Processor. Create and save a batch

(b) Enter the Standard Jobstreams and run a control file update and a master file update

(c) Enter Online Services and enter a journal entry

(d) Run the Post jobstream

OBJECTIVE 2 VERIFY DCA 355 PERFORMANCE WITH HP X.25 (VT & NFT) BETWEEN SLC S/68 & S/37

TOPOLOGY:

Back-to-back DCA 355's used as X.25 switches between S/68 and S/37

HARDWARE REQ'D:

DCA Same as OBJECTIVE 1 adding X.25 capability to DCA 355's

HP 4 EA 37230A shorthaul modems
2 EA 30221A (?) RS232 modem cables
X.25 LINK S/W on S/68 & S/37

IRECO Same as OBJECTIVE 1

ACCEPTABILITY CRITERIA:

No Observed Errors
No CS or DS Reported Errors
Throughput not less than 80% of computed value

METHODOLOGY:

(1) Virtual Terminal (VT) Operation

Repeat all items in OBJECTIVE 1 using Terminal/150/Printer/Plotter on S/37, applications on S/68

Note: Observed results and compare with those obtained in OBJECTIVE 1 testing

(2) Network File Transfer (NFT) Operation

(a) Create file containing 1000 100 Byte noncompressible ASCII records on S/37

(b) Transfer file from S/37 to S/52 using DSCOPY and noting time required. Verify error free transfer

(c) Compute throughput. 1 19.2KBPS and 1 9.6KBS line with traffic balancing should net approximately:

$$\frac{19,200 + 9600}{8} = 1800 \text{ CPS less overhead}$$

(d) Transfer file S/68 to S/37 using DSCOPY and noting time required. Verify error free transfer

(e) Compare throughput with C above

4GL's, COBOL and Data Communications

by
John D. Alleyn-Day
A H Computers
8210 Terrace Drive,
El Cerrito, CA 94530-3059
415-486-8202 or 415-525-5070

Introduction

One of our clients, Underground Service Alert, operates an extensive system for sending messages to hundreds of printer terminals located in the states of California and Nevada. Although not a true real-time system, it must neverless transmit messages in a timely fashion with insignificant delays. They were using on old system that had been around for about ten years and that had come to the limit of its performance because of various problems with the software. It was our task to change this software structure to allow it to handle their increased business.

The Database System

The major problem was that the old system revolved around an IMAGE database and, in particular, a detail dataset that was used to control the transmission process. This was a very complex set with six paths. These paths were present to allow access by a variety of routes but they had a very deleterious effect on performance. One of the critical keys was the "status" of the record, indicating whether or not the message was "Sent", "Waiting" or "Failed". The sending program read down this path to find messages to send. Unfortunately, changing the status, as one must for each message transmitted, involved a delete and an add.

This process was causing a major I/O bottleneck in the system and it was therefore necessary for us to find another method of controlling the transmission without such an excessive overhead. Furthermore, we wanted to introduce an additional feature, namely making the order of tranmission dependent on a "priority" field that indicated the importance of the message.

The "status" had to be a key, so that the messages waiting to be sent could be quickly found. At the same time, it had to be capable of fast updating. Several different methods were considered, including MPE flat files or MPE message files, from which the records would be deleted as they were sent. However, incorporating the priority scheme made a FIFO queue such as a message file inapproriate.

Our final solution was to use a KSAM file for our transmission queue. Making one of our keys "status" followed by "priority" allows us to find messages waiting to be sent in priority order, and to insert new messages into the queue at the point corresponding to the priority that we assigned. Furthermore, KSAM allows us to change secondary key values with comparatively little overhead, without deleting and adding the record. There is a further performance advantage of using KSAM. We usually write more than one record at a time to the file (because any one message is usually sent to several different destinations) and, if we lock around this group of updates, then the buffers are most likely to be written out to disc only at the end of all the updating. This can also save enormously on disc I/O.

The Structure of the Programs

Our main requirement is to control approximately ten outgoing modems, to have each dial a specific number, prepare a message from data on our database and then transmit it. The database then needs to be updated with the information that the message was sent.

There are several subsidiary functions that must also be performed. We have to schedule the transmissions appropriately, ensuring that the most critical messages go first and that two different modems don't try to send to the same terminal. We also need to keep track of failing terminals and not attempt to send to a terminal that is down. Furthermore we need a mechanism for changing the operating parameters "on the fly" and methods for inquiring into the performance and controlling it.

In general, the activities associated with each modem are asynchronous, in the sense that what happens on one modem has little or no connection with what is happening on another. There are also long I/O waits associated with each modem, while it is dialing or transmitting data. We cannot have a program wait for one modem to complete dialing before it continues to another, without serious impact on the total throughput of the system.

We could have written a program with privileged mode no-wait no-buf I/O. This has been done by many people in the past for special-purpose terminal-handlers. However, it would have been very complex and it would still not avoid the problem that there would be no modem port I/O while the program was updating the database. However, this situation, with multiple simultaneous, asynchronous processes is what time-sharing is all about, and is precisely what the HP3ØØØ was built to handle. So we make use of process handling and allow the MPE operating system to handle the complex scheduling of the various processes, rather than trying to do it ourselves.

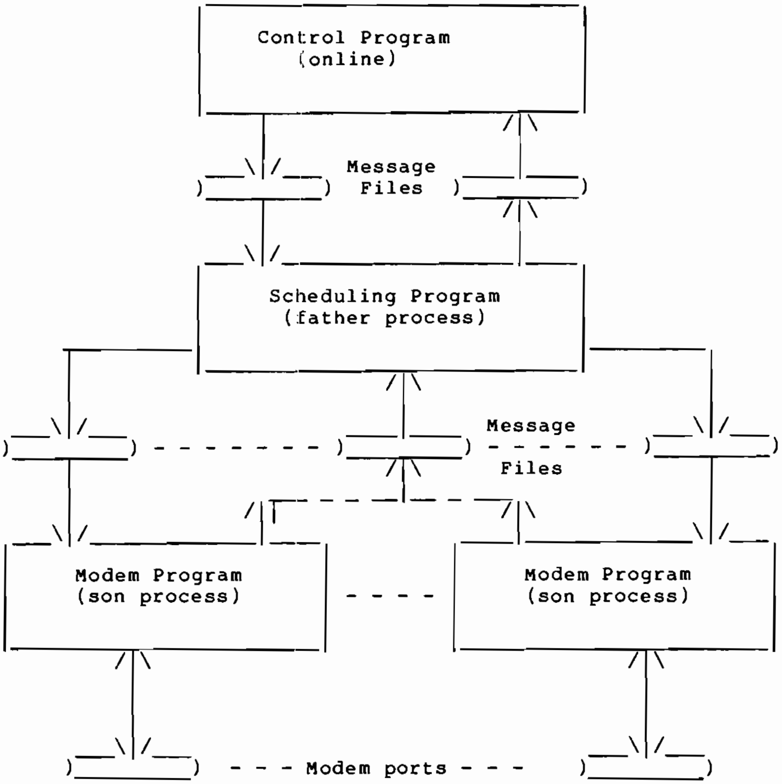


Fig. 1.   Interconection of Programs and Message Files.

The programs are set up as shown in Fig. 1. The main scheduling program is the father process,and keeps track of the messages to be sent and the availability of each line. It also updates the queue file to reflect the status of the messages. Each dial-out modem has its own son process. This process opens the port, sets up the modem, prepares messages, dials the modem, and transmits the messages. The way in which the modems are controlled was discussed in a talk that I gave last year at the Interex conference in las Vegas, called "Dialing out from the HP3000".

There is also a "control" program, run in session mode that communicates with the scheduling program via two permanent message files. This allows the supervisor to see the internal tables, to alter internal parameters and to fail and restore lines and terminals.

We make use of MPE to do our scheduling for us. The son processes spend a great deal of their time waiting on I/O, either from the port or from the message file in which commands from the control program are placed. If there is nothing to be sent, the son process is suspended on a read of the message file and hence uses virtually no computer resources at all.

The father process, on the other hand, cannot suspend in this way. Not only must it respond to replies from the sons, but also to commands from the control program. Furthermore, it has other functions that must be performed at regular intervals.

The Operation of the Programs

The scheduling program has several internal tables that are used to store vital information. The largest table is the "internal queue", which is a miniature version of the "queue" file and is replenished from this file at regular intervals. It is there for performance reasons to save on disc I/O.

The father process runs through a polling loop. It first checks the time, and if five minutes (or whatever interval has been set by the supervisor) has passed, the program reads the "queue" file to determine the messages waiting to be sent, using the key by status, priority, etc. These messages are put into the internal queue table. The program checks each modem and if it is not busy, finds the terminal with the next highest priority message and instructs the modem to send the next four messages waiting for that terminal. The program also checks the return from the son processes. Depending on the result, various actions are taken, such as updating the queue file to indicate that the message has been sent and checking the error tables to update hte error counts.

There is an internal table that keeps track of failing terminals, counting how many times a call failed to go through, or how many message file is read with a timed read, and it takes a second to time out, so the more frequent reads add additional delay time into the loop and thus make the program operate at something less than optimum speed. Its speed, however, is better than with the original strategy.

Useful Programming Techniques

Writing and debugging multiple process programs of this kind can get to be quite complex, and I developed one or two techniques that may be useful to other people attempting the same thing.

As a matter of course, I always program trace messages into my COBOL programs, usually for each entry into a section. In the past I used the "parm" to turn on and off a flag for this purpose. With son processes it gets a little complex doing it this way, so for these programs, I used a different technique, namely a JCW that is set before the programs starts. There is a different JCW for each program, so that the messages from each program can be controlled independently.

With several programs running in the same session, sorting out the messages can get quite complex. Each message, of course, identifies itself in the message with its name, and, in the case of a son, with a number to identify which one it is. This number is passed by the father in the "parm" and corresponds to that program's position in the line table. It is passed back in any reply to identify which son process the reply is coming from.

As an additional debugging aid, there is a special command in the control program that will turn on the debug flag in the father program. This can be invaluable when a problem arises in the live system after the program has run for several hours. The debug messages can be turned on "on the fly" and they prints out on the STDLIST. As an addition to this, turning on the debug flag in this way also sets the debug JCW which is checked by the jobstream. This is important as we normally use "set stdlist=delete" to get rid of the listings if all is well, but in this case we don't want that to happen.

In the tuning of the program there were changes that had to be made to several parameters such as the maximum size of various tables, that necessitated recompiling the program. This is conveniently achieved by making use of macros in the HP COBOL to assign specific values to the parameters and have them substituted in the code as it is compiled. It avoids searching through the code for each place that the number needs to be changed.

The same facility is used for error handling on file intrinsics. A standard routine is laid out at the start of the program, with the appropriate error messages substituted for each use of the routine. This saves a great deal of debugging time for error routines.

Using the 4GL system with Cobol

All the screens in this system have been implemented in a 4GL. The decision to use a 4GL was based on improved development times and the availabilty of the language for future development. Unfortunately things didn't work out quite the way we expected. Our system is somewhat out of the ordinary and many 4GL's work best on very ordinary systems. If the 4GL methods don't fit your development, the 4GL can cause more trouble than it is worth. We found that we didn't save very much development time by using a 4GL.

We carefully avoided using features in COBOL that might cause problems, such as item locking in IMAGE, and we ran into no significant problems with mismatches between COBOL and the 4GL. From this point of view, our melding of COBOL and the 4GL was very successful.

However, it appeared that the 4GL was running particularly slowly and also seriously slowing down the COBOL program. Although we were able to make some marginal improvements we finally discovered that this was because the 4GL in question has an unexpected and undocumented "feature" whereby each write is forced to disc, rather than waiting until the file is unlocked at the end of a group of writes. There is no way of overriding this "feature", and the vendor is not about to change it in any way. Combined with other inefficient ways of handling the data that the 4GL forces us into, this completely eliminates one of the major advantages of KSAM stated earlier. We are looking into methods to overcome this problem.

Summary

Mixing a 4GL with COBOL can be done without any major programming difficulties. However, inefficiencies in the 4GL will necessarily be present and can impinge on the efficient operation of the COBOL programs.

The handling of multiple outgoing telecommunication channels using father and son processes is a very effective method of control. Most of the programming is fairly straightforward, if one is familiar with using intrinsics within COBOL program programs, but tuning the program for optimum performance requires a considerable amount of experience and trial and error.

```
005900*    Define the maximum number of line-groups to be handled
006200*    Define the maximum number of lines to be handled
006300$define %lines=10#
006400
006500*    Define the maximum number of messages to be batched
006600*    into one call.
006700$define %batch=4#

007300*define the standard error routine for intrinsic calls
007400$define %filecheck=
007500     if c-c less than zero
007600       call intrinsic "FCHECK" using
007700                                 \!1\,
007800                                 file-err-code
007900       move low-values to file-err-msg
008000       call intrinsic "FERRMSG" using file-err-code,
008100                                  file-err-msg,
008200                                  file-err-length
008300       move "!2" to quit-msg
008400       perform pquit#

 ...................

027700 01  line-data.
027800     02  line-table occurs %lines times.
027800     02  line-table occurs 10 times.
027900       03 hold-index occurs %batch times    index.
027900       03 hold-index occurs 4 times    index.
028000       03  batch-pointer         pic 99.
028100       03  mt-file-num           pic s9(4) comp.

 ...................

107100     move +1 to dummy.
107200     call intrinsic "FCONTROL" using
107300                                ctlt-file-num,
107400                                \4\,
107500                                dummy.
107600     %filecheck(ctlt-file-num#,Fcontrol Error Controlt File#).
007500     if c-c less than zero
007600       call intrinsic "FCHECK" using
007700                                 \ctlt-file-num\,
007800                                 file-err-code
007900       move low-values to file-err-msg
008000       call intrinsic "FERRMSG" using file-err-code,
008100                                  file-err-msg,
008200                                  file-err-length
008300       move "Fcontrol Error Controlt File" to quit-msg
008400       perform pquit.
```

Fig 2. Example of using macros in COBOL

FOURTH GENERATION LANGUAGES AND PROCESSING EFFICIENCY

John D. Alleyn-Day
A H Computers,
8210 Terrace Drive,
El Cerrito, CA 94530-3059 USA
415-525-5070

Fourth Generation Languages have great power and can be used to write processing programs easily and quickly. However, they also have a reputation for being extremely inefficient -- a reputation which is undeserved. Many programs written in fourth generation languages are inefficient because the programmer is tempted to use programming methods without really understanding what the language is doing.

I am going to discuss a particular situation that arose using RELATE/3000. The same situation could have arisen with several diffferent languages, and with many different data situations. Since the solution would have been similar in each case, I want to share it with you.

I have been working extensively with the Sierra Club, who use RELATE/3000 together with IMAGE for a significant part of their processing. The fourth generation language is often used to access data from both RELATE/3000 files and from IMAGE databases. The Sierra Club has run into devastating problems with efficiency. Some batch programs have taken all weekend to run, just to turn out a report. In some cases, the time needed was so extreme that the jobs were aborted so that other users could get their share of computer resources! In this environment, my task was to put together several batch programs that would be producing statistical summaries of data from files with 500,000 or more records each. I was facing a serious problem of efficiency and, in working out a solution, I learned a great deal about fourth generation languages.

The major conclusion to come out of my work was that the inefficiencies were not necessarily an integral part of the fourth generation language but rather of the way in which the language was used. You see, the simplicity of the programming methods encourages programmers, myself included, to construct very inefficient programs without realising the true import of their code. I will illustrate this for you as we go along.

In order to understand the example I am going to work with, let me give you some background about the data structures that are involved. We will be looking at three files. The first is "grd" which is an IMAGE detail set. It has a unique key of "resource". The second file is "grdx" which is a Relate/3000 file and is a type of history file, with old data from "grd". The third file is called "detail" and contains details of money amounts. The "detail" fields with which we are mainly concerned are "resource", date, and amount.

We want to get the field "amount" from "detail" and collect it together on the basis of the "fiscal year" (which is a function of "date"), "class" which comes from "grd" and "prev_class" which comes from "grdx". We also need the "resource" for the time being as we want the count of the number of records in "detail" for each resource as one of our parameters. This was the first step in a series that ultimately produces a small statistical report. We shall not concern ourselves with the rest of the program as it is of no interest to our present problem. The following example of Relate/3000 code shows how we can do this in a few lines.

```
     open file grdx.data;mode=read,share
     open database gendbz.dba.para;type=image;mode=5;pass=READPW
     open set grd;database=gendbz.dba.para
     modify field grdresourc; name=resource
     open file detail.donors;mode=read,share


    select &
        detail.@, &
        fiscal= &
          $integer($substr($year($new_date(date,+92)),3,2),2), &
        memb_cnt=0, &
        rcount=0, &
        resource, &
        class=grd.grdclass, &
        prev_class=grdx.prev_class &
      by fiscal,class,prev_class,resource &
      where grdx.resource=grd.resource &
        and grdx.prev_class<>"" &
        and grdx.resource=detail.resource


    consolidate &
        fiscal:f &
        class:f &
        prev_class:f &
        amount:t &
        memb_cnt:f &
        rcount:c &
      to testdata;records=150000 &
      by fiscal,class,prev_class,resource
```

The select statement is asking RELATE/3000 to read the "amount" data from the "detail" file, pick up the "class" from the corrresponding "resource" record in the "grd" file, and the "prev_class" from the "resource" record in the "grdx" file when "prev_class" is not blank. Each of these files is at least 500,000 records, but from our familiarity with the data we know that only about 15% or so of the resources will have a non-blank "prev_class".

A program like this is very straightforward in concept and easy to put together. This is the program as I first wrote it. After it had run for the better part of a weekend and was still incomplete, we had to abort it on Monday morning to release computer resources for other users.

If I had written a COBOL program to solve this problem, it would have taken only a few hours. In frustration, I asked myself, "Is this inefficiency an inescapable problem associated with the relational database and the fourth generation language, or is there something that I can do about it?".

In working out a solution I learned that there is a cardinal rule which must be applied. KNOW WHAT YOUR FOURTH GENERATION LANGUAGE IS ACTUALLY DOING. Why does it take so long?

This is what I found out that RELATE/3000 was doing. In my program, the language will select one of the files to read sequentially (usually the shorter one) and, for each record, use the key "resource" to access the corresponding data in the other files. In my particular case, one of the RELATE/3000 files, "grdx", was the shortest, and was therefore the one that was read sequentially. For each of the 500,000 records there is likely to be two disc I/O's for "GRD": one to read the master and the second to read the detail in the IMAGE database. Additionally, for the "grdx" record, there will be about 2 or 3 disc I/O's on the "detail" file, 1 or 2 for the key and one for the data record. (This number depends on the size and the randomness of the files.) A quick calculation gives 2.5 million I/O's, and at 20 I/O's per second, these reads alone will take about 35 hours.

I must then add a few hours for sequentially reading the first RELATE file and for writing the new one. Furthermore the data must be sorted and this is another place for inefficiency. Relational database systems usually sort a file by finding an existing index that is suitable or by creating a new index. In my case the sort key is made up of data elements from different files, so Relate/3000 cannot use either of these options directly. Instead it starts by copying the data needed into a temporary file and then creating a new index for that data. Like IMAGE, Relate/3000 attempts to protect the data integrity by forcing the data to disc after each write. This will account for about 80,000 I/O's, taking about 2 to 3 hours. The keys also have to sorted and written out. The data is then read by key from this temporary file, totalled and written to the new file. It is not hard to see why this program could easily take several days to run.

This tendency to permit inefficient programming is not the characteristic exclusive to RELATE/3000. Let us look at the same basic program written in QUIZ. The following statements achieve approximately the same result (substituting IMAGE masters for the "detail" file and the "grdx" file).

```
     access grdx &
       link to resource of detail &
       link to resource of grd

     select if prev-class of grdx = " "

     define n-date numeric*6
          = ascii(date((days(d-date of detail) + 92)))
     define fiscal numeric*2 &
          = n-date[1,2]
     define memb-cnt numeric*7 = 0
     define rcount numeric*5 = 0

     sort on fiscal, class of grd, prev-class of grdx,
       resource of grdx

     report summary &
       fiscal &
       class of grd &
       prev-class of grdx &
       amount of detail subtotal &
       memb-cnt &
       rcount

     set subfile at resource name testfile
```

This code will carry out a very similar process to the one that the RELATE/3000 code performs. The estimates of the number of disc I/O's obtained for RELATE/3000 apply equally well to QUIZ. The sort considerations are somewhat different. QUIZ uses a record complex made up of the join of all the data and sorts it as one huge record. Because of the large record being sorted, it is unlikely that the method used in QUIZ will be, on average, any more efficient than the method used by RELATE/3000. In any particular case, the relative efficiency will depend on the size of the data record, the size of the key, and other factors.

So the major part of the inefficiency of the processing is not dependent on any specific fourth generation language, but rather on the processing methods that are generally encouraged by fourth generation languages. Specific methods for improving this performance depends on the particular language used, but the general approach is the same. I will illustrate my methodology using the RELATE/3000 example, leaving you to make the necessary adjustments to achieve similar results in your own language.

Now that I know why my program takes so long to run, I can set about making it run faster -- much faster. Twenty or thirty percent improvement in efficiency will not be enough; I need it to run five to ten times faster. For this phase of my work, I adopted another rule, "Use batch techniques for batch programs". This shouldn't be anything new. The "Image Handbook" in the chapter called "Throw off your Chains" contains lots of hints for handling database files in a batch environment. The fact that this is a fourth generation language rather than a third generation language shouldn't make much difference. In my original program I totally ignored the tenet "paths should be reserved for on-line users". The major reason for the poor performance is the keyed reads that are being carried out to obtain data from secondary files. How can I avoid this?

Let us go back a few years to the days of punched cards and sequential files on tape. Without keys and chains, there was no possibility of doing what I have done here. Instead, we used all kinds of processing tricks to get the answers in the most efficient way possible, usually making considerable use of the system sort, record sort breaks, and matching record keys. I can use that experience now to carry out a similar process.

I thought about how I would have written a COBOL program to do the same job. I would have read each file in turn, extracting the fields I wanted and then released the records to the sort, sorting on my key values. I would have arranged the sort so that the records from the different files were sorted together by "resource"; then I would have matched the records and created my output records, each of which would have been a composite of several of the input records. It might have been necessary to sort again before totaling, or, if the final results were sufficiently compact, I might have created a table in memory to accumulate the answers.

The answer to my problem is to do something similar here. Because of the intrinsic limitations of the various software tools, I will not be be able to achieve the same efficiency as a COBOL program, but I can approach it. I can do as much as possible with serial reads and extracted data. I must avoid using paths through datasets which, although they are excellent in on-line situations, are a disaster in batch processing. Also, to simplify things, I will deal with the files two at a time instead of all three as I might in a COBOL program. The first step is to do two extracts and a sort.

I start by reading serially through the "grdx" dataset and extracting an MPE file consisting of only those records and fields that I really need. Just as if I were writing COBOL, before starting I generate my "sort" record layout with the following piece of code.

```
     create file sample;records=Ø;fields= &
         (fiscal,i,2), &
         (class,a,2), &
         (prev_class,a,2), &
         (resource,d,8), &
         (amount,d,1Ø), &
         (memb_cnt,d,7), &
         (rcount,d,5)
```

Now I copy the data I want to my first work file with the following code.

```
open file grdx
copy rea.rcount=1 &
  to rea.data;type=mpe;structure=sample;records=300000 &
  for prev_class<>""
```

This process takes about one and a quarter hours.

Extracting IMAGE datasets is, of course, a job for SUPRTOOL. If you have IMAGE datasets this big and are serious about improving your efficiency, you must have SUPRTOOL, which can also be used for the sorting phase.

Now I switch to SUPRTOOL, extract the IMAGE dataset, appending it to the previous one, and then sort. (I make use of a field "rcount" that is not being using at the moment as a record-type indicator.) The resulting file will be sorted so that, for each "resource", there will be one record from "grd" and sometimes a preceding record from the "grdx" file. The following code achieves this.

```
base gendb
get grd
define f,1,2,integer
define p,1,2,byte
define a,1,4,double
define n,1,4,double
define g,1,4,double
extract f=0
extract grd-class
extract p="  "
extract grd-resource
extract a=0
extract n=0
extract g=2
output rea.data,append
xeq


input rea.data
key 7,4,double;19,4,double,desc
output reb.data
xeq
```

The extract took about 10 minutes and the sort about 20 minutes.

RELATE/3000, in common with most fourth generation languages, can operate on MPE files as well as on its own files. I use functions to "combine" the records and create a new file with the combined records. My first step is to run through my new file getting the "prev_class" field from the records of type 1 into the "prev_class" class field of the records of type 2. How this is done will vary considerably with the particular fourth generation language that is being used and with the format of the data being processed. In my example, I used the following code.

```
    open file sample
    open file reb.data;type=mpe;structure=sample
    let prev_class=$last(prev_class,resource)
```

The "let" statment holds the data in "prev_class" from one record to the next, resetting it to blanks when the "resource" key changes. The overall effect is to blank out "prev_class" in the records of type 1 and to move the value from that record to the records of type 2. This process takes about 30 minutes.

From here I could have proceeded in a variety of ways. One possibility is to copy the type 2 records which we want to another file and handle the "detail file" in a process similar to what I have discussed, namely extracting it to an MPE file, sorting and combining. However, I will actually get about 80,000 records from this file, and it takes only about twice as much time to link this file to the "detail" file using standard RELATE/3000 code as it does to extract the "detail" file, combine and resort. Because this was a once only report, I chose to go back to standard fourth generation language techniques as follows.

```
    create file rec.data;structure=sample;records=250000
    modify file rec.data;crashproof=no;compress=no;scan=0
    close file rec.data

    open file reb.data;type=mpe;structure=sample
    open file detail.donors;mode=read,share
    select &
        reb.resource, &
        reb.class, &
        reb.prev_class, &
        detail.@ &
      where reb.prev_class<>"" &
        and reb.resource=detail.resource
    copy &
      rec.fiscal= &
        $integer($substr($year($new_date(date,+92)),3,2),2), &
      rec.memb_cnt=1 &
      to rec.data
```

This process took about five and a half hours.

From here on the files are getting progressively smaller, and the use of the standard fourth generation language procedures will not be seriously time-consuming.

This raises a final point. A programmer must use judgement in applying the techniques I have illustrated. On small files the increased efficiency possible with my techniques will probably not repay the time you spend doing the additional analysis. However, if you run a program very frequently, analysis and reprogramming for greater efficiency may be very valuable, even if small files are involved. Some installations run small reports everyday at lunch-time in preparation for the afternoon's work. In such a case, the extra effort to increase speed can be justified.

Finally, I suggest that the Fourth Generation Language Developers consider this problem. Many customer representatives claim that their systems run batch programs. This is true -- in a way. Fourth Generation Language programs can be run in batch, but as I have demonstrated, they use on-line techniques most of the time. This should be changed. Language statements used by the fourth generation languages do not necessarily stipulate the processing actually carried out. The two examples that I used from RELATE/3000 and QUIZ now imply the use of keyed reads, leading to inefficient batch programs. Why could not a Fourth Generation Language interpret these examples as extracts, sorts, merges and record matching, similar to the processes that I actually used? A fourth generation language that could choose its processing method based on whether it was considered to be batch or on-line could achieve a substantial improvement in efficiency, and an increased market acceptance.

So far, the forth generation language vendors have not seen this as a problem that they need to address. However, there is one group that has stepped into the breach, namely Robelle. They are just bringing out an addition to SUPRTOOL called SUPRLINK, which carries out the matching of records in an efficient manner. As of this writing the program is in beta test. I have not had time to test it , but it appears to have all the cpability necessary to solve the problem that I have described here.

To sum up, if you are having problems with your fourth generation language efficiency, there are two steps to follow. First, understand exactly how your fourth generation language operates and carries out its processing. It may take quite a bit of work to get this out of your fourth generation language supplier or to do the detective work to find it out on your own. My advice is, "Be persistent". Secondly, make use of batch techniques for your batch programs and not the on-line techniques that you may be seduced into using by the your fourth generation language. And, of course, use your judgement as to when it is worth the trouble and when it is not.

Control Techniques for User's Global Resources

Kevin Darling
The Gap, Inc.
Eastern Distribution Center
3434 Mineola Pike
Erlanger, Ky. 41018

INTRODUCTION

In the past, we have designed and implemented systems to do a stream of tasks and only concerned ourselves about resource contention when it was necessary. The implication of this is that most processes were batch and processed serially. To run a job out of sequence or against a program currently running could impact the data and user access.

Well, systems have become more user oriented. The systems are now online and users have direct control of the input and can now see the result of their work online. To support this user interactivity, we have had to respond to the resource and data contention needs of these new applications.

To support the users, we have developed a software and data base solution that provides systems with a means to control the common data and resources. These global utilities or GUTIL routines as we will refer to them, have varying uses. They provide system developers a common set of rules and routines to complete the work and keep code common. These routines have also helped reduce the amount of coding in COBOL programs since the routines are simply called by the program. The catagories of routines available include:

- Global parameter control
- Work-file processing, including build definitions and file equations
- Output file equation definitions
- Error and warning message processing
- Non data base dependent routines

Following will be a description of the routines and data base to support these user controls.

GLOBAL PARAMETERS

Global parameters are data structures to hold fixed data and modifiable data. These parameters are generally accessed by multiple processes and features contained in global paramters allow them to be access globally and also allow a specific process to lock the parameter exclusively.

To support the global parameter data structure, a master data set is used in a GUTIL data base. The data structure can be represented as follows:

GLOBAL-PARMS (master):

* Parameter name - This is the key to the data set. This field that contains the actual name of the parameter programs will reference.
* Access ID - This field contains the ID for the program that currently is using the data item. This ID number is assigned when the data base is opened.
* Flag - This field is to maintain the current access status of the parameter. The value determines whether there is open access, read access only or locked with no access by any other process.
* Type - This field is to determine the format the data is returned to the program. If the type is 0, it returns a single element integer. If the type is a 1, it returns a character string of up to eight characters. If the type is a 2, it returns a 2 element double integer.
* Length - The definition of the length of the data field to be returned.
* Binary data - This data item is a 2 element data area where the integer data is stored.
* ASCII data - This data item is a defined character string for storage of the character data.
* Binary min and max - These values control the range the parameter falls in. If the value becomes greater than the maximum, the routines automatically wrap the value around to the minimum.

Other data items maintained in the data structure but not critical to the actual data processing include a time stamp for the last modification and the modifier name for the process that last modified the data.

The uses for the global parameter data structure include:

* Nearest printer to an LDEV - This structure allows you to define a terminal parameter that stores the nearest printer to that terminal. Programs could dynamically access this information and route the output being requested by the user to the printer. If a change is necessary while a user is working, like to route to a different printer, the parameter could be modified and the next request for a report would reference the new printer location.

* Informative parameters to control programs - These parameters include concepts for location identification for reports and program control. The parameters could also be used to supply data for controlling programs. This data could be constant or could be very dynamic and changed periodically.
* File building key information - This structure enables you to define parameters used to build files. In our use of this concept, we have defined integer data types that get incremented from some start value through and ending value to create a unique key for data work files. If the number series reaches the maximum number, the routines automatically wrap the value around to the minimum value. Using this and the file building features we will discuss later, we create files for use by processes and maintain the key series separately for each major process.
* Dynamic data parameters - This data structure is used to hold data that could be and usually is modified frequently by many processes. Such uses include data retention values, time stamps, and parameters that provide unique key entries for data bases.

The routines that are in place to support global parameters include:

* GETGPARM - This routine gets the global parameter data and based on the type, returns the data to the calling program.
* SETGPARM - This routine sets the data value based on the type to the value that is passed to the routine.
* INCGPARM - This function increments by one the data value in the parameter. The parameter needs to be a type 0.
* DECGPARM - This function decrements by one the data value in the parameter. The parameter needs to be a type 0.
* GETGPARMSTAT - This routine returns the current status of the global parameter. This status is the flag value kept in the data base for the global parameter.

WORK-FILES

Work-files are the most complex component of the GUTIL routines system. The software supports the automatic creation, maintenance, file equation definition, and purging of these files. This is done in such a manor that the system developer only need define the characteristics and file equation for the file one time and the system is intellegent enough to resolve the processing for the file.

Work-file equations are generally pre-defined file equations that get established by the procedure for which they are established. These file equations allow some use of variable definition that the routines are intelligent enough to resolve.

The work-files themselves are files which when built can be used for any purpose. The files can also be created with user labels. The user label area is especially good for maintaining processing, restart, and data that is common to all data in the file. In the case of the restart information, the programs could use this concept to be able to help control the restart of programs in event of a program failure or graceful termination. The files themselves with the user labels and the data base entries also provides a secure version control on the files.

To support the work-file concept, there are four master data sets and three detail data sets. The data sets and their data include:

CATALOGS (master):

* Catalog - This is the key to the data set and is a number that represents a grouping of files. These files generally have a root that is used to store and reference the file.
* File root map - This is the actual map that is used to generate the root for the file. There could be GUTIL variables included in the definition that the serving routines resolve in order to complete the file build.
* Description - This is simply a definition, for reference, of the defined purpose of the catalog.

FILE-CODES (master):

* File code - This is the number that is used by the build command's CODE= parameter. It clearly defines the file type for use by other routines but also allows programmers and support personnel to easily determine the use of a file.
* Description - This is simply a definition, for reference, of the defined purpose of the file code.

FILE-ROOT (automatic master):

* File root - This is the actual root that gets assigned to files when the file is created. There is one entry per unique root.

PROCEDURE (master):

* Procedure - This typically is a program or procedure name. It is used as a global reference to data so routines can do things for a specific program or routine.
* Description - This is simply a definition, for reference, of the program or procedure and basically what it does.

WORK-FILE-BUILD (detail):

* Catalog - This is used to define the catagory the file was created under. It allows a path into the data to display all files that are referenced by such a grouping. It is a key to the data set.

- File code - This is used to define the specific file code class the file belongs. It typically is a finer resolution as to getting specific files needed. It is a key to the data set.
- File name map - This is the definition of how the name is to be constructed if it contains variables to be resolved.
- Lock word - This is the lock word that is assigned to the file when it is built.
- Group name - This is the group in which the file is to be created.
- Record size - The length of the data record.
- Label length - The number of words that are required for the label information the file may use.
- Various MPE file create information fields including blocking factor, extents, record limit, and KSAM key information if the file is a KSAM file.

WORK-FILE-EQUATIONS (detail):

- Catalog - This is the global catagory for the file equation grouping. It is a key to the data set.
- Procedure - The program or procedure name that is used to index and reference a specific group of file equations. It is a key to the data set.
- File code - This is the code number assigned to a more specific group of files. It is a key to the data set.
- File equation - This is the actual file equation the routines use to be set up for the calling program.

WORK-FILES (detail):

- Catalog - This is the global catagory for the file grouping. It is a key to the data set.
- File code - This is the code number assigned to a more specific group of files. It is a key to the data set.
- File root - This is the actual root or name of the file as it appears on the system. It is a key to the data set.
- Date/Time created - This represents the date and time the file was created. This information is also kept in the label and is used to help maintain version control.
- File name - This is the full name of the file including the group in which the file resides.
- Label length - This represents the length of the label as it is defined.

The use of work-files can be extended from their basic use of storing user data. One extention which uses the concept of user labels includes storing common data used in processing in the label. This could reduce the amount of data that has to be stored in each record. A second application of work-files is to use the user label for storing check points for processing. The programs would update fields in the label at critical points of the processing. If the program would fail, a simple clean-up of the data back to a check point could be done and then the program would be restarted. Upon restart, the program would first determine if it was already processing the data and where it had left off. Processing would then be set to begin at the point of interruption and the program would continue until normal completion. Use of user labels does require some tools to be written to maintain the data should modification be necessary. Such utilities could and should be user developed to meet the need of the particular application data file. But, tools like DISKED5 can be used to do the simple data editting tasks.

Most of the applications we have developed rely heavily on being able to generate unique files for processing through the day. We also use work-files that are created once and that are used over. Such applications include data base clean-up programs. The file has restart information in the label that initially starts as ready to process. The program uses the file to store the keys found in searching that are candidates for deletion. Upon completing the search the phase is set to deleting and the record number is set to the top of the file in the label. The program begins purging and updates each time the pointer. When completed with that phase, the phase is updated to the next search to be done or set to program completed for the next run. In this example, if the program fails during the search phase, it simply redoes the search when restarted. If the delete function has begun, the program gets the record number it last purged and moves on to the next record and continues processing.

To support the work-files philosophy, several routines are available. They include:

* BUILDWFILE - This routine uses the definitions defined in the work file build data set to actually build the file. If there are variables or parameters that need resolution for completing the actual build, this routine resolves them. Typically, such items that need resolution include getting a global parameter to fill in a particular value.
* DELETEWFILE - This routine deletes the work-file entry from the data base but does not purge the data file. This allows the entries that need to be searched for a program be reduced while still maintaining the data for backup.
* ENTERWFILE - This routine reads the label information of the work-file and then recreates the work-files data set entry. This is required so that the automatic version control built into the system will allow use of the data file.
* EQUATEWFILE - This routine uses the data passed to establish the

defined file equations for the specific file. It uses the definition in the work-file equations data set. And like the build function, will resolve any parameterized variables.

- * GETWFILEINFO - This routines retrieves the label information and returns the data to the calling program.
- * PURGEWFILE - This routine actually purges the data file and data base entry for the data set.
- * SECUREWFILE - This routine logically locks the data file in the work files data set to control the access of the data file.
- * SETWFILEINFO - This routine writes the label information back to the data file.
- * STATUSWFILE - This routine returns the status of the data file, including the security and logical lock information.

OUTPUT FILE EQUATIONS

Output file equation control is available to control the set-up and issuing of file equations for a program of process. Before a program begins processing, it establishes the necessary file equations for referencing output devices. During the processing if there are changes or new equations that need to be set, the program can reexecute the file equations store in the data base and continue processing.

Data base support for the routine include two master and one detail data sets. One master already discussed for work-files is Procedures. The others used to support output file equations include:

FILE-FORMAL (master):

- * File formal - This item is the eight character reference for the file equation which also can be seen in a SHOWOUT FNAME field for the output queue.
- * Description - This item describes the file formal for reference and documentation purposes.

PRINT-FILE-EQUATION (detail):

- * Procedure - This is a key entry from the master data set. It is used to be able to define all file equations for a specific program or procedure.
- * File formal - This is a key entry from the master data set. It is used to be able to define a specific file equation for a program and not the entire set available.
- * File equation - This field contains the actual file equation the routines use to set up the equations for the programs.

Output file equation control in a controlled environment has two purposes. The first is upon initial start-up of the program, all necessary file equations are defined for the output needs of the program. Much like work-files, the equations can be parameterized such that some resolution of the equation may be necessary. A second use that relies even more heavily on the parameterization available is the use of one common file equation with the device ID being undefined. This value could then be resolved by either using a global parmeter that has been updated to reflect the output location or by using the global parameter NEAREST-PDEV-TO-###, where ### is the terminal requesting the output, to define the final destination.

The single routine used to support this feature is:

- EQUATEPFILE - This routine uses the procedure/file-formal to define the file equations requested. It resolves the parameterization in the file equation based on the same rules as work-file equations are defined.

WARNING AND ERROR MESSAGE PROCESSING

Warning and error message processing is available for the special processing of messages back to users, operators, and programmers. This facility enhances the error messages the HP provides to better define errors and display the data to various places in specified formats. These routines use highly parameterized data to resolve the processing of information passed by programs. Warnings are typically used for informative purposes while errors usually will be followed by the program terminating in some error state.

The data base support for message processing includes again the Procedures data set already defined and also the following:

MESSAGES (detail):

- Procedure - This is a key to this data set from the master. It is used to be able to access messages for a specific program and allow for the same error number to be stored for different programs.
- Message - This is a sorted data item which is the actual message number.
- Type - This defines whether the message is a warning or an error and what level of warning or error it is.
- Flags - This item defines information concerning the display and responce needed for a message, if any. Examples of this include definition of the display locations (eg. console, log file, or $STDLIST) and whether a console reply is necessary or not.
- Message parameters - These define up to four data parameters and the format in which to be displayed. These parameters typically contain data that should be displayed for better error resolution.
- Message text - This defines the message text, including any parameters that need to be resolved. Also included are references to the message parameters which are passed by the program.

The key use here is to consolidate the messages for systems into one place for easy maintenance. It also provides programs with functionality that can be used to better debug program errors and aborts. By storing the parameters and messages in a data base, the form of the message can be easily modified even while programs are running. This should be especially handy if a program begins to issue messages to the console which are simply informative and the operations person wishes that the messages would stop displaying. The message could be modified to be rerouted to $STDLIST or modified to better display the error data if it appears difficult to understand.

Routines available for message include:

- DUMBMSG - This routine processes messages based on information in the data base. This message can be sent to more than one location and may request operator reply. It can automatically cause an abort or simply log the message to a file and continue. If the message causes an abort, 104 words of the stack, starting with the address of the first parameter, is displayed.
- LOGABORT - This routine logs an entry to the abort log file and terminates the porgram.
- LOGSTART - This routine creates an entry in the process log file indicating that a program has started.
- LOGSTOP - This routine creates an entry in the process log file that a program has stopped.
- SMARTMSG - This routine is the same as DUMBMSG except that it checks the last message before returning the new message. If the new message is more serious than the old, it returns the message; otherwise, it does not.

SPECIAL ROUTINES FOR GUTIL

Any program that uses the routines described above that need access to the data base need to execute two other routines. The first is INITGUTIL. This routine is designed to open the data base and then initialize the data and common areas associated with using the routines. Second is RELEASEGUTIL. This routine closes the data base and releases any special resources consumed by the GUTIL routines.

NON DATA BASE DEPENDENT ROUTINES

GUTIL also includes routines that are not data base dependent. These routines are typically are for specific tasks or data manipulation. Examples of the routines included are:

- CAPCHECK - This routine uses the Local Attribute available with MPE as well as a security template to determine whether the user has the authority to use a particular function.
- STARTJCW - This routine starts a process that displays a reply message to the console and then processes the reply back for the calling program. The JCW needs to be checked by the program to determine what should be done.
- TIMESTAMP - This routine returns the date and time in a two element double integer. This is used to track modification in the GUTIL routines and it can be used by programs to apply a time stamp to user data to know when a particular update occurred.
- CDIGIT - This routine calculates check digits based on the check digit calculations available in the routine.

CONCLUSION

These routines have been used in our shop for over four years now for a variety of uses. These routines have provided our staff with common routines and data structures for the development of user systems. These routines also provide the control and checks and balances necessary for the successful operation of the systems.

ACKNOWLEDGEMENTS

I would like to acknowledge Mr. Chris Hagood for writing a significant portion of the routines discussed. And also to his documentation of the system which made it easier to use and manage.

I would also like to acknowledge our staff for helping to continue to develop new routines and enhance old ones to make the system effective for getting the job done.

Integrated Information Management - Get The Connection?

Jim O'Brien
O'Brien Downs Systems, Inc.
P.O. Box 429369
Cincinnati, Ohio 45242

INTRODUCTION

Integrated Information Management ("IIM") is a concept that has been derived from the sharing of information. Information sharing that is essential to operating a modern business enterprise. Traditionally, software developers have not provided the ability to share information. Standard access methods to find the necessary information typically require the user to utilize different programs and inquiry screens to determine relationships that must be known to serve a particular request.

For example, a customer service representative may need to know expected delivery dates of a selected inventory item for a particular customer. This request would be difficult to accomplish with non-IIM software systems. The time element involved can be a serious hindrance to answering a customer request. The lack of integration with the corresponding quick access may potentially lead to customer problems such as lost orders, irritated customers and generally unacceptable customer service levels.

One important question to ask is why were systems written in the past that are functional for the areas being accessed such as customer order entry, purchase order entry, inventory control, etc., but are not integrated to each other? The answer to this question is fundamental to this dissertation on IIM. Applications were written for a specific task, for a specific purpose or for a particular user or group of users. No thought was given to integration of the various applications within the overall system being used.

Another question to ask is why haven't IIM systems been developed in the past? The MIS staffs of organizations have been viewed as a service department. Their main responsibility was to "take care of the computer system" and handle the computer needs of the end user. Therefore, since little evidence of IIM is seen, it may be because the user has never asked for it or wasn't aware that IIM can be a reality. Unless management permits the MIS department to take an active role in offering solutions to business problems or to give some incentive for offering innovation, IIM systems of the future will never be possible.

A logical reason for lack of IIM may be found by analyzing the history of computer software systems. The history of computer software systems can be demonstrated in three phases, as follows:

PHASE I:

BATCH-ORIENTED PROCESSING:

Batch-oriented processing was used for very specific functions. These were without any on-line, real-time capabilities or any thought of integration. Software system developers were extremely limited due to the hardware and software technology in existence at that time.

PHASE II:

ON-LINE PROCESSING:

Current software systems utilize on-line processing, but still have roots in the design from the batch-oriented processing. Separate systems are written for specific functions and information relationships are only addressed where necessary to run these specific applications. This often results in a narrow focus with built-in design limitations that may make the IIM concept difficult to utilize around these applications.

PHASE III:

INTEGRATED INFORMATION MANAGEMENT:

Analysis of existing IIM based software supports the statement that it provides organizations the ability to use their information resources to their best competitive advantage.

A symptom of the need of IIM is "I'll have to talk to John over in our purchasing department to find out about delivery on that inventory item" or "I'll get back to you later when I find out all the pertinent facts." How many times have we all heard those types of things being said, been put on hold, transferred to another department or told we would be called back with the information we requested? IIM can put a stop to these issues and improve customer service levels immediately. Frustration and lack of responsiveness to user needs have increased over time and will only get worse if IIM is not employed as soon as possible within the company organization.

The lack of system integration may have been acceptable in the past, but if we are to be competitive in today's marketplace, it is essential that computer solutions employ the new conceptual framework of IIM. IIM will allow us to move forward with our system development and to meet the challenge for integrated information access. We will be able to use the valuable information which is being collected at all levels within the organization to our fullest advantage.

IIM will be made possible only when information management has taken place and the organizational barriers have been removed. When IIM techniques are employed within the organization and the various company departments are sharing information, dramatic changes may occur. Examples of impact areas that can benefit from IIM are customer service, responsiveness to problems, inventory turns and better purchasing.

There are four major areas for discussion in employing IIM within an organization:

I. SOFTWARE:

Software designers must develop systems that integrate the available information within the organization and also be very user friendly. If the newly developed IIM systems are not user friendly then they will not gain the wide acceptance that must prevail if IIM is to be used to its full potential. IIM systems must be developed by experienced, proven system professionals. Systems should be designed in a cooperative effort between management, department heads, individual users and the MIS staff using the following steps:

A. DEFINITION OF REQUIREMENTS:

This is a difficult step as most systems were developed around a particular application. Brainstorming sessions must be held to formulate which kinds of IIM tools can be developed to use company information to the fullest benefit. The end users and department personnel need to discuss their individual needs as well as the company's needs and expectations. Everyone must close their mind to pre-conceived notions of how systems are developed. User input in areas that may not come under their job description must be considered in overall requirements. Management should encourage them to step outside their normal functions because the best definition of need may come from those that do the majority of the daily workflow. Inter-departmental and inter-user relationships should be discussed and documented to assist in requirement definitions. Company management must be made fully aware of the urgency of the IIM need so they can utilize it to become more competitive in the marketplace.

B. DOCUMENTATION OF REQUIREMENTS:

Formal documentation procedures should be adopted and established. This may include, but is not limited to, write-ups of job descriptions, organizational charts, management/user objectives, narratives, flowcharts, screen layouts and any other information which may assist in the overall system design.

One important aspect to consider is in the area of user and data access security. This should not be taken lightly as once IIM techniques are employed, access to company information becomes easier. Documentation of security requirements, with corresponding user and management approvals, must be obtained before final system specifications are developed.

C. SYSTEM DESIGN:

System design becomes much more complex as more integration of information is required. Structured system design is highly recommended in order that the integrated techniques and inter-relationships of the data can be fully understood by all system and user personnel.

D. PROGRAMMING:

Access and retrieval speed is mandatory to be as fast as possible under certain hardware/software constraints. Special care should be taken toward selection of the programming tools and languages used so speed can be maximized. Programmers and system designers should be constantly searching and/or developing new tools to provide quicker, easier and more user-friendly access to the data within the systems. Structured coding and programming standards are required when IIM systems are to be programmed. This will make program de-bugging and future program maintenance easier and therefore result in better programs that will run in accordance with specifications. On-line help prompts can be incorporated into the programs to assist users in how to operate the software. Screen printing to a selected printer is also helpful since this provides users with a mechanism to print out pertinent information that they do not have to write down. They will be able to highlight information and pass it on to the proper person for follow-up. This should result in quicker follow-up on problems.

E. TESTING AND DEBUGGING:

Various users can alpha and beta test the software programs as they are developed to ensure that they have been programmed according to the specifications and requirements as presented in A) and B) above. This will prevent some of the lack of integrated system design that has occurred in the past. As users start working with IIM programs, they will continue to modify and enhance existing programs. New programs may be developed as they become more familiar with the endless possibilities now available to them. An information revolution may take place as we make great progress in the IIM area.

F. IMPLEMENTATION AND TRAINING:

Realistic time tables should be set for training and implementing the IIM system. Many good project scheduling and management software tools exist out in the market today that can be used in this area. Training should take on a greater emphasis than it has in the past so that users can get increased benefits earlier in the implementation cycle. User manuals should be developed that demonstrate all the features that are available to the user. Full examples should be provided so the user can follow along with his/her own screen for easy assimilation to a particular environment.

II. DATA:

Policies must be developed for data standards so that information is easily transportable from system to system. This can be on a single CPU or may involve a multitude of systems. Data standards also need to be established for non-EDP data as well. The standardization of the data may also encompass using corporate databases to collect and group data for all entities or a single entity. Access may even be required between the home office and remote sites or vice versa. In many companies, various hardware/software environments exist, which require data standards to be enacted so that all the required information can be accessed from anywhere within the organization.

III. PROCEDURES:

Procedures and policies must be developed for integrated system development. These should be adopted companywide aligning integrated system development strategies.

IV. MANAGEMENT SUPPORT:

Full management support is required for success in order that maximum advantage be taken of the company's valuable information resources.

IIM may require some changes in management philosophies. All pertinent information must be related and integrated. This means that inter-departmental barriers must be broken apart and the information systems linked together. The ultimate purpose of IIM is to make available information that has been created by departments or users that specialize in creating and using that data to all users. Management must adopt a policy of integrating data and systems where possible to bring the entire resources of the company together, much like a conductor who integrates the various instruments together in the symphony.

Management will have to allocate funds for IIM projects in order to be able to reap the tremendous benefits available. Management will have to begin to measure the benefits provided and will see astounding increases in user satisfaction, customer service levels and the cooperation between employees. Management must support IIM fully to maximize the benefits that this kind of technology can provide.

CONCLUSION

IIM poses many challenges to management and system professionals that need to be addressed. When the IIM technology is used in "real-life" situations the results are amazing. We have seen and documented significant productivity increases in companies where IIM has been implemented. IIM systems have been developed which work with existing software technology. IIM is not just a vision, but a reality.

Capacity Planning: Getting Started

Charles Rice
University Systems
The Ohio State University
1121 Kinnear Road
Columbus, Ohio 43212

Introduction

Capacity planning can be defined as planning the future use of your computing resources by what has taken place in the past and what is currently being planned for the future.

There is a proverb that says "any enterprise is built by wise planning, becomes strong through common sense, and profits wonderfully by keeping abreast of the facts." This proverb sums up what capacity planning is all about.

Planning the use of your computing resources is a wise thing to do. One of Murphy's laws states that the demand for your computing resources will always grow to exceed your computing resources. The purpose of a capacity plan is to accommodate the demand for computing resources before those demands exceed the resources available.

Capacity planning requires the use of common sense. You can almost always add another system to your computer, but common sense says "what is the affect of the new system on the other systems?" The new online system could possibly affect the response time of other online systems.

Above all, capacity planning requires you to keep abreast of the facts and rumors. A capacity plan needs to keep track of what happened in the past and what is being planned for the future. Are future versions of a program going to use more resources? Are there plans to add new systems to the current load? As time goes by the future becomes more clear and adjustments are made to previous plans.

Getting Started

The first step to capacity planning is to identify the resource that will limit your capacity. What resource is keeping you from adding more systems? Is the CPU busy 100% or maybe the CPU is paused for disc 50% of the time. As a general rule the resource that limits your productivity is the resource that you want to concentrate most of your effort.

Before MPE started to cache disk IOs in main memory, the number of disk IOs was often the resource that limited activity on our HP. With disk caching though, CPU has become the limiting factor. Therefore at University Systems, we concentrate most of our effort on tracking and predicting CPU usage.

Even though our Capacity Planning Report concentrates on CPU, for historical purposes we track disc IOs, memory activity and global response time. These resources can indicate problems that would not be noticed by tracking just CPU. For example if memory activity seems high, you have a historical record of memory activity to use as a yardstick.

For all of the resources we track we have set a threshold value that we do not want to cross. For instance 85% CPU busy is the typical threshold value for CPU in the industry. If the CPU is busy 85% of the time then you are approaching the capacity of your CPU. Some thresholds that we set are arbitrary i.e. a point where we think we will start to have problems although we have no evidence that would indicate any problems.

There are many tools on the market today that can assist you with capacity planning. The first capacity planning report produced at University Systems' used some tools from the Contributed Library. Before we produced the next report MPE V was installed and the tools we were using were not compatible with the new version of MPE. It is best to use tools that are either supported or supportable i.e. you have the source code and knowledge to support the tools.

At University Systems we use HPTREND from Hewlett Packard and SYSPLAN from Carolian Systems. HPTREND has been very good for tracking resource usage by account and CPU by activity e.g. CPU busy, memory management, paused for disk etc. SYSPLAN has been excellent for identifying CPU activity by process and/or user. SYSPLAN, by allowing us to get at the raw data, offers us a lot of versatility despite its short comings, which I will identify later.

Organizing the Report

At University Systems we produce a Capacity Planning Report quarterly. What follows will be a summary of how our report is organized and then a description of all the sections.

1. Index.
2. Summary.
3. Definition of Business Elements.
4. CPU Graphs:
   a.) Average CPU used during 24 hour weekday.
   b.) Average CPU used during weekend.
   c.) Prime Time CPU Use by Business Element During Quarter.
   d.) Peak Time CPU Use by Business Element During Quarter (see page 7.)
   e.) Table of Projected Growth by Business Element for prime and peak time (see page 8.)
   f.) Graph of Projected Growth by prime and peak time (see page 9.)
5. Graph of Disk IOs during prime and peak time.
6. Graph of Free Disc Space.

7. Miscellaneous Information.
8. Actions taken to improve performance.
9. Recommendations.
10. Glossary.

Report Summary

Most people who read this report will have their own areas of interest. For example the Director of Operations might be more interested in CPU usage on third shift while the Director of Development will be interested in response time the programmers are receiving. The purpose of the summary though, is to direct the readers attention to the areas that need attention. This is where you point out when the current CPU will run out of capacity.

Business Elements

At University Systems we wanted to know what processes were using the CPU. So we divided all processes into eight different categories that we call Business Elements. The eight Business Elements are:

1. Editing Programs (Editor, TDP, SPEEDIT etc.)
2. HPDESK.
3. Office Automation (HPWORD, GRAPHICS etc.)
4. Data Communications (MRJE, DS, NS, IMF etc.)
5. Utilities (MPE subsystems, LIB, TECH etc.)
6. Production (Income producing accounts.)
7. Other (Programs not included in other filters.)
8. Unknown (CPU used but not accounted for by any business element. We have assumed that this is operating system overhead.)

CPU Graphs

Although we are mostly concerned with prime time (8:00 AM to 6:00 PM Monday thru Friday) we do include a graph of CPU usage during the week nights and weekend. This serves two purposes, it gives management an idea of what is happening after hours and records this information for historical purposes. There might come a time when this information will become useful.

We produce two pie charts for prime and peak time that display average CPU used by business element during the quarter. By adding a slice of pie labeled IDLE CPU we force the total of all the business elements to add up to 100% i.e. I00 minus the sum of all business elements equals IDLE CPU.

The table of projected growth by business element is a product of two tables. The first table consists of the percentage that each business element is projected to grow by month (see TABLE ONE.) The second table consists of the actual and predicted percentage of CPU consumed by each business element (see TABLE TWO.) The last month of actual data is multiplied by the percentage that the business element is expected to grow in the next month. For example if HPDESK is expected to grow 1% in the months of May thru September. The last month of actual data shows that HPDESK used up 15.4% of the CPU. This number (15.4%) is multiplied by 1% to obtain the projected growth of HPDESK for the month of May.

TABLE ONE

	3/88	4/88	5/88	6/88	7/88	8/88	9/88
HPDESK			1%	1%	1%	-25%	1%
EDITORS			2%	2%	2%	2%	2%

TABLE TWO

	3/88	4/88	5/88	6/88	7/88	8/88	9/88
HPDESK	14.0%	15.4%	15.5%	15.7%	15.8%	11.8%	11.9%
EDITORS	15.2%	14.2%	14.5%	14.8%	15.1%	15.4%	15.7%

The table of projected growth by business element is detailed by month and should cover as many months as you think your current CPU will last. This table should reflect your judgment of the expected growth. For example if you know that the performance release of HPDESK is going to be installed in August, then theoretically the amount of CPU used by HPDESK during that month should decline. So the expected growth for that month would be a negative number.

The Graph of Projected Growth is produced from the table of projected growth by business elements. The sum of all business elements are added together and plotted by month in a graph. This is probably the most important graph of the Capacity Planning Report. An example of the table of projected growth and graph of projected growth is on page 8 and 9 of this paper.

You can use this graph to show your track record of predictions versus actual values recorded. By doing this you can determine if the predictions for growth need to be revised. Your first few reports will probably not be very accurate but with experience, you will get a better feel for predicting future growth.

The Capacity Planning Team for our IBM mainframe has found that the margin of error on all the business elements usually averages out to be fairly small. That is they do not have good track record predicting individual business elements, but the margin of error on all business elements averaged together usually comes out to be very small.

SYSPLAN does have a TREND command that does a linear projection of resource usage, but this command has a major flaw. When the TREND command is used, its projection is based on all hours and days of the month with no exceptions. We use our HP very heavily during prime time, but at night and on the weekends, it is just about an idle CPU. CPU usage during prime time is 70% to 80%, however using the TREND command shows the CPU usage around 50%. I don't know about you, but I would have a hard time justifying the purchase of a new CPU based on the graph produced by the TREND command.

Ideally, the TREND command should let you input what the growth will be from month to month and let you eliminate exceptional days. You would be surprised what a few holidays would do to a linear projection.

DISK IOs

A graph of disk IOs is also included in the report. This is for historical purposes. Since we are not ready to upgrade to a HPPA processor, we needed to extend the life of our current CPU. To do this we are replacing all of the 7933XP disc drives with 7937XP disc drives. Then we will cache IOs at the disc level instead of in main memory. The theory being that the 7937XP disc drives can cache IOs ALMOST as effectively as main memory can. We anticipate freeing up 5% to 10% or our CPU but at the risk of becoming IO bound. Therefore we want to track our average number of disk IOs during the prime time.

Disk Space

Freespace is also included in our capacity planning report. We order another disc drive when 75% of our total disc capacity is full. In general it is hard to offer more services or add new systems when disc space starts to get tight. A lack of disc space can become a performance and/or service inhibitor.

Miscellaneous

The Miscellaneous Information is a good place to put trivial facts that have potential of impressing somebody. For instance this section is a good place to put how many sessions and jobs that occurred on the HP on a average month.

University Systems as a general rule doesn't charge anything for the use of the HP. However we do have a charge back program. We are going to use this program to produce bills to send to our customers with a note explaining that this is the amount of free services we provided them with last month. This produces good will among our customers and conditions them to the day when we might have to start charging for use of the HP. The dollar amount that the HP could bring in if we so chose is good miscellaneous information!

Actions Taken to Improve Performance

You always want to include in your capacity planning report what you did over the last period to improve performance. Sometimes it is hard for upper management to see the monetary value of a system programmer because a system programmer doesn't directly bring in any income. This is a section to let readers of this report know what you do and how valuable you are.

Recommendations

This section is used to make recommendations to improve performance. The recommendations may be as simple as optimizing disk IOs over LDEVs to buying a larger CPU. The idea is to be planning as far into the future as possible. You do not want to wake up one morning and find out you need new CPU. On the other hand you do not want to buy a larger CPU sooner than required.

Conclusion

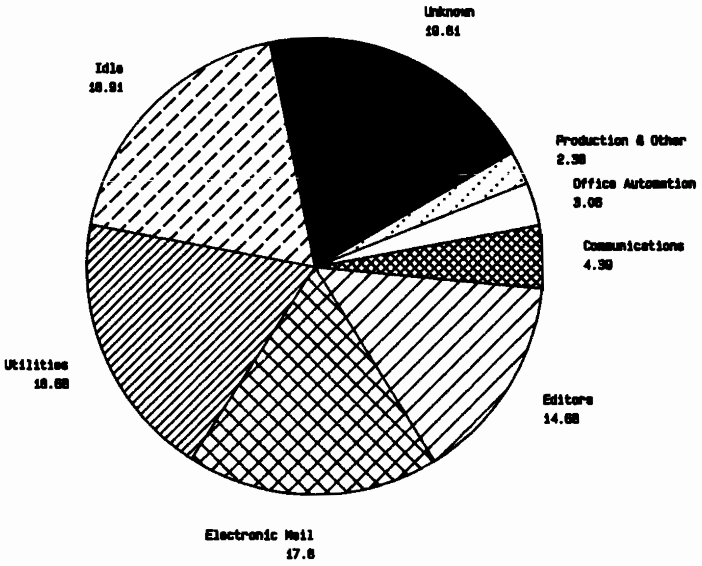
There are several mistakes that can be made in producing the capacity planning report. One mistake is for the systems programming group to become consumed with producing the capacity planning report and have no time for any other activities.

Another mistake is to not expect any surprises. A capacity planning report can only guess what will happen in the future. No matter how good one is at predicting the future, there will come a time when the prediction will miss by a long shot. Usually the cause of the inaccuracy is from underestimating the amount of CPU a system will consume. Be prepared for the worst.

Every computer center should have some idea of its current and future computing load. A capacity planning report can provide management with the necessary information to formulate a plan to accommodate or deny future demands for computing power. The hard part about producing a capacity planning report is getting started. Once started though you will find the report evolving into very useful tool.

UNIVERSITY SYSTEMS HP3000 SERIES 70 US3/RED

Average CPU Used by Business Element during Peak Hour



October 1987 thru December 1987
10:00 AM to 11:00 AM

PROJECTED PEAK TIME CPU FALL QUARTER '87

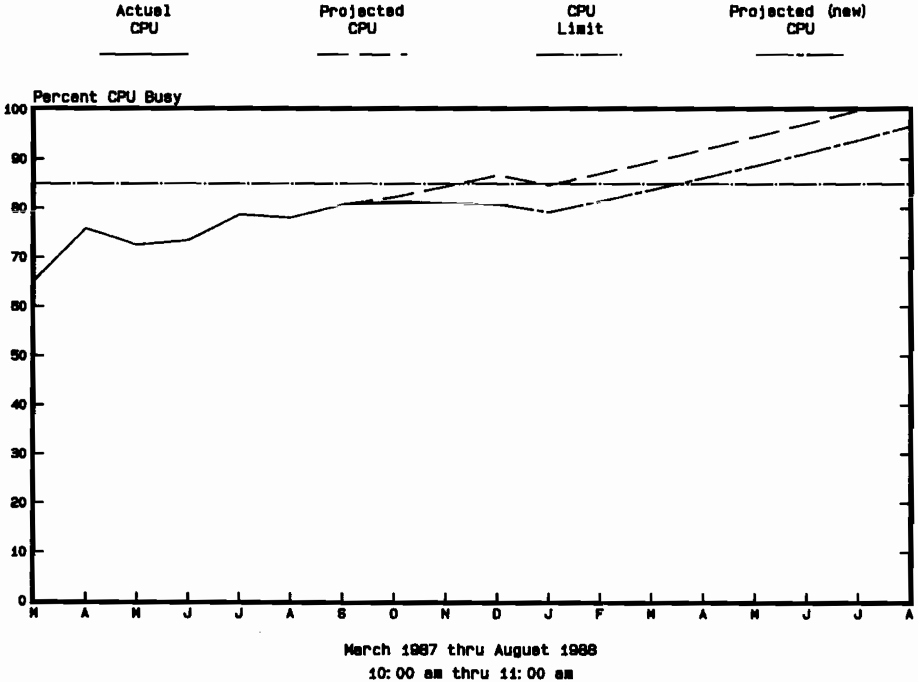
	7/87	8/87	9/87	10/87	11/81	12/87	1/88	2/88	3/88	4/88	5/88	6/88	7/88	8/88
Production	3.6	3.2	1.9	1.1	0.4	0.35	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Office Auto	2.0	1.4	2.2	3.3	3.0	2.76	2.8	2.8	2.9	2.9	3.0	3.1	3.1	3.2
Datacom	4.5	4.5	5.2	4.3	4.4	4.14	4.1	4.2	4.2	4.3	4.3	4.4	4.4	4.4
Utilities	15.5	16.4	15.4	17.8	16.0	22.33	22.7	23.2	23.7	24.1	24.6	25.1	25.6	26.1
Editors	17.2	16.6	17.9	15.1	14.3	14.10	14.3	14.6	14.9	15.2	15.5	15.8	16.2	16.5
HPDESK	16.4	17.0	15.3	20.1	17.6	15.09	11.6	11.8	12.0	12.3	12.5	12.8	13.0	13.3
Other	19.3	18.7	22.5	19.4	24.9	22.00	23.1	24.2	25.4	26.7	28.0	29.4	30.9	32.5
Total	78.7	78.1	80.7	81.4	80.9	80.78	79.2	81.4	83.7	86.1	88.6	91.2	93.8	96.6

PROJECTED GROWTH BY BUSINESS ELEMENT

	7/87	8/87	9/87	10/87	11/81	12/87	1/88	2/88	3/88	4/88	5/88	6/88	7/88	8/88
Production							1	1	1	1	1	1	1	1
Office Auto							2	2	2	2	2	2	2	2
Datacom							1	1	1	1	1	1	1	1
Utilities							2	2	2	2	2	2	2	2
Editors							2	2	2	2	2	2	2	2
HPDESK							-23	2	2	2	2	2	2	2
Other							5	5	5	5	5	5	5	5

UNIVERSITY SYSTEMS HP3000 SERIES 70 RED/US3

Average Peak CPU Projections



DATA INTEGRITY AND RECOVERY

AUTHOR: TERESA BRZOZOWSKI

CAROLIAN SYSTEMS INTERNATIONAL INC.
3397 American Drive, Unit 5, Mississauga, Ontario L4V 1T8 CANADA

Database failures and resultant recovery efforts cost HP3000 users thousands of dollars every day in lost processing time and inconvenience. While this paper provides a discussion of IMAGE failure types and various methods of recovery, it also intends to educate the reader as to how some basic database design and implementation procedures can act as proverbial "ounces of prevention" - protecting you and your company from having to needlessly exist and suffer with logically and physically broken databases.

IMAGE FAILURE MODES

A quick review of IMAGE failure modes reveals that such common occurrences as system failures or hangs, disc media failures, datacomm line failures or application failures can all bring database processing to a screeching halt. Failures can also result in physical or logical damage to the database, with physical failures resulting from having bad data on disc (filesys), broken chain pointers or inconsistent root files. Logical failures can result because of missed updates, puts or deletes or missing delete flags. Whatever the cause, there are several standard and some new ways to repair the damage.

CLASSIC IMAGE RECOVERY

A standard method of IMAGE recovery is to restore your most recent copy of your database and forward recover using DBRECOV. However, there are some major deficiencies with this method of recovery. First and foremost, the process is extremely time consuming as it keeps users away from productive processing.

Also, DBRECOV uses a technique of recovery known as staging, whereby the restored DB is updated from the log file via staging files. The problem with staging is that large numbers of transactions can be ignored if an "end" is not found, resulting in these transactions not being applied to the database. The result can be a great deal of time and effort spent on forward recovery, with no guarantee that the recovery will be complete.

INTRINSIC LEVEL RECOVERY - ILR

If a failure occurs during an actual DB intrinsic such as updating, ILR can ensure physical integrity of your database by undoing the intrinsic. The problem with "undoing" is that with IMAGE databases, in some instances the log file and the database may not agree! Improvements made to TurboIMAGE have alleviated this problem.

TURBOIMAGE RECOVERY

With Turbo, ILR will complete the intrinsic call so that the logfile and the database agree, as opposed to just "undoing" it. Turbo allows you to forward recover with DBRECOV as does IMAGE, but it also allows you to initiate a rollback recovery.

Rollback recovery is a more timely method of recovery as it eliminates the need for a DBrestore and to reapply logged transactions to the database. Rollback recovery allows users to bring their current database up, and back out the last incomplete transactions, while complete transactions are left in place.

The use of ILR and Rollback recovery will generally ensure that more data is recovered than is possible with roll forward techniques. This is due to the fact that ILR with rollback recovery requires physical logging.

Physical logging ensures that the changes to the database are recorded and written to the log file as they occur. This prevents the log record from remaining in memory where they can be lost in the event of a failure.

Despite the time savings that can be realized with Turbo's newer recovery features, neither these or IMAGE recovery procedures are of assistance with another common occurrence that results in logically broken databases program aborts.

RECOVERING FROM PROGRAM ABORTS

Programming bugs, user errors and datacomm line failures are just a few of the occurrences that can result in a database becoming logically corrupt. To date, HP3000 sites have had to live with the fact that once their databases have become logically corrupt, that they have to endure this inconvenience until a full recovery procedure can be initiated.

PROBLEMS ASSOCIATED WITH ABORT RECOVERY

Again user downtime is the penalty that must be paid as users have to terminate, partial transactions are deleted or completed, and then users are allowed to access the machine again. However, if strong locking is not in place, the transaction interaction that has been rolled out can inadvertently undo a completed call. The real solution to this dilemma is to have a "net change" rollback. This is currently unavailable, as a "net change" rollback requires a detailed and intimate knowledge of the application.

SOLUTIONS - HOW TO MINIMIZE RECOVERY HEADACHES

The benefit of such facilities as DBRECOV and Rollback recovery can be greatly enhanced if you implement the following safeguards.

1. Turn on logging - despite persistent misconceptions, logging **does not** significantly degrade the performance of your machine. If you are not logging you have precluded yourself from virtually all methods of recovery.

2. Use Begins and Ends - Without DBbegins and DBends, by definition no logical transactions exist. Therefore, database logical integrity is impossible to determine. The best that can be done is to provide for an audit trail of physical transactions.
3. Strong Locking - Some method of strong locking should be implemented. Without strong locking, a transaction can interact with another transaction before it has completed, thereby making the result of a rollback recovery questionable.
4. Turn on ILR - Turning on ILR will ensure that your database will always be physically intact.

Implementation of these four key points is crucial if you are to ensure database integrity and ease of recovery for your company. They can result in tremendous reductions in user downtime and the time is spent on recovery procedures. There is however, another alternative method of database recovery, that when implemented with the aforementioned safeguards, will render downtime due to the initiating of recovery, or the existence of logically corrupt databases due to program aborts, a thing of the past.

AN ALTERNATIVE - DYNAMIC "ROLLBACK"

A facility that provides a dynamic rollback, will actually undo an aborted transaction as the abort occurs. This "real time removal" of an aborted transaction will result in your database always being logically intact. Without the existence of incomplete transactions in your database, it would also unnecessary to have to take the system down to initiate a cleanup.

Such a utility does exist, and is actually one product for the HP3000 from the Carolian Systems Research and Development group. Known as INTACT, this product provides these major capabilities which have been previously unaddressed and unavailable to HP3000 users.

your database, it would also be unnecessary to have to take the system down to initiate a cleanup.

For the HP3000 classics, Carolian Systems has developed a real-time roll-out utility called INTACT. Hewlett-Packard is also seriously investigating this data integrity issue for the Spectrum series, however, dates for the release of such a capability have not yet been released.

Pitfalls of Offloading Applications to PCs

by Mark W. Miller
JMA Technology, Inc.
P.O. Box 570727
Houston, Tx 77257-0727

And on the eighth day, Hewlett-Packard said "Let there be no graphics, spreadsheets or word processing to slow down business transaction processing on the HP3000" and lo, the users heard these words and they believed.

Consider the HP3000. For years, it has served as the central work horse in numerous companies. The general-purpose machine embraced every kind of application people could dream of. It has served well as a file repository, a point-of-sale processor, information storage and retrieval, sales tracking, business forecasting and even process control and monitoring.

It's processed transactions, statistics, documents, results, budgets, financial statements, labels, graphs and reports. It contains one of the most efficient, reliable and stable arithmetic logic units available in the market. It's output capacity outstrips all but the fastest impact and laser printers. It facilitates central control of data in a dynamic business environment.

For many businesses, whether for profit or non-profit, it was (and still is) the stepping-stone into the future of computing. It obeyed their every wish while serving as an all-purpose business computer, meeting all comers with enthusiasm. HP even pushed their own word processing, graphics and spreadsheet software for years (must have been to sell more hardware), saying that this was the best solution. And now, they tell us we can't do that any more. These applications will not be supported starting with the new spectrum line.

We must remove all vestiges of centralized office automation and relegate any perceived CPU intensive applications to their younger brother PC's. Thus ends an era of one vendor, one machine, total solution.

This 'HP way' of things is being touted by many influential users and a plethora of papers, presentations, seminars and propaganda have surfaced promoting it. Before moving all these "CPU intensive" applications off the HP3000, consideration should be given to the long-term effects. The philosophies involved require carefully planned and thought out decisions.

This paper exposes the flaws inherent in this 'HP way' of one-sided thinking by pointing out the issues of centralized versus offloaded applications. Further, it discusses the different aspects of permanently moving all of these applications to PC's. Finally, some of the myths involving 'CPU intensive' applications will be dispelled and guidelines for determining which applications to keep and which ones to offload will be included.

The HP Way of Things

As an immediate response to the 'HP Way' of thinking, I must say that this had to be a regression from their earlier stance and that they are ignoring a certain segment of their current users. This blanket approach is an insensitive and ignorant way to support these users. I'm talking about those companies that depend solely on the HP3000 to meet all their Data Processing needs.

On the HP3000, it is the activities, not the applications, that need to be monitored, quantified and justified on an individual user usage basis. This is the key to determining which activities need to be offloaded to PCs.

If all user activity is individually oriented and not vital to business operations, all this activity would probably be best offloaded to the single user oriented PCs. Upon close examination, however, most companies will find that much of the activity is vital to business operations. If this activity gets offloaded, loss of control over that information occurs.

You may ask why I so vehemently state these things when it is so painfully obvious that the future of computing will include PCs and distributed processing. It is because of the sheer simplicity of the concept that the HP3000 can be all things to a small or even medium size company. Additionally, not all companies believe that they must remain 'state-of-the-art' by installing PCs and implementing distributed processing.

For purposes of discussion, I will be limiting this talk to HP3000s and IBM PC, XT, AT and compatibles. The PC operating system mentioned is PC-DOS/MS-DOS version 2.0 and up without any special add-on hardware or software. The reason this paper is limited to these machines and operating systems is because these are the predominant PCs already installed in most HP3000 shops.

I am acutely aware of the multi-tasking capabilities of Windows and Wave-Mate, however, these packages are not yet in wide-spread use. The same is true for the PS-2 and compatible machines, OS-2 (not available yet), Unix operating systems and high-powered workstations. These are rarely found in the same environment as the HP3000.

Here, I will show you some reasons not to offload applications:

10 Reasons Not to Offload Applications

1. Wholesale Output Power

The HP3000 has more output power than any PC sold today. This output can be with a large variety of output devices including plotters, printers of all types, punch cards (what?), magnetic tape, serial communications, display terminals and many others. This power, coupled with spooling, allow tremendous output in relatively short time frames.

Because of the limited output of a PC, the user is often faced with the challenge of finding a way to print a report or plot a graph. Some software packages are now emerging that allow spooling to the HP3000 and networked PC's usually have a shared output device. This limitation, however, can cause delays and frustration among users.

2. Production Power

Many companies close their books monthly to allow analysis of financial status on a more timely basis. The vast quantities of numbers to be crunched can easily overwhelm a PC. The HP3000, however, is able to perform complete jobs of number crunching, reporting and graph plotting. These outputs are generally reviewed to aid in decision making for the business.

This is known as Production Graphics and Production Spreading. These jobs would take days to complete on a PC, whereas, the HP3000 handles them in hours. Control of the process is completely taken care of on the HP3000, whereas, if split over many PC's could cause confusion and possible loss of data, results or precision.

3. Application Droning

The HP3000, after fifteen years, has developed quite a third party hardware sales sector. As a result of this, older model HP3000s are available for less than ever before. Consider a dedicated HP3000 for a single application that is causing bottlenecks. This could free up considerable computing, input and output power for your primary machine. This type of distribution of applications is known as droning.

As the company grows, more drones may be implemented to distribute the load. This can even include more than one application. These HP3000's may also be networked with data and programs residing on different machines. With this scenario, the company could grow infinitely without ever needing mainframes.

5. End-user Communications

As more people use the computer, communications becomes more and more important. The electronic mail systems on the minis are excellent communications devices for those who used to play telephone tag all the time. The messages may even be forwarded in a network of HP3000s.

The management and administration may wish to use these mail systems to disseminate information, company policies, memos, etc. In any case, the quality of communications is higher with electronic mail than without it. Better communications means better productivity.

The ease at which data may be passed to other users is also a factor in productivity. On the HP3000, this is a simple and straightforward task. Even in a network of HP3000s, this is still relatively simple. If this is tried between the HP3000 and the PC it becomes a trying task at best. With terminal emulation software, it is easier, but the difficulty remains in assuring correct file sizes, data types, and special characters.

6. Data and File Differences

Binary data located on the PCs and the HP3000s are not the same. Care must be taken in transferring this data and loading it into the new PC application. At times, the data must also be re-organized and possibly require some decisions concerning 'excess' data from the HP3000 applications.

In the case of graphics, it may require massaging and possible re-organizing of data from raster to vector or vice-versa.

7. Lack of Features and Operational Differences

Much of the PC software available today lacks many of the features available on mini software. These features, while seeming very unimportant, may make the difference in how the company operates because they don't exist in the PC application. This can prove to be very frustrating to end users and management alike. This can translate to loss of productivity and profits.

Very few applications on the PC operate in the same manner as on the HP3000. These differences will cost in terms of re-training, loss of productivity and error corrections during the implementation. This loss will eventually pay for itself, unless the data transfers are awkward or untimely. Re-evaluating the usage of the application may change the companies needs and require scrapping whole systems because they are 'canned' or not worth adapting.

In any case, changes will be necessary to the data if downloaded to the PC. For spreadsheets, this may require changing commands, macros, use-files, etc. to correspond with the PC spreadsheet because they are incompatible. For word processors, it may require changes of commands, removal of imbedded commands and adding new imbedded commands or converting all text files to a new format.

8. Costs and Investments

Before you discard your HP3000 applications, consider the investment that you are throwing away. Not just the software costs, but user productivity and experience, data files, use files, worksheets, etc., and any 'systems' created. After all, you don't throw away your 4GL just because it produces slow reports from time to time. Remember, prototyping, testing theories and experimenting bring advances in processing and business.

This investment has been built over years and does not easily die. Gradual migration to similar PC products should be implemented to avoid alienation of users and loss of valued activities.

9. Back-up and Data Control

For many shops, no solid back-up procedures have been identified for PCs, much less implemented. This lends itself to loss of data integrity and control for any data that is passed through or to PCs. This condition, while reparable, could cause many headaches or loss of critical data.

10. Data Transfer

Any data needed from the HP3000 will need to be extracted and placed into a file, transferred to the PC and then imported into the application. This requires time and resources. Resources include HP3000 processor time, file access, transfer processing on both the HP3000 and PC, and PC processor time to import the data into the application.

Because this can quickly add up to more than it would take to finish the process on the HP3000, it could be a reason not to offload the application. If this is for ad-hoc or what-if types of applications, it may be justifiable. However, if it is a set process that never or seldom changes, it may be better off on the HP3000 than offloading.

Central Repository of Information

In many installations, the HP3000 serves as a repository of information. This usage is common for businesses because the idea of having centralized locale of information is appealing. If all information is located on one computer, the relationships of this information is simpler to envision and easier to manipulate into comprehensive analysis and reporting. This concept has prevailed since the first businesses used computers to organize massive volumes of data.

This is not to say that all information and its' processing must be continued in this environment, but merely a testimony to the solidity of the concept of centralized information. Today, that concept is expanded to include various other types of processing, including the use of personal computers as well as other types of computers for specialized processing needs. The HP3000 is not perfectly adapted for many of these unique needs, but rather becomes complementary to fulfilling them.

Data Security and Control

If the users are allowed free reign over data, control and security will be virtually non-existent. It becomes simple to introduce many copies of sensitive data on relatively unsecured PCs. While data must be accessible, it must be controlled enough to prevent loss or damage. Effort must be made to prevent carelessness with data (especially sensitive data).

In general, data is captured through various means, collated, categorized, massaged, 'transactioned' and placed into any of a variety of file structures. Once this data is captured, the company must decide what the value of this information is. If it is of value in steering the business into profitable markets or possibly more of a share in the current markets, then it must be identified as such and given the same protections and care that any other corporate asset.

Determining the value of this type of data is no simple task and not the purpose of this paper, so I will not discuss it here. Once a value is placed on it, however, ensuring that the data is exploited sufficiently to justify its' capture becomes the most important task at hand. After all, dead data is useless to any company. By using the HP3000 as a central repository for important information, the company may take complete advantage of it through any of various processing steps, regardless of the means used to accomplish these goals.

Data can be monitored, controlled, secured and tracked, just like any other asset. This may include issuing data like a company vehicle may be issued to an executive. It is expected to be returned in good condition and intact when the user is finished with it. This concept could very well protect this asset and insure its' integrity without possible compromise. Since a 'virgin' copy remains in the central repository, it is safe and assuming the employee who checked it out is honest, the copy will not be lost or allowed to be abused in any way.

Offload Decision-Making

There are, indeed, applications that should or could be offloaded to PCs. As previously mentioned, individual or ad-hoc oriented activities are best offloaded. But, what about that huge financial report we produce every night with that spreadsheet on the HP3000? Would it take less 3000 resources if we downloaded it to the PC or would we be better off leaving it where it is? In order to better understand the questions raised, we must examine the dynamics involved.

Architectural Considerations

First, we must understand the architecture of each machine, and what it was designed for. The HP3000 was designed as a multi-tasking, multi-user system using the time-slice/q-priority method of time sharing user (and system) processes. In theory, it can handle over 200 interactive user sessions and 50 or more batch jobs, each capable of tasking up to 63 programs. However, rarely will each user be running this many programs at the same time, all of the time.

Because of the nature of an interactive session, much of the time, the session is idle. During these idle times, other processes get serviced, and efficient use of the CPU resources is accomplished. When the session is not idle, the process monitor ensures that the user process gets served a reasonable number of times per second to allow a smooth flow of activity to the user.

The PC was originally designed as a single-task, single-user system using no time sharing concept. However, since the user is the only process to worry about, the user is always served instantly. While this concept is powerful, it can also be said to be wasteful when it is idle. Thus the un-used CPU resources remain un-usable because no other processes can access the CPU while the user is occupying the keyboard (of the PC). This is known as serial computing. One user, one process at a time.

In order to get a handle on using this idle resource during off hours, we can use them for completing processes previously accomplished on the HP3000. This is called distributed processing.

Assume for one moment that we are working with a production machine and that we want to offload this single process to a PC to finish the number crunching. We must decide whether or not to offload.

Formula Calculations

We need a method for determining the feasibility of offloading a process. To do this, we want to calculate a time index for each process technique desired. This first formula is designed to calculate the index for time to process the data on a PC with downloaded information from the HP3000, and then, upload the data for file update or for output.

$$I1 = E + T1 + A + P1 + O1 + T2 + O2$$

This next formula is for computing a time index for leaving the process on the HP3000.

$$I2 = E + P2 + O2$$

Variable Definitions:

I1 - Index of PC processing time.

E - average time to extract data (HP3000)

T1 - average time to transfer data down

A - average time to import data into PC application

P1 - average time to process data (PC)

O1 - average time to output data (PC)

T2 - average time to transfer data up

O2 - average time to import/output data (HP3000)

I2 - Index of HP3000 processing time.

P2 - average time to process data (HP3000)

These time indexes represent the average time it takes for each of the methods to complete the process. Note that these also represent single-tasking, head-to-head competition between the HP3000 and the PC. When the multi-tasking aspect of the HP3000 is taken into consideration, the time indexes for the PC must be multiplied for each additional task able to be processed on the HP3000. Using this methodology, fairly accurate indexes are produced and the real advantages of multi-tasking begin to show up.

Of course, these formulas are not usable with complete applications that are being offloaded. Offloaded applications derive pure and simple cost/benefit ratios and only HP3000 extract and download times (if any) need be considered. The formula (using above variable definitions) would look something like:

$I1 = E + T1 + A + P1 + O1$

In any case, this index is likely to be less than the HP3000 index, unless uploading becomes necessary. Additionally, the import, process, output cycle may be repeated as many times as desired with the same data without affecting HP3000 resources. If no HP3000 extract and download are necessary, the time index will always be better than that of the HP3000.

Of course, calculating E, T1, T2, O2 and P2 requires some analysis and testing since some factors may vary including quantity of data, system activity load, file availability (fragmenting), file record size and quantity of data manipulation (transformations, calculations, translations, formatting, etc.).

If the two indexes (I1 and I2) are equal, chances are, no benefits are achieved by offloading the process. If I1 is less than I2, the benefits are realizable. If the cost of developing the PC application/programs is not tremendous, the benefits would outweigh the costs. If, however, the costs are approaching the extreme, the benefits would not outweigh the costs. If I1 is larger than I2, forget about offloading the process at this time.

Naturally, there are ways to improve this cost/benefit ratio, however, that subject is not the purpose of this paper.

Ideas for obtaining these values include recording job wall time for one month for this process. Determining PC processing time for the import, process, export and transfers could be obtained by prototyping them. These figures are averages and not absolute since different numbers take different times to crunch.

Strategic Planning

Planning these moves is just as important as making them. By outlining them and studying them ahead of time, you can avoid critical mistakes. If you buy HP3000/PC compatible software (spreadsheets, graphics, word processing, 4GLs for prototyping, etc.), most of your job is done. Because these applications can easily use the same files, no translation/modification is necessary. Feasibility testing is simple and straight-forward.

If the software is incompatible, however, your job has just begun. Programs will be needed to manipulate the data into usable formats for the PC and the HP3000. If you share your plans with your end-users as well as management, you will most probably find out ahead of time whether or not any unforeseen obstacles exist.

Issues of Concern

There are some obvious areas that need attention before offloading any applications permanently.

PC Purchasing and Maintenance Costs

Consider that a copy of each software package must be purchased for every PC needing that application. Consider also, that you may need to expand memory, disk space or I/O boards to use the new application efficiently.

Oversized Applications

Some applications may be larger than the PC is designed for. One of our customers built a PC spreadsheet so complicated that it took over 5 hours to re-calculate (goal seeking). In two hours, he converted the spreadsheet to run on the HP3000 where it took less than 50 minutes to compute during a busy time frame.

It's fine to tie up a PC during off hours, when it is not needed. But, if this stretches into the day when an engineer or manager needs the PC, you are encroaching on end-user time. The moral: Don't offload extremely complex applications/activities to PCs.

Learning Curve for Applications

If the end user will be utilizing the application, they will have a learning curve. This will work fine if the application is used only on PCs after that. If the user must constantly switch from PC to HP3000 and back, it may be better off left on the HP3000. This also goes for learning to interact in general with the PC.

Productivity Improvements

Some productivity improvements are never realized because the gains are imperceivable. This is usually the result of poor planning and feasibility testing. The biggest problem involved here is usually data control (who's got what, where).

Home-Grown Software

If your end-users like to create their own applications and want to use company data in them, watch out! Most end-users do not sufficiently test their programs and this could result in contaminated data. If this data is then used to update company data, the contamination spreads. Encourage high standards and avoid contaminated data.

Duplication of Efforts, Systems and Results

Many times, without good communications, users may duplicate the efforts of others sometimes creating the same systems with differing means. Establish a good communication base and encourage sharing of applications.

Equipment Types and Compatibility

If no precautions have been taken, users may get incompatible equipment and forego decisions concerning sharing and communicating. Establish company guidelines to avoid this possibility.

Software Compatibility

If the software packages on the different PCs are incompatible, sharing data and ideas becomes difficult. The same is true of software between the PCs and the HP3000. Any time a new upgrade is received, be sure it is installed on all equipment, including operating systems.

Operating Systems

The DOS operating system is barely adequate at best. It requires excess typing and is filled with traps and holes. It may do well to leave the PC to the tasks it does best. At least, obtain a good menu facility to ease the end-user burdens.

Data Duplication

With many users downloading data, there is a big risk of data duplication. A timing problem could also occur. Consider the scenario where one user downloads a set of data; another user, not knowing this, does the same. Each does calculations and manipulations to get final results for updating the HP3000 data. One reloads and updates the data on the HP3000, soon followed by the other. What is the integrity of these updates? Questionable, at best.

Future Needs

Consider an application that is completely offloaded to a PC. The old software on the HP3000 is laid to rest. Six months later, the results of this activity are determined necessary to another application on the HP3000. You must now reverse the roles and upload data to the HP3000. Moral: Run your HP3000 applications until there are no more perceived needs for it for at least one year.

Data Base Dependent Applications

Some applications depend on data in a data base to complete certain tasks such as validation, inventory checks, etc. If this data is static and rarely changes, it is easy to justify offloading it. If the data changes regularly, however, your I/O involved in the transfer may easily outweigh the benefits. Graphics production and real-time budgeting are two activities that readily come to mind to remain on the HP3000.

Summary Data Bases

In every organization, there are certain key numbers that play an important role in decision-making. These might include inventory levels, revenues, cost of product, depreciation and/or sales. These figures are often needed in a summary form to be used in spreadsheets, graphics and/or word processing.

If this data was researched and identified, it could become standardized for the company. Further, it could be extracted, calculated and deposited into a summary data base by a standard job at standard time intervals. By having each application use this data whenever possible, a large reduction in processing can be achieved.

This 'summary' data base could then be reviewed periodically to determine if any changes are needed. This accomplishes two things: 1. a reduction in processing and 2. a reduction in the need for constant access to production data by user activities.

Communications and File/Data Transfer

When you begin to use distributed processing with the HP3000 and PCs, a word about communications must be said. Since time is of key importance, good communications are necessary to facilitate efficient processing and timely results. There are three basic types of connections available: direct or hardwired, phone (with modems) and networked.

The direct connection is as fast as your HP3000 can communicate (2400, 9600, or 19200 baud rate depending on the CPU and port types). The quality depends on the type of cable and distance from the CPU.

The phone connection is as fast as the CPU, but limited by the modem speed. The quality depends on the modem and the phone line type.

The network connection is the fastest of all. Speeds of up to 1 MByte are available now and speeds of up to 10 MByte are promised in the near future. The quality depends on the quantity of traffic and errors encountered during the transfer. Of the three types of connections, this is the most sensible.

Don't skimp on the connections. After all, the communication speed and quality are an integral part of the processing load and can greatly limit timely throughput. Any good terminal emulation software package will do for the interface. Be sure you have options for alternate protocols since they may be needed for tricky transfers.

Myths About 'CPU Intensive' Applications

Over the years, many people have talked about system 'HOGS' and their nature. Fingers have often pointed towards these applications with anger, disgust and ridicule. Here, I will dispel some of these myths.

Spreadsheets

The spreadsheet began its life on a PC. It was a tool for manipulating numbers quickly and displaying (or printing) them in a desired format and order. Soon, the spreadsheet was available on the mini, and now, even on mainframes. Many people point to them as CPU hogs.

First, the only time this is true is when a re-calculation is taking place. Since re-calculations are done only occasionally, the overall CPU usage, while significant, is spread over a period of minutes or hours. Much of this CPU activity is done when the CPU may otherwise have not been busy.

Second, the quantity of calculations is solely dependent on the size of the spreadsheet. By keeping spreadsheet size down and combining tasks, this problem can be reduced.

Third, the quantity of work accomplished in comparison with hand-made calculations is overwhelming. The HP3000 can do more calculations in one minute than most people can do in an hour. Such a versatile tool could not be replaced with 100 COBOL programs.

Graphics

Another victim of prejudice are graphics programs. Consider the quantity of data that must be crunched to produce a set of charts. Of course, to collect, prepare and compile the data, the HP3000 must be used anyway. The PC cannot accomplish this task alone. Since most of the overall CPU time is used up in these tasks, offloading the results to a PC for visual or print output barely makes sense.

Production graphics require a mini, since the PC is incapable of crunching this much data in a reasonable amount of time. Since this type of graphing is necessary to make meaningful decisions in business, why wait for the PC to finish?

Word Processing

Another perceived culprit of excess CPU theft is word processing. The only way this could be true would be by typesetting documents regularly. If the text is not automatically re-formatted, the word processor takes up no more resources than any text editor currently running on the HP3000.

If, indeed, documents are typeset on the HP3000 as a part of production, chances are that this is the best place to continue to perform these tasks. It becomes especially so when form letters are being produced with information from a data base. This high quantity output could not be accomplished on a PC in a timely manner.

Centralized word processing is still in demand and only recently several software packages became available on minis. This is positive proof that some companies still want to keep tight control over text production.

Which Ones Go?

According to a 1987 survey made by Datamation, planned PC purchases were up three percent and planned mini purchases were down thirteen percent compared to 1986. It also stated that users are using their PCs an average of 5.6 hours a day. This indicates a trend towards new distributed applications on PCs.

We can't simply ignore these machines. They must be used more to aid in data processing. Let's look at what activities are best suited for each machine.

<u>PC Oriented Activities</u>

what-if games	to pre-test decisions based on various possible scenarios and outcomes
project budgeting	to create/change the budget for a project
ad hoc letters	that don't rely on data from the HP3000
desk top publishing	for presentations, literature, documents
cost justification studies	discover economics of cost/benefits
proposal development	proposals that don't affect others
presentation graphics	for ad-hoc presentations

personal budgeting	for a small department or a person

In general, anything that does not directly affect the whole company or whole departments. Any application that does not require constant access to the HP3000 for data, files, or other resources. Less complex financial spreadsheets, graphics, word-processing or statistical analysis. Stand-alone type applications. Data collection and verifying applications.

HP3000 Oriented Activities

production graphics	month-end, administrative or management charts
production letters	form letters, personalized mass production, statements of accounts, etc.
whole company/division dependent	information critical to or directly affecting them
multi-user dependent	data needing sharing, memo distribution
real-time budgeting/allocation	same-day re-structuring of budgets, allocations of personnel/resources
data base dependent	requires access to kinetic on-line information
production spreading	month-end financials, statements, special financial reports, goal seeking
information dependent activities	depends on changing information
sensitive/confidential info.	information that needs guarding from potential leaks
high-precision calculations	requiring at least sixteen significant digits
financials & extrapolations	production financials and reports

In general, applications/activities that are too complex, data dependent, sensitive or bulky to be accomplished on a PC. Any application that is multi-user and/or multi-tasking. Activities that require vast information resources, gathering, compiling and/or sorting. Activities that need to be shared among a group of people for the purpose of splitting responsibilities and later re-combination.

So, what does it all add up to?

In this paper, we discussed the pros and cons of offloading HP3000 applications to PCs and reasons not to offload an application or activity. Suggestions were given concerning decision issues, what to offload and how to decide what to offload. Additionally, we dispelled some of the myths about CPU intensive applications and their proper place among software on the HP3000. Finally, we covered some of the related issues concerning offloading and gave ideas about avoiding pitfalls when deciding to offload an activity.

In conclusion, let me state that the PC can play an important role in your data processing strategy if you take the time to properly plan for it. Do not just offload an application before considering the implications involved. How you accomplish this migration is as important as the resulting configuration and the affect it has on your company may cause you trouble if you don't look ahead.

Keeping all these issues in mind, you may not ever hit a snag, or you may discover that the best laid plans weren't considerate enough. In any case, you have the choice of acting intelligently or blindly following the HP rhetoric. You may win, but it could be at a cost higher than you wanted to pay.

Bibliography

1. Ralph E. Carlyle, "Midrange Shootout: Mini/Micro Survey" Datamation, November 15, 1987, 60-76.

2. Robert Green, David Greer, and Mike Shumco, "Squeezing the last bit out of your HP3000," Interact 8:4 (April 1988):68-80.

3. M. E. Kabay, "Office Automation - Appropriate Technology," Interact 8:3 (March 1988):90-91.

4. Rozan S. Brown, "The PC to HP3000 Connection," Interact 8:3 (March 1988):99-107.

5. Karen Heater, "Networking the Mini and the Micro," Interact 8:3 (March 1988):55-61.

6. M. E. Kabay, "Office Automation - Data Interconversion," Interact 7:11 (November 1987):106-107.

7. Gary Thompson, "Electronic Publishing with the HP3000," Interact 7:8 (August 1987):50-57.

8. Bruce Edwards, "Office Automation - PCs in the Office," Interact 7:1 (January 1987):54-55.

9. David R. Lee and Del Jones, "MIS - Applications Development on PCs and LANs Part 1: New Design Tools," Interact 7:1 (January 1987):60-63.

10. David R. Lee and Del Jones, "MIS - Applications Development on PCs and LANs Part 2: Four new-user support tools," Interact 7:2 (February 1987):56-62.

11. David R. Lee and Del Jones, "MIS - New Techniques in Applications Development on PCs and LANs Part 3: Living in a nonsequential world: Supervisory Programming on PCs," Interact 7:3 (March 1987):63-64.

12. Tomorrow's Management Generation Roundtable Staff, Datamation, September 15, 1987, 126-138.

13. Rich McCahan, Jerry Johnson and Sandi Fruehling, "Office Automation: Managing Your PC-based Word Processing Documents," Interex Las Vegas Conference Proceedings, 1987.

14. Jean Pierre Martin, "Business Graphics: Micro vs. HP3000," Ibid.

15. Steven Carnegie, Richard Corn, and Robert Mattson, "Production Graphics on the HP3000 - It Can and Should Be Done," Ibid.

Decision Support System

Parvin Rahnavard
Integrated Decision Elevation And Science

6809 Wisconsin Ave
Chevy Chase, Maryland 20815

1. Introduction

A Decision Support System(DSS) must be capable of presenting an integrated representation of diverse types of knowledge and manipulation of that knowledge. This depends on the effective management of two types of knowledge: 1. Descriptive knowledge(i.e., data, information) with which MIS is concerned, and 2. Procedural knowledge, also called a model, which specifies an algorithm that tells us how to derive new knowledge in the sense of facts, expectations, or beliefs. Popularity of personal computers and spreadsheets have proven that users will build and use models given the right tools. In addition, increased availability of relational database systems, 4th GL, and AI languages significantly facilitate user access to data resources. Senior organizational management should take advantage of these facts and technology to accommodate the organization's strategic planning in a timely fashion rather than relying on the outdated file structures and programming languages.

This paper discusses some of the problems that MIS is confronted with and recommendations for short-term and long-term approaches to integrate information into a cohesive framework with the ultimate outcome of becoming an integrated corporate knowledge across time and people.

2. Barriers to the Evolution of Information Management

Information management has been forced to grow in order to accommodate the increased variety of automated tasks for the operational control and the increased demands for information by varied users within the organization. These two trends have brought to the forefront a growing need for data control and management.

2.1 Data Control

It has always been assumed that the system which used the data automatically owned the data. Thus, if the same data resides in different systems with name, editing, and validation rules that could vary within the systems. Then, when a user wanted to perform data analysis and information planning through the use of the data that went across system boundaries, the extracted information would contain data inconsistencies that would need to be resolved. This has led to loss of credibility of MIS departments.

2.2 Data Management

When systems are usually developed without concern for overall requirements, it is difficult to share data for the organization's multiple needs. Thus, a lot of time and effort is spent in creating new files and programs for existing data. This results in management's lack of ability to obtain the required information for making timely and knowledgeable decisions. Figure 1 shows such an environment:

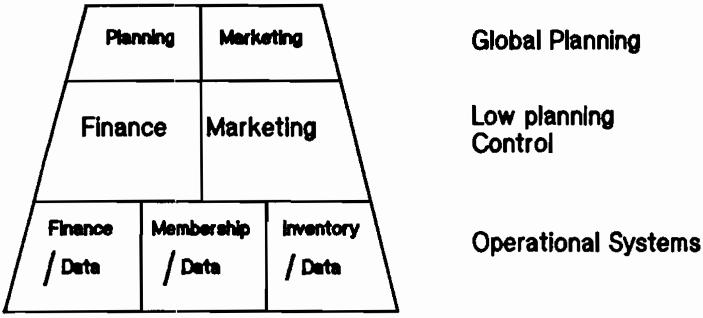


Figure 1

To overcome the above mentioned problems, there is a need for an organizational decision support system which would satisfy the need for data control and data management in the corporate environment.

3. Decision Support System (DSS)

For the purposes of this paper a summary of the objectives and basics of the DSS components and its implementation in Information Management environment is discussed.

3.1 Objectives and Goals

- Improve and increase management's access to data.
- Improve manager's ability to more quickly respond to ever changing organizational information.
- Reduce the level of effort for accessing information.
- Reduce the lead time for acquiring information.
- Integrating capabilities such as spreadsheets, graphics, and mathematical models which can predict, simulate, or optimize the consequences of decision.

3.2 Fundamentals of DSS

3.2.1 Dictionary

The primary control element in a information management services is the information dictionary. It has been estimated that the Federal Government can save $150 million over the next decade by adopting a standard dictionary system. The dictionary contains descriptions of, and relationships with other information resources. The dictionary is instrumental in the planning, administration and operation of an organization's activity. The concept of the dictionary has been used in the following major areas within the DSS:

a. Data Dictionary

The data dictionary is the logical place for sharing and centralizing the control of data. This part is the "active" dictionary, which means every request for data passes through the dictionary which is the arbiter of how the data is cataloged, where the data is located, the value transformation, edit validation, presentation characteristics, and so on. DSS would then direct the file handler, and the operating system to obtain the data items and apply the various operations on the item before final presentation to the user.

b. Reports, View tables, Models Dictionary

This dictionary is a logical place for defining, sharing and centralizing control of reports, view tables and models.

Reports, view tables and models consist of a general structure and its associated data. This dictionary is vitally concerned with how to orchestrate data in order to perform any kind of manipulation which turns into defining dynamic relationships.

c. Security Dictionary

In addition to the built-in security that the relational database contains as a global security, it may be necessary to create a security dictionary based on the organizational structure. Figure 2 shows an example of the security levels that can be incorporated into the dictionary.

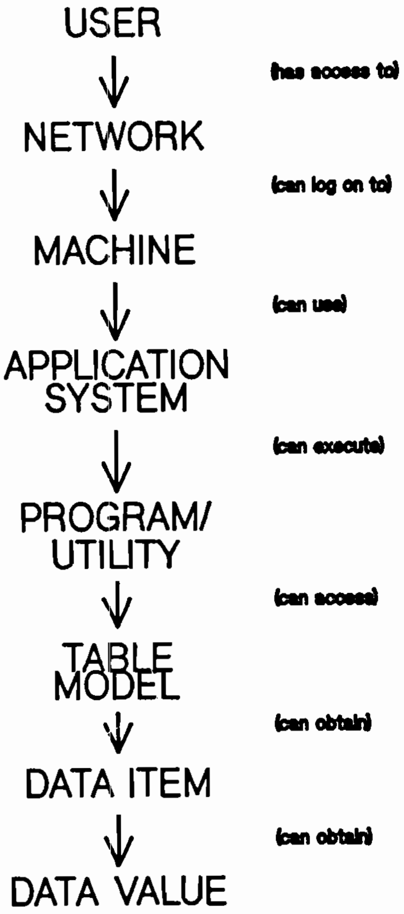


Figure 2

3.3 System structure

This is a customized on-line command processing system which manipulates data in response to functionality features implemented in the system. These functionalities are based on the nature of the organization's decision making applications.

The system design is based on a hierarchy of reducing the most complex functions to simple functions resulting in a series of single modules residing in program/module database. There are two types of single modules: directional and functional. Directional modules determines the availability of the other modules within the system hierarchy. Functional modules are responsible for processing input data to create output data. Since modules have been defined in the program/module database, automatic documentation would be generated initially and/or anytime there is a modification to the system.

This design provides an open-ended dynamic capability for implementation and/or integration of new features based on user needs.

This eliminates managements reliance on switching among separate software tools. Such an approach will reflect the nature of the business organization, and exhibit much of the flexibility inherent in a person's mental knowledge.

4. Implementation

Based on the organization's strategic plan, Senior management may decide on two different approaches: 1. Implementation of short-term and long-term plan; or 2. Implementation of long-term plan only. If the organization decides on selecting plan one, then there needs to be a parallel operation to incorporate both.

4.1 Short term

Refer to Figure 3. Since the existing structure is based on the application system owning the database, retrieval of the information can only be achieved through a vertical or one way access, restricting complete use of the system capabilities. Such will result in quicker access to information, however, lack of flexibility across databases.

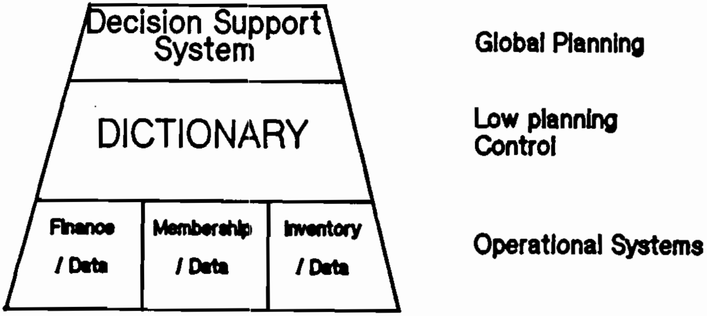


Figure 3

4.2 Long term

This step requires a total redesign and restructuring of the information at the corporate level rather than at the operational level. The redesign and restructuring would take place within relational data structure environment. The dictionary would be the sole communicator with the data area and the operational and planning systems. All systems should be designed independent of the data structure. Figure 4 shows such environment.

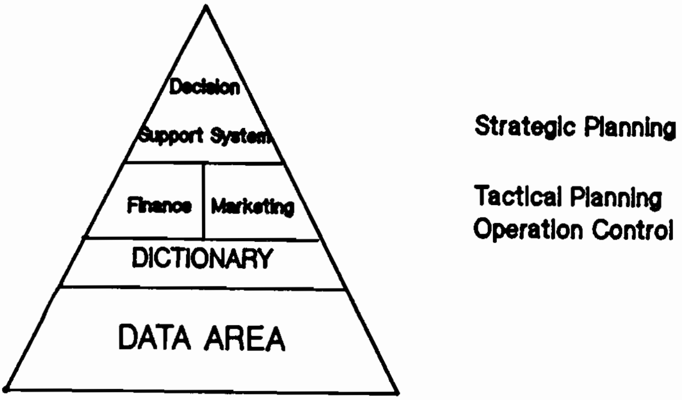


Figure 4

Understanding Migration

David Elward
Taurus Software, Inc.
770 Welch Road Suite 3A
Palo Alto, CA 94304

Introduction

If you're considering purchasing an HP3000 Series 900 machine, then there's a very good chance you will be "migrating" once it arrives. By migrating, I mean moving your existing programs and data files from your "classic HP3000" to your new "Spectrum Machine".

Migration has never been much of an issue within the Hewlett-Packard world, because up until now, all HP3000's have had the same hardware instruction set. This means that compiled programs will run on any HP3000, with the only difference being performance.

As you probably know by now, the Series 900 machines have a completely different instruction set than classic HP3000's. This instruction set has fewer instructions than traditional computer instruction sets and is based up RISC (Reduced Instruction Set Computer) technology. Hewlett-Packard calls this technology Hewlett-Packard Precision Architecture (HPPA). Within this paper, the terms HPPA machine, Spectrum machine, Series 900 machine, and MPE XL machine will all be used interchangeably to indicate HP's new computer. The term "classic HP3000" will be used to indicate older machines that run MPE/V.

When you begin learning about migrating and the new HPPA machines, you soon realize that there is an abundance of terms you've never heard before. Terms like "native mode", "mapped files", "Sysgen", and "MPE XL" are some of them. This presentation will focus on what these terms mean to you and your migration. Specific topics that will be discussed are as follows:

1. What is migration?
2. What is the difference between Compatibility Mode and Native Mode?
3. How do I prepare for migration?
4. What are some of the new MPE XL features and commands?
5. What are some of the new MPE XL utility functions?

What Is Migration?

Migration is simply moving programs and files which run on MPE/V based machines to HPPA based machines. The question then is: "Why is any migration effort needed at all?"

When the HP3000 was first introduced, it had an operating system called MPE, meaning Multi Programming Executive. Subsequent revisions of MPE were released, and most of you are probably now running the MPE V/E operating system. On the HP3000 Series 900 machines, the operating system is called MPE XL. The operating system on all classic HP3000's is written in a language specifically designed for the HP3000 hardware called SPL, Systems Programming Language. For the most part, MPE XL is a complete rewrite of MPE/V in a language called Pascal/XL. Pascal/XL is standard Pascal plus a variety of programming extensions added by Hewlett-Packard.

When MPE XL was designed, paramount importance was given to the issue of compatibility. With few exceptions, MPE XL is, and was designed to be, completely compatible with all MPE/V software. To make all MPE/V software run on MPE XL, Hewlett-Packard had a big problem because all MPE/V programs contained instructions for the old hardware which won't run on the new machines. The solution HP chose, was to write software that would emulate the classic HP3000 hardware and to integrate this emulator deep within MPE XL. By doing this, Hewlett-Packard has made the differences between MPE/V and MPE XL almost completely invisible.

Compatibility Mode versus Native Mode

"Compatibility Mode" (CM) is the term HP has chosen to mean running a program that contains instructions for the classic HP3000. When a compatibility mode program runs on MPE XL, the instructions are actually being emulated by the emulation software.

Almost all, if not all, of your programs will execute in compatibility mode with little or no effort on your part. Compatibility mode programs are typically created on an MPE/V machine, and then moved with `:STORE/:RESTORE`, or `:DSCOPY` to an MPE XL machine. You can create a compatibility mode program by running an MPE/V compiler in compatibility mode on a MPE XL machine. Running an MPE/V compiler in compatibility mode generates object code with classic HP3000 instructions. This means they can run on the classic HP3000 using its instruction set or in compatibility mode on the HPPA machine emulating the classic HP3000 instruction set.

"Native Mode" (NM) is the term HP has chosen to mean the running of a program that executes HPPA instructions. A native mode program must be created by a native mode compiler on MPE XL, and will not run on MPE/V.

Native mode programs execute much faster than compatibility mode programs because the process of emulation takes time. On the average, a native mode program will execute about 12 times faster than its compatibility mode counterpart. Currently, there are four native mode compilers available on MPE XL. They are: Pascal/XL, FORTRAN 77/XL, COBOL/II/XL, and C/XL.

OCT

You are probably nôt surprized that HP has created a middle ground. The OCT (Object Code Translator) produces programs that can be executed in native mode and on the classic HP3000 with out having to recompile.

What the object code translator does is translate the classic HP3000 instructions contained in executable code (PROG or SL files) to native mode instructions. The native mode instructions are then added to the program or SL file, leaving the classic HP3000 instructions in place.

There are two reasons for leaving the old instructions in place. One reason is because the old code is used by the translated instructions, and the other is so that the translated file may be moved back to MPE/V and still work. It will still work because MPE/V ignores the HPPA instructions within the file. Translated programs tend to run about 4 times faster than untranslated programs.

Switching Modes

Sometimes it is necessary to have a program run in both native mode and compatibility mode. For example, a native mode COBOL program needs to call an SPL procedure. Since there is no native mode SPL compiler, it must remain in compatibility mode. Hewlett-Packard has provided a mechanism for doing this called the "Switch Subsystem".

The Switch Subsystem is a set of procedures that enable a program to call a procedure that runs in the opposite mode than the program is currently running in. In order to understand how the switch stub works, we need to understand the internal data structures.

On MPE/V, the only data structure that is created when the program is executed is the program's stack. (See Figure 1) However on MPE XL three different data structures are created: the Compatibility Mode Stack (Same format as MPE/V), the Native Mode Stack, and the Native Mode Heap. (See Figure 2)

Environment for MPE/V Program

```
      +--------------+
      |     PCBX     |
DL    +==============+
      |  DB Minus    |
DB    +--------------+
      |  DB Plus     +
      +--------------+
      |              |
Q     +--------------+
      |  Q plus      |
S     +--------------+
      |              |
Z     +--------------+
        Program Stack
```

Figure 1.

Environment for MPE XL Program

```
     +--------------+   +--------------+   +---------------+
     |     PCBX     |   |              |   |               |
DL   +==============+   |    Global    |   |    Dynamic    |
     |  DB Minus    |   |     and      |   |    Storage    |
DB   +--------------+   |     Local    |   |               |
     |  DB Plus     +   |   Variables  |   |               |
     +--------------+   |              |   |               |
     |              |   |              |   |               |
Q    +--------------+   |--------------|SP |               |
     |  Q plus      |   |              |   |               |
S    +--------------+   |              |   |               |
     |              |   |              |   |               |
Z    +--------------+   +--------------+   +---------------+
     Compatibility      Native Mode        Native Mode
     Mode Stack         Stack              Heap
```

Figure 2.

When a program is executing in CM, it has access only to the Compatibility Mode stack. When a program is executing in NM, it has access only to the Native Mode Stack and Heap. Because of this, a "switch stub" is required to change modes. The switch stub is responsible for calling a switch procedure with the correct parameters to direct it to copy the procedure parameters from the current stack to the stack for the opposite mode. An easy fill-in-the-blank utility program is supplied with MPE XL to help the programmer write switch stubs.

Which is better for my program - CM or. NM?

When it comes time for you to migrate, you'll be faced with many decisions, not the least of which is whether your programs will run in native mode or compatibility mode. The advantage of native mode is that your programs will run much faster. The advantage of compatibility mode is that you can move your programs from MPE/V to MPE XL and run them all within a matter of minutes.

In the long run, an attempt should be made to get as many programs as possible to run in native mode. Pascal and COBOL/II programs should not present much of a problem to migrate to a native mode compiler, and FORTRAN 77 programs should migrate to native mode easily but watch out for the IEEE floating point.

Relatively Easy Programs to Migrate to Native Mode:

1. Pascal programs. Pascal/XL has many new features over Pascal/V, however you should be careful when using them because they will not work on MPE/V.

2. Cobol/II programs. Cobol 66 programs will need to be changed to Cobol/II, before they can be compiled in native mode.

Potentially Difficult Programs to Migrate to Native Mode:

1. Fortran Programs. All Fortran 66 programs will have to be changed to Fortran 77 before they can be compiled in native mode. Floating point numbers are stored differently internally on HPPA machines than on classic HP3000's, so any floating point data that has been written to files or data bases must be changed to IEEE format using an intrinsic supplied with MPE XL.

Difficult or Impossible Programs to Migrate to Native Mode:

1. SPL Programs. There is no native mode SPL compiler available from Hewlett-Packard, although there is one called SPLASH available from a third party. SPL programs must run in compatibility mode or must be rewritten in another language.

2. Basic Programs. Basic/3000 will never be available in native mode on MPE XL, however HP Business Basic will be.

3. Programs that use privileged mode (PM). Many privileged mode programs will run correctly in CM, and many will not. Because the MPE XL operating system is completely different internally than MPE/V, there is very little chance that privileged mode programs can be run in native mode without major changes.

Native Mode Considerations

In general, there are two big differences between the behavior of compatibility mode programs and native mode programs. One is data alignment, and the other is floating point. Compilers like to align data on word boundaries; the trouble is that classic HP3000's have 16 bit words, and Series 900 machines have 32 bit words. This will cause the data to be stored differently in compatibility mode programs than in native mode programs. This could cause problems for programs that use existing data files, or programs that call external procedures expecting the data in a different format. Hewlett-Packard has supplied a compiler option for the native mode compilers to direct the data to be aligned the exact same way as it would be using a compatibility mode compiler.

Classic HP3000's use their own format for storing and manipulating floating point (Real) numbers, and HPPA machines use the IEEE Standard for storing and manipulating floating point numbers.

Single precision HP3000 floating point numbers have a precision of 6.9 digits and a range of ± 1.2E77 to ± 8.6E-78. Single precision IEEE floating point numbers have a precision of 7.2 digits and a range of ± 3.4E38 to ± 1.4E-45.

Double precision HP3000 floating point numbers have a precision 16.5 digits, and the same range as single precision numbers. Double precision IEEE floating point numbers have a precision of 15.9 digits and a range of ± 1.8E308 to ± 4.9E-324.

This format difference will probably have a negligible affect on mathematic results, however it is a problem for floating point data stored within files. Classic HP3000 floating point numbers stored within files, will have to be converted to the IEEE format using the `HPFPCONVERT` intrinsic if you wish to access them in native mode.

Preparing for your Migration

HP has provided a number of facilities to allow you to prepare for you migration prior to your HPPA machine arriving. One of the utility programs which runs on the classic HP3000 is called RTM (Run Time Monitor).

The RTM is a utility intended to help you identify areas within your MPE/V programs that could cause a problem when ported to MPE XL. Because of fundamental differences between MPE/V and MPE XL, there are some things that may work on MPE/V that will fail on MPE XL. The RTM is intended to help you detect these things by logging calls to MPE/V intrinsics that could be a potential problem on MPE XL.

Logging is controlled using a program called `RTMSYS.PUB.SYS` and the logging results may be printed using a program called `RTMREP.PUB.SYS`. Typically, the user will monitor an application being considered for migration for a number of days. Then the RTM reports will printed and any potential problems will be investigated. The advantage of the RTM is that it runs on MPE/V, and problems may corrected long before migration actually begins.

OCA

Another utility which that can be run on the classic HP3000 to help your prepare for migration is the OCA (Object Code Analyzer). The OCA is a utility program that scans program or SL files for potential MPE XL problems.

It is intended to perform the same function as the RTM, however its method of operation is different. The RTM logs intrinsic calls when they actually happen; the OCA scans a program or SL file looking for intrinsic calls. The OCA has the advantage of being able to obtain the results immediately without waiting for days of logging. Its disadvantage is that it is may not be as accurate because it cannot always tell what parameters an intrinsic is being called with. Like the RTM, the OCA runs on MPE/V, and may be used before migration actually begins.

MPE XL New Features

A number of new features have been added to MPE XL machines. The features fall into three areas: intrinsics, mapped files, and command interpreter changes.

Intrinsics

As you may have guessed, new intrinsics have been added to MPE XL. Many of the new intrinsics have to do with switching between NM and CM. Others are provided to access features of the new command interpreter, including a new HPCICOMMAND intrinsic that can execute any MPE XL command including User Defined Commands (UDCs). Another new intrinsic is HPFOPEN that opens a file just as the FOPEN intrinsic does, but HPFOPEN is implemented with a couple of new features and is designed to be easily expanded.

Mapped Files

Using the HPFOPEN intrinsic, a program may now open a file "mapped". When a file is opened as a mapped file, a pointer is returned to the calling program. This pointer may be used to access the file directly without going through the file system. The mapped file may be treated exactly as if it was a data array within the program's stack. The advantage of mapped files is a tremendous performance improvement because the file system overhead is virtually eliminated.

MPE XL Command Interpreter

The MPE XL Command Interpreter has almost all of the commands that the MPE/V Command Interpreter has. The commands that have been deleted were deleted because they have no place within MPE XL. The deleted commands include CACHECONTROL, DATA, FULLBACKUP, GIVE, LISTVS, PARTBACKUP, PTAPE, and VINIT.

Some of the existing MPE/V commands have been modified or enhanced on MPE XL. Some of these commands are: IF has been greatly enhanced, LISTF has had some new options added, LISTACCT, LISTGROUP, LISTUSER have been changed to give output similar to LISTDIR5, REDO has been enhanced, RUN has had some options added.

Several new commands have been added to MPE XL that give dramatic improvement over MPE/V. The SETVAR, SHOWVAR, DELETEVAR, and INPUT commands have been added to manipulate variables. Variables are similar to JCWs except that they may contain string and boolean values in addition to numeric values. About 60 variables are predefined within MPE XL and they contain a variety of information about the users environment, such as the CI prompt, user's jobname, user, group and account among others.

A redo stack has been added that saves that last 20 or more commands. A DO command that does any command within the redo stack has been added, and a LISTREDO command that displays the redo stack has also been added.

A fast COPY command has been added that copies files 10-20 times faster than FCOPY. A PRINT command has been added to display the contents of a file, or prints the file upon a line printer without having to use an editor.

Perhaps the greatest improvement is the addition of command files. Command files are similar to UDCs except that they may only contain one group of commands, and they do not need to cataloged. Whenever a command is entered that MPE XL does not recognize, it is assumed to be a command file or program name, and MPE XL searches the user's group, PUB group, and PUB.SYS group file a program file or command file with the same name.

SYSGEN

There is no SYSDUMP program on MPE XL to configure your system. SYSDUMP has been replaced by a program called SYSGEN to configure your system and make cold load tapes. Backup is performed using STORE. SYSGEN has a much different approach than SYSDUMP; it is command oriented rather than question oriented. This approach is much more direct than SYSDUMP's.

In addition to SYSGEN, there is a program NMCONFIG that is used to configure your LAN (Local Area Network), DTC (Distributed Terminal Controllers), and terminals connected to DTC's. Since all terminals on MPE XL except the console must go through a DTC, the NMCONFIG program must be used to configure all of your terminals.

A program called VOLUTIL replaces VINIT and is used to configure your disc drives.

DIRMIG

If you wish to replace one of your classic HP3000's with a Series 900 machine, you will want to move your entire accounting structure along with all of your files to the new machine. Hewlett-Packard has supplied a program to assist you in doing this called DIRMIG. DIRMIG runs on your MPE/V system and creates a tape containing accounting and configuration information to be moved to MPE XL.

Utilities

Many of the system utility programs have been either removed or replaced on MPE XL. The following list summarizes the major changes:

1. LISTDIR5 has been removed. Its functionality has been moved to standard MPE XL commands. LISTF options 3, and is similar to the LISTDIR5 LISTF command. LISTF option 4 is similar to the LISTDIR5 LISTSEC command. The LISTUSER, LISTGROUP, and LISTACCT commands now have output similar to LISTDIR5, and the octal dump mode has been eliminated.

2. The FREE5 program has been replaced with the DISCFREE program. DISCFREE performs the same function as FREE5, but the output format is different.

3. SPOOK has been modified internally, but no differences are visible to the user.

4. The LISTEQ5 program has been eliminated. It no longer has a use in MPE/V or MPE XL because of the LISTEQ and LISTFTEMP commands.

5. DEBUG has been completely rewritten and does not bear any any resemblance to the MPE/V debugger. The MPE XL debugger is many times more powerful and makes extensive use of windows to display information.

Summary

The first step towards a successful migration is education. MPE XL contains many new things that at first can be overwhelming. What is comforting is that when you begin to use MPE XL, you don't even need to know you're using it. All of the commands you are likely to use perform just the same, and programs moved to MPE XL in compatibility mode just run. Only when you are ready to maximize the benefits of your new machine do you need to have a good understanding of the migration process.

Dodging Bullets in Your DP Shop

Victoria Shoemaker
Taurus Software, Inc.
770 Welch Road Suite 3A
Palo Alto, CA 94304

Introduction

At last count, there were exactly 3,572,614 HP3000 programs that mishandled error conditions. Do any of your coworker's programs write records to already full data sets without giving anyone a clue? How many of you have received phone calls at obscure hours by some poor user wondering exactly what is going on in this $STDLIST? What percentage of operators do you think know how to read an octal stack dump and then fix the problem? The cost of recovering from a program that kept on running when it should have stopped because something was wrong can be staggering.

Error handling within a computer programs and JCL means recognizing error conditions and taking appropriate action. There are generally three courses of action that can be taken when an error is detected:

A. Recover from the error,
B. Print an error message and abort the program,
C. Ignore the error entirely.

All too often programs take Action C. This can lead to countless headaches and nightmares for users and programmers alike, and is rarely the best way to handle an error condition.

Action B, print an error message and abort is an acceptable method of handling errors; however, it is often used as a copout by lazy programmers. Unfortunately for most of us, this is the method used by MPE when it encounters an error: Print a system failure message on the console, and die.

Action A, recover from the error, is often the best method of error handling, but it is also the most difficult and costliest one to implement. It would be unreasonable to always attempt to recover from error conditions. If your program can't open the data base, it's tough to recover, so print an error message and abort. Batch programs should abort more often than online programs. Often an online program should print an error message to the user and let the user decide what to do next. You wouldn't want your editor to abort if you tried to Text in a file that you misspelled.

The key to good error handling is to detect the error as soon as it occurs. Whenever you wait, assuming any errors will get detected down the road, you run the risk not being able to figure out the cause of the problem or an even worse fate of never discovering there was a problem until it's too late.

As an example: What if a program that writes to a database, doesn't check to see if the DBPUT worked? (As I've seen before) If it's an online program, the user may simply keep entering data for hours without knowing that all of her bits are going into the great bit bucket in the sky. If it's a batch program, then maybe hundreds of honest hard-working employees mysteriously won't get paychecks on Friday.

Error handling within programs

The first step to handling an error condition is detecting it. The second step is for the program to decide what to do with it: recover, abort, or ignore.

Detecting errors

For example: If your program is reading down a chain in an Image detail data set when the DBGET fails, what should the program do? Your program should probably be able to recover if the error is an end-of-chain error, but should probably abort with an error message if it's any other Image error. Your program should check for an error condition after every system procedure call. It cannot be stressed enough, how important it is to check for an error condition after EVERY system procedure call. Remember, the sooner the error is detected, the better off you and everyone else will be.

Processing Errors

Whenever a program makes a system procedure or intrinsic call, the following steps should be taken after checking for an error to ensure proper error handling:

- If no error occurred, then continue processing
- If an error occurred:
    1. Retrieve error number
    2. Determine if error is recoverable.

       If recoverable, recover.

       If error is not recoverable:

        1. Retrieve and print error message based on error number.
        2. Abort the program, if appropriate.

When to Abort

When should the program abort and when should the program simply print an error message and then continue processing?

1. All severe errors should abort the program.
2. Errors within a batch program should abort.

3. Errors within an online program should print a message and continue if possible.

4. Programs that may run batch or online SHOULD CHECK whether they are being in batch, or online using either the WHO or FRELATE intrinsic and abort or continue accordingly. This is where many programs have problems.

File System Errors

File system errors should be detected by checking the condition code after every file system intrinsic call. The condition code is part of the hardware status word and is set by every file system intrinsic. The condition code can have one of three different values:

CCE - Condition Code Equal. This means that the intrinsic call was successful.

CCG - Condition Code Greater than. This means that a warning condition occurred, and that the intrinsic call may or may not have worked, depending upon which intrinsic was called. Check the intrinsics manual.

CCL - Condition Code Less than. This means that an error condition occurred and that the intrinsic failed.

Checking Condition Code in SPL

In SPL, check the condition code by simply using a relational operator with no expression. The condition code must be checked immediately after the intrinsic call. In addition, be careful not to assign the return value of an intrinsic call into an indexed array because this will destroy the condition code returned by the intrinsic call.

Some examples:

```
LEN := FREAD(FNUM, BUF, BUFLEN);
IF = THEN
   PROCESS'RECORD
ELSE
   IF > THEN  << END OF FILE REACHED >>
      END'OF'FILE
   ELSE  << SOME OTHER ERROR >>
      FILE'SYSTEM'ERROR;

FNUM := FOPEN(FILENAME, FOPTS, AOPTS);
IF < THEN
   FILE'OPEN'ERROR;
<< WE DO NOT NEED TO CHECK CCG, BECAUSE
   FOPEN DOESN'T RETURN IT >>
```

Do **NOT** do the following:

```
FILENUMBER(N) := FOPEN(FNAME, 3);
```

The array index N, will destroy the condition code.

```
BUFLENGTH := FREAD(FILENUM, BUFFER, LEN);
NUMREADS := NUMREADS + 1;
IF <> THEN   << READ FAILED >>
   HANDLE'READERROR;
```

This will not work, the statement after the FREAD will destroy the condition code.

Checking Condition Code in PASCAL

In Pascal the condition code may be checked by using the CCODE function. The CCODE function works as if it were a local variable to the current procedure that is set each time an intrinsic is called. CCODE may be checked any time before the next intrinsic call within the same procedure. Unlike SPL, the condition code does not need to be checked immediately after the intrinsic call, and you may assign the result of and intrinsic call into an array. The CCODE function returns the following values: 0, for CCG, 1, for CCL, and 2 for CCE. It often helps to use a "const" statement at the beginning of your program defining these three values. Example:

```
LEN := FREAD(FNUM, BUF, BUFLEN);
IF CCODE = 2 THEN  (*  READ WAS OK  *)
   PROCESS_RECORD
ELSE IF CCODE = 0 THEN
   DO_END_OF_FILE
ELSE  (*  CCODE MUST BE 1  *)
   FILE_SYSTEM_ERROR;
```

Checking Condition Code in COBOL

You can only check condition codes in COBOL II. You must define the name of your condition code variable within the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION. The COBOL condition code is similar to the SPL condition code in that you must check it immediately after the intrinsic call and may not assign the result of the intrinsic call to an indexed variable. The condition code variable may only be compared with zero. Condition code equal to zero means CCE, less than zero CCL, and greater than zero CCG. Example:

```
ENVIRONMENT DIVISION.
SPECIAL-NAMES.
   CONDITION-CODE IS CONDCODE.
```

```
PROCEDURE DIVISION.
.
CALL INTRINSIC "FREAD"      USING FNUM, BUFFER,
BUFFERLEN
                            GIVING BYTESREAD.
IF CONDCODE > 0 THEN
   PERFORM END-OF-FILE
ELSE
   IF CONDCODE < 0 THEN
      PERFORM HANDLE-FILE-ERROR
   ELSE
      PERFORM PROCESS-RECORD.
```

Printing Error Messages

Whenever your program detects a file system error that it does not know how to recover from, it should print an error message and stop. There are several intrinsics that facilitate this.

FCHECK Intrinsic

The FCHECK intrinsic is used to request the file system error that has most recently occurred. The first two parameters to FCHECK are:

- the file number of the file on which the error occurred
- the error number to be returned to the program.

If an error occurred during an FOPEN intrinsic, a value of zero should be passed to FCHECK as the file number. Be careful when using FCHECK because there is an unfortunate ambiguity with file system error numbers. A file system error of zero can mean one of three things occurred:

- The end of file was reached.

- There was **no** file system error.

- In rare circumstances a file system error occurred, but the system did not set the internal error number for FCHECK to retrieve.

Other intrinsics

The FERRMSG intrinsic is used to translate the file system error number into an error message. This message is usually more helpful than simply printing the file system error number.

PRINTFILEINFO prints a file tombstone and is another intrinsic that is often called when a file system error is detected. EDITOR, FCOPY and other programs call PRINTFILEINFO when they discover a problem.

The undocumented GENMSGU intrinsic can be used to print out a file system error message. It has two single word integer parameters passed by value. The first parameter is the message set number from CATALOG.PUB.SYS, and the second parameter is the error message number. Use message set 8 for file system errors. (Set number 2 can be used for MPE errors returned by the COMMAND intrinsic.)

Image Subsystem Errors

Image error handling is a bit easier than the file system. After every Image call, check the first word of the status array. If the first word is zero, then what you tried to do worked; otherwise it didn't. If the program understands the error number, such as 15 for end-of-chain, the program should be able to take appropriate recovery. If the program doesn't understand the error number, it should go into error mode. There are basically two ways to handle Image errors:

- Call DBERROR to get the Image error message, print the message and then resume with the program.
- Call DBEXPLAIN to print all pertinent Image error information, then abort the program.

VPLUS error handling is similar to Image error handling. After every VPLUS procedure call check the first word of the COM area. If the first word is zero then everything is OK, otherwise you've got problems.

Usually, the program should call VERRMSG to get the error message, then should zero the status word, then call VPUTWINDOW to put the error message in the VPLUS window. If the program decides to abort, the program should either call VCLOSETERM before printing any error messages or it should do the following:

- Call the FCONTROL intrinsic using the 49th word of the VPLUS COMAREA as the file number and 12 as the control code to turn the terminal echo back on for the user.

- Escape sequences should also be printed to the terminal to turn format mode off (<esc>X), turn block mode off (<esc>&k0B), turn memory lock off (<esc>m), and home down (<esc>F).

Other Intrinsics

There are many other intrinsics, virtually all of which return a status code via the condition code, or return an error number or both. Unfortunately, MPE does not provide any mechanism for converting an error number into error message. The program must either convert the error number itself, or simply print the error number out as part of the error message.

Aborting a program because of an error

When a program encounters an error that it cannot recover from, then it should abort by going through a special abort procedure. The abort procedure should do the following:

1. Print the file system/IMAGE/VPLUS error message that caused the abort.

2. Print an error message specific to the program and location within the program that detected the error. Ideally, no two error messages detected from different points within a program should print the same message.

3. Terminate the program by calling the QUIT intrinsic, do not use STOP RUN in COBOL programs or the TERMINATE intrinsic to abort a program.

As an alternative to calling QUIT, the program may set the high order bit of the system JCW by calling the SETJCW with a negative value, then call the TERMINATE intrinsic. Doing this causes the program to end in an error state. The QUIT intrinsic does this automatically for the user. Note that the JCW may also be set with the PUTJCW intrinsic, or the COMMAND intrinsic with a SETJCW command.

Error Handling Within Job Streams

Error handling within job streams is often done carelessly. The :CONTINUE command should only be used when necessary. Wanton placement of :CONTINUE commands within a job can be hazardous to your health.

For example, many programmers make the mistake of putting :CONTINUE commands before :PURGE commands. This is almost always incorrect. If the :PURGE command attempts to purge a file that does not exist, then a WARNING is issued, not an error. No :CONTINUE command is necessary for the job to continue. In most circumstances in which the :PURGE command fails, the job *should* stop running because there is a problem, such as the file being accessed by another program.

Using JCW

The system defined job control words (JCWs) CIERROR and JCW can be used very successfully within job streams. These JCWs are managed by both the system and the user.

The JCW called CIERROR is set by the MPE whenever an error or warning occurs with an MPE command. It is set to the command interpreter error number if there is a problem, otherwise its value is not changed. Unfortunately, there is no way to tell by looking at the number whether it is an error or a warning. There are many uses of CIERROR. This example will check if a file exists:

```
:SETJCW CIERROR = 0
:CONTINUE
:LISTF MYFILE;$NULL
:IF CIERROR = 0 THEN
:  TELLOP  MYFILE is alive and well.
:ELSE
: TELLOP  HELP!  MYFILE is not there!
:ENDIF
```

The JCW called JCW is used to help the job determine what happened with a program run. When a program is run, MPE sets JCW to zero.

If the program is successful, then JCW may be set to a value from zero to 16383 to indicate its success.

If a warning occurs during the program, then the program may set JCW to a value from 16384 to 32767.

If an error occurs during the program, then the program should set the value of JCW to a value from 32768 to 65535.

If a program terminates with the value of JCW from 32768 to 65535, then MPE will consider that the program has terminated abnormally and generate command interpreter error 989. A job would have had to have a :CONTINUE command before the run of the program for the job to continue.

Aborting a Job

When a job encounters a nonrecoverable error, it should abort. This is most easily done by doing nothing because MPE will handle it for you if you don't use :CONTINUEs and your programs abort properly. By using this method, an operator can always tell if a job succeeded or failed by taking a quick glance at the bottom of the $STDLIST.

A perhaps better way of aborting a job is to use :CONTINUE commands before each run of a program, then use JCW checking to determine if the program succeeded. If ever a program within the job fails, use the :TELLOP command to notify the operator that the job has failed. Example below:

```
!JOB AR1002J,BATCH.AR
!SETJCW ERROR, 0
!CONTINUE
!RUN AR10021P.PROG.AR
!IF JCW >= FATAL THEN
!   SETJCW ERROR = 1
!ENDIF
!IF ERROR = 0 THEN
!   CONTINUE
!   RUN AR10022P.PROG.AR
!ENDIF
!IF ERROR = 0 AND JCW >= FATAL THEN
!   SETJCW ERROR = 2
!ENDIF
!IF ERROR = 0 THEN
!   CONTINUE
!   RUN AR10023P.PROG.AR
!ENDIF
!IF ERROR = 0 AND JCW >= FATAL THEN
!   SETJCW ERROR = 3
!ENDIF
!
!IF ERROR <> 0 THEN
!   TELLOP **************************************
!   TELLOP **      JOB                         **
!   TELLOP **      AR1002J                     **
!   TELLOP **      FAILED!                     **
!    TELLOP **************************************
!    SHOWJCW
!    ABORT
!ELSE
!    TELLOP Job AR1002J completed successfully.
!ENDIF
!EOJ
```

This job stream has several noteable features:

1. It is written is such a way that can be easily read and modified. The :IF statements never get more than one level deep.

2. When the job fails an easily identifiable message is printed on the console for the operator. Presumably, the program that failed has printed an error message on the $STDLIST that will enable the operator or programmer to pinpoint the problem.

3. By using the :ABORT command to terminate the job stream, MPE will stop processing the job and print a message at the bottom of the $STDLIST that the operator can easily recognize as a failed job. Note that the :ABORT command is not intended for this purpose but serves nicely.

4. When the job succeeds, a simple message is printed to the console and the job terminates with an :EOJ command, a signal to the operator reading the $STDLIST that the job was successful.

Error Handling Within UDCs

Let's not forgot abort user-defined commands. Basically, error handling within UDCs is the same as it is within job streams. The :CONTINUE command performs the same function in UDCs as it does within jobs, it allows the rest of the UDC to complete if one of the commands encounters an error. An example of this would a copy UDC as follows:

```
COPY FROMFILE,TOFILE
FILE INPUT = !FROMFILE
FILE OUTPUT = !TOFILE
CONTINUE
RUN MYCOPY
RESET INPUT
RESET OUTPUT
****
```

This UDC would make certain that the INPUT and OUTPUT file equations were reset regardless of the success of the program MYCOPY.

Programatic error handling:

1. Check the condition code or status word after EVERY intrinsic or system procedure call.

2. Retrieve and display system error message whenever appropriate.

3. When a program running within a job cannot recover from an error, it should always abort, even if it is an online program.

4. When a program running online detects an error, print a message and continue if possible, otherwise abort.

5. When aborting a program, always print a unique error message in addition to the system error message, and always set the job control word JCW to a fatal value.

Error handling within jobs:

1. Do not abuse the :CONTINUE command. If there is a problem with the job, it should abort.

2. Use the :CONTINUE command before each command that could fail; then use JCW checking to ensure it succeeded.

3. If a job fails, print an easily recognizable message on the console.

4. Make certain that an operator can easily determine if a job failed by glancing at the $STDLIST.

5. Successful jobs should always end with an :EOJ command.

Conclusion

For a data processing shop to run as smoothly as possible, the programs and job streams need to be written so that errors get detected as soon as possible after they happen. Once an error is detected, appropriate handling of the error is imperative, whether it be a simple error message or a program abort.

By following the guidelines in described in this paper, you will be well on the road to DP pie in the sky.

Migration Made Easy

Victoria Shoemaker
Taurus Software, Inc.
770 Welch Road Suite 3A
Palo Alto, CA 94304

So you have purchased a Hewlett-Packard Precision Architechture machine (HPPA/Spectrum/Series 900). Congratulations! So you are going to migrate. What does migration mean, exactly? How does one migrate anyhow? Oh...you haven't thought that all the way through yet. Hmm, maybe you should start planning your migration now.

This paper will take you through the steps you will need to take to make your migration effort to the HPPA machines complete and successful. There are seven steps:

- Education
- Analysis of Existing Applications
- Developing of a Migration Plan
- MPE/V Conversions
- Installation of HPPA machines
- Compatibility Mode Operation
- Migration to Native Mode Operation

Each of these steps will be discussed in detail.

Education

The HPPA machine is completely different from the classic HP3000. There are very few similarities in their hardware architectures. HP has gone to great pains to ensure that old job streams and programs will run on the HPPA machines, but all of the classic HP3000 hardware emulation is done via software on the HPPA machine. Underneath the software are two different machines. For all of the differences between the two computers, they might have been put out by two different manufacturers.

The point I am trying to make, is that what makes the classic HP3000 hum, does not make the HPPA machine hum. In order to take advantage of the performance gains projected by HP, you are going to have to migrate your software to native mode. All the knowledge you have gained about stacks, PCBs, MPE tables and such are of little help. You must learn about this new machine from scratch. So, step one, get educated.

There are a number of different ways to learn about the HPPA machines: user groups, manuals, FASTLANE consulting, HP classes and books. Some of the topics you are going to want to make yourself familiar with are as follows:

- Native Mode vs Compatibility Mode. Actually this is a very simple lesson. Native mode operation means that it is using the native instruction set (RISC) of the HPPA machine. Compatibility mode means that software is emulating the native mode operation of a classic or MPE/V based machine. In order to take advantage of the performance gains of the HPPA machines over the classic HP3000s, you are going to have to convert your applications to native mode.

- System Management. The HPPA machines don't use SYSDUMP. They have a new configuration manager called SYSGEN. It is very different from SYSDUMP. The HPPA machines don't use OPT or any of the other performance monitoring tools you are used to. The HPPA machines handle terminal connection differently than you are used to.

- Operational Changes. From the simplest thing, bringing up the machine to private volumes, the HPPA machine is different. You are going to have to learn how to do everything over again.

- New/Changed Commands. There a number of new commands in MPE which is now called MPE XL. The command resolution and syntax have changed dramatically. All the changes are very positive. You are going to want to take advantages of the positive changes immediately.

- Data Migration. The classic HP3000 is a 16 bit machine. The HPPA machine is a 32 bit machine. This spells data migration. You need to know how this will affect your data files and program's internal data structures.

- Programming Environment. There are a number of new programming tools available including a symbolic debugger. You are going to want familiarize yourself with its features before you begin conversion to native mode.

- Programming Languages Issues. Each programming language has its own set of challenges when moving to native mode that you are going to have to be familiar with before you begin. We will discuss some of those issues later in this paper.

- New/Changed Intrinsics. A number of intrinsics have changed. In addition, a number of new intrinsics have been added to take advantage of some of the new features in MPE XL. I am sure that you are going to want to take advantage of these when converting to native mode.

As you can see the list is fairly impressive of some of the basic things you are going to need to know before you begin your migration. So, how do you go about this? Easy, there are number of different ways, some even free!

1. User's Groups. There are a number of different speakers talking at this conference alone about migration issues. Attend their talks. Read their papers. Ask questions. Some of the users have already gone through migration experiences. Learn from their successes and mistakes. HP has released early bird sites and FASTSTART companies from their confidentiality agreements. These companies are a wealth of knowledge.

2. Manuals. HP has published a number of manuals to help companies have a successful migration experience. Some of the titles are as follows: *Migration Process Guide*, *Programmer's Skills Migration Guide*, and *COBOL Migration Guide*. I found the first two listed very helpful in getting an overall picture of what is involved in migration. Remember when you read these manuals, read them as Alfredo Rego recommends, like love letters.

3. Consulting. Both HP and independent consultants are offering migration assistance. I have not used either one. HP's consulting is called FASTLANE consulting. It includes a one day class taught at your site and a migration planning meeting. Independent consulting seems to be varied. As with any consulting, make sure you are dealing with experts.

4. Classes. HP offers two classes to help you in migration: system management, and programmer's class. The system management class is three days long and deals with the new SYSGEN and other system utility software. The programmer's class is seven days long and is for programmers who plan to assist in the migration effort. It covers language specific information and the symbolic debugger.

5. Books. There is a book covering Spectrum issues called *Beyond RISC! An Essential Guide To Hewlett-Packard Precision Architecture*. I am sure it is

worth reading. It is available through Software Research Northwest, Inc.

Application Analysis

Once you are educated, you are now ready to review your applications for any known migration issues. Some obvious gotchas are:

- FORTRAN. There is no FORTRAN/66 native mode compiler on the HPPA machines. You must first convert your FORTRAN/66 to FORTRAN/77. I don't code in FORTRAN, but from what I understand this is not a trivial task.
- COBOL. All COBOL/66 programs need to be converted to COBOL/II.
- BASIC. There is no BASIC/3000 native mode compiler on the HPPA machines. Business Basic has a native mode compiler.
- SPL. There is no SPL native mode compiler on HPPA. A third-party compiler, SPLASH compiles SPL to native mode.
- Privilege Mode. Remember when HP said "Don't use privilege mode." Well now you know why. MPE XL is a completely different operating system internally from MPE/V. This means there is 50-50 chance that your privileged mode programs will need to be changed before they will run in native mode.
- Floating Point. The HPPA machines uses IEEE floating point arithmetic. The classic HP3000s used their own brand of floating point arithmetic. If you use real numbers this could be an issue for you. For your information, COBOL does not use floating point, PASCAL and SPL have floating point facilities, and FORTRAN uses floating point extensively.

Other than these obvious issues, there a number of specific language and intrinsic related issues. Luckily for you, HP has put together a migration tool package. The price is right, $100. The package includes two programs: RTM and OCA.

RTM (Run Time Monitor) is a utility program designed to help you identify areas within your MPE/V programs that could be a problem when ported to MPE XL. It logs calls to MPE/V intrinsics which have been changed or are not supported on MPE XL as they happen. The logging is controlled by RTMSYS.PUB.SYS. The logging reports are printed using RTMREP.PUB.SYS.

OCA (Object Code Analyzer) is a utility program which scans program and SL files for potential problems. OCA scans a program for "problem" intrinsic calls. The advantage to using OCA is that you are able to obtain immediate results without the need for logging. OCA may not be as accurate as RTM because it cannot always tell what parameters an intrinsic is called with.

Both of these migration tools run on MPE/V based systems. You can run them before beginning migration. Don't forget to analyze both internally written applications and third-party solutions. Your migration effort will be affected by both types of applications.

At the end of this phase of migration, you should have a list of applications and any potential migration problems. With this list, you will be able to begin the next phase of migration, planning.

Planning

Planning is the single most important element of your migration. Regardless of how many applications run in your shop, how many machines you have, how much third party software you run – your migration's success depends on how well you have planned it out. Spend the time to plan. It will pay off.

The first thing you will need to decide is which of your applications are worth migrating to native mode operation. Some factors that may enter into your decision are:

- How often does this application run? If the application only runs once a year, does it really need the performance gains provided by converting it to native mode?

- How much trouble will it be to convert this application to native mode? If you have a frequently called FORTRAN/66 routine, is it really worth converting it first to FORTRAN/77 and then to native mode?

- Do you have the source code for this application? Obviously for third-party solutions, you probably are going to have rely on the vendor for native mode solutions. For contributed routines, you may have to let them run in compatibility mode for lack of a better solution.

- Is this a high volume application? If the application processes a great deal of transactions, it may be worth converting for the increase in transaction throughput.

- Do you have to maintain compatibility with MPE/V based machines? If this is true, then you may not want to take advantage of the native mode compilers. Native mode object code will not run on MPE/V based machines. There is a middle ground for such applications. OCT (Object Code Translator) converts an MPE/V object module and adds native mode instructions to the end of the program file. This way the program can run on both MPE/V and MPE XL programs. OCTed programs do not run quite as fast as native mode programs, but it is a good compromise for programs which need to run on both types of systems.

Once you have determined which of your applications will be migrated to native mode, you must develop a migration plan for each application. HP can help with the development of this plan through their FASTLANE consulting.

Along with your application specific plans, you will need an overall strategy for handling privilege mode programs/routines. Some of your privilege mode programs and routines will have to be rewritten entirely. Some of the functions can easily be replaced with complimentary functions on the HPPA machines.

Regardless of the situation, you must decide if you wish to migrate privilege mode programs to native mode or let them run in compatibility mode.

The last consideration is site planning. During your migration, there will be a time during which both your classic and HPPA machine will share the same room. This means you need to prepare your computer room for two scenarios: parallel operation (HPPA and classic) and HPPA. During parallel operations, you will need power, air conditioning, and peripherals for two machines. You must plan the additional strains on your computer room. To help in your planning, if you were to just move your existing applications directly onto a HPPA machines, you would need an additional 20% disc space. Make sure you have enough disc ordered for your machine.

So to review, you are going to need plans for the following:

- Application specific migration
- Privilege mode strategy
- Site planning.

MPE/V Migration

Prior to receiving your HPPA machine, there are a number of tasks that can and should be completed on the MPE/V base machine. A checklist of these tasks follows.

1. Upgrade to the latest release of MPE/V. MPE XL's base release was UBdelta4. This is the starting point for MPE XL. It is best it you get on this release of MPE/V before migrating.

2. Get on the latest release of your programming languages. If you are using BASIC/3000 convert your programs to Business BASIC. If you are using FORTRAN/66 convert your programs to FORTRAN/77. If you are using COBOL/66 convert your programs to COBOL/II. If you are using SPL, either convert your programs to another language or buy SPLASH. For all compilers, get to the current version of the compiler.

3. Call INTRINSIC. For all system intrinsics, the new format of the call on MPE XL is CALL INTRINSIC. Change all COBOL programs to use this format.

4. Block Mode. If you developed your own block mode terminal routines, convert them to use the new standard block mode routines: VTURNON/OFF, VPRINTSCREEN, and VBLOCKREAD/WRITE.

5. FOPENs of LDEVs are not supported. The DTC does not assigned a fixed LDEV numbers for devices. Opening a specific LDEV may not produce the same result on MPE XL as it does on MPE/V. It is possible to assign fixed LDEV numbers, but this is not the default configuration.

6. UDC Conversion. You will want to change UDCs which have the same name as the new MPE XL commands. You will also be able to omit 70% of your system-wide logon UDCs which run programs in PUB.SYS because of the new implied run and HPPATH variable.

7. SYSDUMP does not exist. You will want to convert your job streams which use SYSDUMP to use STORE. The commands FULLBACKUP and PARTBACKUP are not supported on MPE XL.

8. Peripherals. Not all the peripherals that are supported on MPE/V based machines are supported on HPPA machines including paper tape, and cartridge tape drives. Get a list from your HP CE.

Installation

The awaited day finally arrives! Your HPPA machine finally arrives! Well, here is the bad news, you are probably going to need some help getting everything configured and setup. As you recall, SYSDUMP does not exist on the HPPA machines. It has been replaced by SYSGEN.

When migrating from MPE/V to MPE XL you will use DIRMIG to create your directory, accounting structure, user logging parameters, RIN table, and private volume information. SYSGEN is the device configuration dialogue. In addition, there is another utility, NMCONFIG, used to configure your LAN and DTCs.

Well, if you are like me, these are all new and it would be nice if someone was there to help you.

Now the good news, once your accounting structure and other internal structures are in place, you can just restore your applications and they will just run! Compatibility mode operation just works! So let's look at our next step compatibility mode operation.

Compatibility Mode

As you recall compatibility mode is the term HP has chosen to mean running a program that contains instructions for the classic HP3000. When a compatibility mode program runs on MPE XL, the instructions are actually emulated by the emulation software.

The purpose of running your programs in compatibility mode is to ensure that the operation of these programs on MPE XL produces the exact same results as it did on MPE/V. Almost all of programs work exactly the same.

Typically compatibility mode testing continues through one complete accounting cycle. During this testing, your programmers can begin familiarizing themselves with the new features of MPE XL and the feel of the HPPA machines.

Migration to Native Mode Operation

During this phase of your migration, you are going to implement the application migration plans. This is an appropriate time for your programmers to attend the programmer training course and to learn the new debugger. They will also need to familiarize themselves with switch-stubs, which is a technique for switching between native mode and compatibility modewithin a program.

Once the conversion to native mode is complete. You can address yourself to optimizing the performance of your programs. There are several things which can affect performance of your applications:

- Mixed mode applications. Mixed mode applications run slower than native mode applications because of time required to go through switch-stubs.
- OCT applications. Programs which have been run through OCT will not run as fast as those which have been recompiled using a native mode compiler. This is due to the literal translation of MPE/V instructions.
- Extra data segments. Extra data segments are an MPE/V data structure and should be converted to mapped files for extra performance.
- Use of KSAM/RIO/CIR/MSG files. These file structures are supported using compatibility mode file intrinsics only. Because the intrinsics are supported in compatibility mode only, they will be inherently slower than other file structures.
- Word alignment. You can expect increased performance for programs that align their internal data structures to 32 bit word boundaries.

As with the classic HP3000, we are going to have to experiment with what exactly makes this machine perform. I am sure that over the next few years we will all be learning about the performance tuning techniques for the HPPA machines.

Summary

The most important lesson from this paper is two-fold: get educated and plan your migration. I believe that if you learn as much as you can before you get started and then plan your migration, you cannot have anything but a successful migration. Good luck to you!

I want to express my gratitude to John Bria of Hewlett-Packard for his help in developing this paper. It is his talk that provided me with the basis for this paper. Without his help, this paper would still be blank.

TITLE: The Face of Data Processing

AUTHOR: E. R. Simmons, Ph.D.

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 0035

MPE/XL Variables and Command Files

Brett Clemons
Softwarewizardry, Inc.
Tampa, FL. 33615

Abstract. With the introduction of Hewlett-Packard's Precision Architecture came a new command interpreter - MPE/XL. One of the most powerful new enhancements to the basic MPE command interpreter is a new type of file - the command file. Command files can be used in place of, or in conjunction with, UDC's to produce a versatile tool at the disposal of the veteran or novice programmer alike. The most exciting aspect is the introduction of variables, or 'vars' to allow you the ability to customize your command interpreter to perform very sophisticated tasks with a single keystroke. Examples include compiles of wildcard filesets, file purges or compressions, as well as analysis of files in a wildcard fileset. Variables, as well as a few new MPE/XL commands, allow the programmer to place control of the command interpreter where it belongs - with the user.

I. Introduction.

The evolution of the MPE command interpreter has been slow and gradual. Existing commands were enhanced, but no major change occurred in the basic 'core' of the command interpreter until recently. With the advent of MPE/XL Precision Architecture command interpreter, something new has evolved from the old command interpreter. For the purposes of this paper, MPE/XL shall be termed a command interpreter, which more technically expresses what it is.

Hewlett-Packard's new approach to command interpreters has breathed new life into the old MPE command interpreter, bringing with it something which allows for the command interpreter itself to be used as an extension the programming languages, or as a programming language itself. Many of the elements found in fundamental programming languages are present in the new MPE/XL command interpreter. For example, variables allow you to store temporary information in any of the formats that may be found in a programming language. In addition, testing of variables in Boolean expressions allow for a much expanded analysis of conditions that may occur external to the programs. Also, several commands have been added to allow recursive performance of a block of commands. Previously, a set of commands that you wanted to execute could only be stored in a file, and :SETCATALOGed. This file contained User Defined Commands or UDCs. Command files, which are meant to expand upon UDC's, and not replace them, allow the execution of a block of commands - and command files are easy to set up and maintain.

This paper shall also investigate more advanced topics using command files and variables, as well as some new MPE/XL commands that add flavor to command files.

II. Variables - the new JCWs

Under the MPE command interpreter, the only way of storing temporary information external to a program was in a file, or with Job Control Words (JCWs). JCWs are not eliminated with MPE/XL, but are enhanced with a new form of storage called variables.

Variables names identify the variable referenced in the commands. Variable names start with an alphabetic character or underscore (_) character, and contain from one to 255 characters. Of course, variable names must be unique.

There are three several different types of variables. Variable types are defined with the use of the new MPE/XL command SETVAR. SETVAR allows the definition of a variable name explicitly with the SETVAR command; the variable type is implicit with the initial value assigned to the variable. SETVAR is also used to redefine the value of an existing variable. The basic type of variables are Boolean, Integer and Strings. The type of a variable is set at time of definition, depending upon the information to be stored in the variable. For example, to define a boolean variable, use the following command construct :

```
                         TRUE
:SETVAR varname,{             }
                         FALSE
```

The second type of variable, integer variables, allow you to define variables containing numeric values, and are defined at definition time with a

```
:SETVAR varname,integer-value.
```

The third type of variable, string variables, are defined with a

```
:SETVAR varname, 'string-value '.
```

String variables may contain any valid string, from none to 256 valid alphanumeric characters.

Variables are removed with the DELETEVAR command. When a variable is defined it retains its characteristics initially defined with the SETVAR command until the DELETEVAR command is used to remove the variable from the user's job or session. Its form is

```
            :DELETEVAR varset
```

and as shown in the syntax, may contain wildcards (@, #,etc) to specify a set of variables to be deleted.

The final variable command is the SHOWVAR command. SHOWVAR lists variables to $STDLIST, and uses wildcards in the same way as the LISTF command does. This command has the format

```
            :SHOWVAR [varset]
```

If the varset parameter is omitted, only user-defined variables are displayed; if @ is used for varset, then all variables will be shown. Other wildcards may be used to list any subset of variables. For example,

```
            SAVE_@
            HP@
            SAVE_VAR_#@
```

are all valid subsets that may be used to represent a one or more variables that are defined.

Variables can be used in a variety of ways, most usefully in command files, but they are not excluded from use in UDCs or jobs.

III. Using Variables

When MPE/XL parses a command line, the Expression Evaluator (a part of MPE/XL) first looks for variables in the command line. The process of substituting a variable's value in a command line is called dereferencing; dereferencing takes precedence over all other operations in an MPE/XL command line, including the recognition of the command name itself! There are two methods of dereferencing variables. The first, implicit dereferencing, is where the variable name is placed in the command line and MPE/XL substitutes the value of the variable before parsing the command line. For example, when the following commands are entered

```
            :SETVAR INDEX1 17
            :IF INDEX1 <= 20 THEN
```

the expression evaluator responds with the familiar

```
           *** EXPRESSION TRUE
```

and will continue executing commands until a matching ENDIF is encountered, because the value of INDEX1 is 17, which is less than 20.

In the second case of dereferencing, explicit dereferencing, the variable name is preceded by one or more exclamation points (!) which directs MPE/XL to substitute the value of the variables represented at that place in the command line. The most important thing about explicit dereferencing is that MPE/XL substitutes a pair of exclamation points with a single exclamation point, and a single exclamation point forces MPE/XL to perform value substitution. For example

```
:SETVAR var1,'stringvalue'
:SETVAR var2,'!!var1'
:SHOWVAR var2
 VAR2 = !VAR1
:ECHO !var2
 stringvalue
```

This example also shows how variables can be set to any valid expression, including other variables. However, expression types may not be mixed as in

```
:SETVAR VAR_VALUE 17 + 'foo'
```

The Expression Evaluator would flag this command as an error.

Functions allow the manipulation of text in command line. Again, the part of MPE/XL responsible for evaluation of the results of functions is the Expression Evaluator. There are several functions, just a few of which are

len - string length function
str - string extraction
ups - upshift string

These powerful functions allow complicated variables to be built and examined in command files.

IV. Global Variables

Variables may be defined by the user, but MPE/XL maintains its own set of variables. All global variables start with HP (what else?), except CIERROR and JCW, which, under the old command interpreters, were JCW names. Global variables may be read only or read/write, and may not be deleted.

Jobs and sessions begin with the Global variables defined with initial values. Global variables allow for lots of things in the user environment to be tested or displayed:

HPJOBLIMIT - is the system's job limit
HPMONTH - is the month according to MPE/XL
HPCIERRMSG - is the error message that corresponds to another variable, CIERROR

HPMSGFENCE - allows MPE/XL error messages to
be suppressed.

One global variable of particular interest is HPPATH. This variable defines the 'path' that MPE/XL will search for command files (and program files, too). Initially, it has the value !hpgroup,pub,pub.sys. This means that MPE/XL will first look for command files or programs in your current group (!hpgroup), the the public group of your logon account (pub) and last in pub.sys.

V. Defining command files

Ever since MPE III, users have had the ability to define their own commands. The way this was accomplished in the past was through the User Defined Commands (UDC's), which are familiar to most users. However, a new way of storing user commands was introduced with MPE/XL. This is the command file. Command files and UDCs are both very similar and very different. One of the main differences is that multiple UDCs are defined in a single MPE file and the SETCATALOG command is used to invoke the UDCs at the appropriate level - system, account or user. On the other hand, a command file represents a single user command, and that command is invoked by virtue of the name of the file itself. In other words, to execute the commands in the file COMMFILE, merely say

```
:COMMFILE
```

Kind of simple, isn't it? Optional parameters may be added after the command file name, but must follow the rules outlined in the header portion of the command file. Command file command lines may contain commands that are valid MPE/XL commands, or user commands in UDCs or command files. Command lines may even contain the name of program files, since with MPE/XL, the :RUN command is implied if the command name is a valid program file. In MPE/XL, user commands may even invoke themselves (unless disallowed with the OPTION NORECURSION).

Command files are much more simple to create than UDCs. Simply use you favorite brand of text editor, and place commands in an MPE file. To execute command files in a different group or account, you must have the appropriate access to that file, in the same fashion as UDC's.

VI. Command File Structure

The basic structure of command files is simple,and very much like UDCs. The first line is the optional parameter line. Up to 63 parameters may be specified. The syntax for this line is

```
parm parm1[='defaultvalue'],...parm63
```

The second line is the options line. These lines specify how the command file will be accessed or the basic environment the command file will operate in:

```
          option option1[,option2[,...]]
```

Some available options are BREAK/NOBREAK, HELP/NOHELP, etc. Remaining lines are command lines. All MPE/XL commands except :DO and :REDO are valid and may be used in user commands.

Parameters for command files are specified in one of two ways. New MPE/XL users will feel comfortable with the POSITIONAL parameter sequence, in with the parameters in the command are specified in the same sequence as in the PARM line of the command file. The other way is by using the KEYWORD construct. A user file (named COPYFILE) containing the lines

```
         parm filein='$stdin', fileout='$stdlist'
         fcopy from=!filein;to=!fileout;new
```

may be invoked by any of these valid user commands:

```
         :copyfile oldfile, b
         :copyfile oldfile b
         :copyfile fileout=b,filein=oldfile
```

VII. Programmatic Access to Command Files

One of the really nice things about command files (and also UDCs) is that with MPE/XL, all user commands may be invoked from programs. Those that cannot are those that have the OPTION NOPROGRAM specified in the header portion of the command file. Hewlett-Packard had the foresight to provide MPE/XL users with the new HPCICOMMAND intrinsic, which allows any valid user command to be invoked from a program. This powerful new intrinsic allows the distribution of programming power to the command interpreter, and will be discussed in a subsequent section. The consequences are that any command, MPE/XL or user, can be executed from a program.

VIII. New MPE/XL commands to support user commands

Several new commands were introduced in MPE/XL that allow maximum utilization of command files. Although command files can be written without the use of these commands, the power of MPE/XL really comes through when these are used.

The first of the new commands is the WHILE and the ENDWHILE commands, which have the the syntax

```
          :WHILE boolean-expression [DO]
                  < commands executed as long as condition
                      is true >
```

```
            :ENDWHILE
```

The WHILE and ENWHILE commands allow for multiple repetition of of a block of commands while a condition is true. As long as the condition is true, the commands will continue to be executed.

Another command of great use in commands files is the new COPY command. The COPY command is the much modified FCOPY command, with a much more simplistic syntax:

```
                                                     { ASK }
            :COPY [FROM=]fromfile[;TO=tofile] [;{ YES }]
                                                     { NO }
```

COPY perform a multi-record, no-buffered file copy of files much the same way that FCOPY does. There are some restrictions: fromfile and tofile may not be system defined files or spool files. The options determine whether the user is asked to replace the tofile if it exists - ASK will prompt the user; YES will replace the file and a NO option will leave the tofile intact if it exists.

The PRINT command is very similar to the COPY command. This command's syntax is

```
            :PRINT [FILE=]file
                 [;OUT = outfile ]
                 [;START = startrecord]
                 [;END = endrecord ]
                 [;PAGE = linesinapage ]
                 [;{UNN ]
                   {NUM}
```

Although primarily for printing files to $STDLIST, once you realize that any file may be specified for the file parameter (unlike the COPY command) and about any file may specified for the outfile, then the PRINT command is not unlike the FCOPY command except for real neat feature - it may called while in BREAK mode! PRINT and COPY are commands, and unlike FCOPY which is a program, may be used in BREAK mode.

The INPUT command allows interactive acceptance of variable values. The format for the INPUT command is

```
            :INPUT [NAME=]variable-name
                 [;PROMPT = promptstring]
                 [;WAIT = waitseconds ]
```

This command allows the changing or the creation of variables, and optionally will prompt the user with a prompt string, and wait a given number of seconds for the user to respond. If the user does not respond in that time, command file execution continues, but CIERROR will be set with a value of 9003.

Another really useful command, introduced previously, is the ECHO command which has the format

```
            :ECHO [message]
```

Echo does not perform implicit dereferencing, but instead requires explicit dereferencing of variable names. One thing to remember about the ECHO command is that a carriage return is always generated after the message is displayed, and if message is null, only a carriage return is generated.

Other MPE/XL commands are the CALC command (used to generate the result of an expression to $STDLIST and to HPRESULT), and the RETURN command (returns to previous level of Command file).

IX. Command File Examples

Our first example will be to use three existing variables to create a fourth. The three pre-existing variables are global variables and contain the year (last two digits), the month (digits, not name) and the day of the month, respectively. We use these to create a new variable, HPYYMMDD:

```
SETVAR HPYYMMDD,!HPYEAR * 10000
SETVAR HPYYMMDD,!HPYYMMDD + (!HPMONTH * 100)
SETVAR HPYYMMDD,!HPYYMMDD + !HPDATE
```

The next example uses the PAGE option of the PRINT command to print a file without pausing:

```
PARM FILENAME
PRINT !FILENAME;PAGE=0
```

This example shows the use of the parameter line, and the use of the parameter in the command file to cause the parameter value to be replaced with the value (REQUIRED) specified when the command was executed.

The next example shows the easy way to recover lost filespace by 'squeezing' the end of file to the file limit.

The following example is the command file SQUEEZE:

```
PARM FILENAME
COMMENT
COMMENT  This specifies that the command file will have ONE
COMMENT  required parameter - the filename to be squeezed.
COMMENT
IF NOT FINFO('!FILENAME',0) THEN
COMMENT
COMMENT  This line tests for the presence of the file
COMMENT  specified in the parm line. The FINFO is a function
COMMENT  requiring two parameters : the first the filename in
COMMENT  string form, and the second, the function number.
COMMENT  Zero as a parameter queries for the existence of a
COMMENT  file, and returns a Boolean result.
COMMENT
    ECHO !FILENAME does not exist - cannot squeeze
ELSE
COMMENT  ^
COMMENT  If the file does not exist, no need to even try any
COMMENT  of this stuff.
COMMENT
    SETVAR END_OF_FILE,![FINFO(!FILENAME,19)
COMMENT  ^
COMMENT  This command line will set a variable to the number
COMMENT  of records in the file (end of file) by using
COMMENT  the FINFO function with a parameter of 19.
COMMENT
    FILE NEWFILE;DISC=!END_OF_FILE;SAVE
COMMENT  ^
COMMENT  This command line will set a variable to the number
COMMENT  of records in the file (end of file) by using FINFO
COMMENT  with a parameter of 19.
COMMENT
    SETVAR CIERROR,0
    COPY !FILENAME,*NEWFILE;YES
COMMENT  ^
COMMENT  These lines prime CIERROR to zero and copy the file
COMMENT  from the oldfile to the newfile.
COMMENT
    IF CIERROR = 0 THEN
        PURGE !FILENAME
        IF CIERROR = 0 THEN
            RENAME NEWFILE,!FILENAME
        ELSE
            ECHO !FILENAME was not purged nor renamed
        ENDIF
    ELSE
        ECHO Copy of !FILENAME failed
    ENDIF
COMMENT  ^
COMMENT  If the COPY command succeeds, purge the oldfile and
COMMENT  rename the newly created file to the name of the old
COMMENT  file. Otherwise, tell the user the copy and rename
COMMENT  has failed.
COMMENT
ENDIF
COMMENT * end of the command file.
```

Now for a really good example of the power of command files. First, lets consider the mundane output from the LISTF command. Typically, it has the format that we are all familiar with:

```
ACCOUNT= TESTACCT    GROUP= TESTGRP

FILENAME  CODE  ------------LOGICAL RECORD-----------
                  SIZE  TYP        EOF       LIMIT R/B

DISCFILE           80B  FA         597         597   3
CRLFILE   NMRL    128W  FB         689         689   1
COBTEXT   EDTCT  1276B  VA         785         785   1
V23AXFRM  VFORM   256B  FA       27958       50000   1
W23AXWSP  TSR    1024B  FA           3          39   1
XL        NMXL    128W  FB       44260     4096000   1
Y23AXUSL  NMPRG   128W  FB        2031        2031   1
U00AAX89  USL     128W  FB         266        1023   1
```

Careful examination of the output will show that if we redirect this output to a disc file, we could use fileset wildcards in some command files. PURGESET is such a command file. It invokes another command file (XEQFILE), which itself invokes other command files. The result is a wildcard file purge.

The following example is the command file PURGESET.

```
PARM FILESET
COMMENT
COMMENT This command file will purge filesets. The
COMMENT only parameter is the desired fileset.
COMMENT
SETVAR SAVE_HPAUTOCONT,HPAUTOCONT
SETVAR SAVE_HPMSGFENCE,HPMSGFENCE
COMMENT
COMMENT In these lines, we save off the current value
COMMENT of HPAUTOCONT, the autocontinue var, and
COMMENT HPMSGFENCE, the var that determines if MPE/XL
COMMENT error messages are displayed.
COMMENT
SETVAR HPAUTOCONT,TRUE
SETVAR HPMSGFENCE,2
COMMENT
COMMENT  Here, we'll set HPAUTOCONT to true, meaning that
COMMENT  we won't have to preface every command line with
COMMENT  a :CONTINUE; it's implied. HPMSGFENCE = 2 tells
COMMENT  MPE/XL to override printing of error messages.
COMMENT
SETVAR FILE_SET,"!FILESET"
ECHO Please wait..Determining value of fileset
COMMENT
COMMENT Save off our original fileset and tell the user
COMMENT to hang on a sec.
COMMENT
SETVAR CIERROR,0
FILE TEMPFILE;TEMP;REC=-80,1,F,ASCII;NOCCTL
FILE OLDTEMP=TEMPFILE,OLDTEMP
```

```
LISTF !FILE_SET,1;*TEMPFILE
COMMENT
COMMENT Here is the LISTF of our fileset into a temporary
COMMENT file. The ,1 format will give us lots of good info
COMMENT about each file, as shown above.
COMMENT
IF CIERROR = 0 THEN
    RUN EDITOR.PUB.SYS;STDIN=PRGSTDIN;STDLIST=$NULL
ELSE
    ECHO Fileset !FILE_SET Is Invalid.....
    ECHO CIERROR is !CIERROR which means:
    ECHO !HPCIERRMSG
ENDIF
COMMENT
COMMENT  In these lires, if CIERROR is zero, run EDIT/3000
COMMENT  with a redirected $STDIN, else tell the user what
COMMENT  went wrong.
COMMENT
RESET TEMPFILE
RESET OLDTEMP
XEQ XEQCOMM
COMMENT
COMMENT  Reset the tempfile file equations and XEQute the
COMMENT  text file that EDIT/3000 created previously.
COMMENT
DELETEVAR FILE_@
DELETEVAR GOTFILE
DELETEVAR QUALNAME
DELETEVAR LISTF_@
DELETEVAR YES_@
COMMENT
COMMENT Cleanup. Delete vars created in this and other
COMMENT command files.
COMMENT
PURGE XEQCOMM
PURGE TEMPFILE,TEMP
COMMENT
COMMENT Purge the XEQ file and the temporary file.
COMMENT
SETVAR HPAUTOCONT,SAVE_HPAUTOCONT
SETVAR HPMSGFENCE,SAVE_HPMSGFENCE
COMMENT
COMMENT Reset these vars to their previous values.
COMMENT Delete the SAVE_ vars
DELETEVAR SAVE_@
```

The following example is the MPE file used as the $STDIN for EDITOR.PUB.SYS in the command file PURGESET. This file is PRGSTDIN:

```
TEXT *OLDTEMP
CHANGE 50 TO :": IN ALL
CHANGE 1 TO :PURGEFLE ": IN ALL
KEEP XEQCOMM,UNN
```

```
EXIT
```

The following command file is invoked from PURGESET. It has a single parameter, the line from the LISTF command that was prefixed with the name of this command file (PURGEFLE):

```
PARM LISTF_LINE_IN
COMMENT
COMMENT The single parameter 'passed' to this command file
COMMENT file from PURGESET is the line of the LISTF file,
COMMENT TEMPFILE.
COMMENT
COMMENT REMEMBER : HPMSGFENCE = 2 and HPAUTOCONT = TRUE
COMMENT
SETVAR LISTF_LINE,"!LISTF_LINE_IN"
COMMENT
COMMENT Save off the parameter passed for examination.
COMMENT
SETVAR FILE_NAME,"![STR(LISTF_LINE,1,8)]"
COMMENT
COMMENT Examine the LISTF file. There are three types of
COMMENT lines in that file:
COMMENT    1. header lines.
COMMENT    2. blank lines
COMMENT    3. filename lines.
COMMENT Lets save off the first eight character (the STR
COMMENT function, discussed previously, does that) into
COMMENT a variable called 'FILE_NAME'.
COMMENT
IF "!FILE_NAME" = "ACCOUNT=" THEN
COMMENT
COMMENT This LISTF line is a header line. It will contain
COMMENT the group and account name of those files that
COMMENT will be listed in the succeeding LISTF lines.
COMMENT
    SETVAR GOTFILE,FALSE
    SETVAR LISTF_ACCOUNT,"![STR(LISTF_LINE,11,8)]"
    SETVAR LISTF_GROUP,"![STR(LISTF_LINE,31,8)]"
COMMENT
COMMENT Lets set a variable to say let us know that this
COMMENT is not a file. Also, extract the account and
COMMENT group in variables.
ELSE
    IF ("!FILE_NAME" = "FILENAME") OR &
       ("!FILE_NAME" = "        ") THEN
        SETVAR GOTFILE,FALSE
    ELSE
COMMENT
COMMENT These lines are of no concern to us. We ignore them.
COMMENT
        SETVAR GOTFILE,TRUE
        SETVAR QUALNAME,"!FILE_NAME" + "." + "!LISTF_GROUP"
        SETVAR QUALNAME,"!QUALNAME" + "." + "!LISTF_ACCOUNT"
        STRIP QUALNAME
    ENDIF
ENDIF
COMMENT  ^
COMMENT  If it is not a header line or a blank line, it is
COMMENT  the name of a file. Lets fully qualify the filename
COMMENT  such that QUALNAME contains file.group.account.
COMMENT  If QUALNAME contains imbedded blanks (file,group or
COMMENT  account < 8 chars), STRIP will strip out blanks.
COMMENT
IF GOTFILE THEN
COMMENT ^
COMMENT If our Boolean var is set, the line just processed
COMMENT contains a filename, which was preceeded by blank and
COMMENT header lines (ALWAYS).
COMMENT
    SETVAR YES_NO_PROMPT,'Purge ' + '!QUALNAME' + ' (Y,N)?'
    SETVAR YES_NO," "
COMMENT ^
COMMENT Setup the variable names to contain the userprompt
COMMENT and the user's response.
COMMENT
    WHILE ("!YES_NO" <> "Y") AND ("!YES_NO" <> "N") DO
        INPUT YES_NO,PROMPT="!YES_NO_PROMPT"
    ENDWHILE
COMMENT '
COMMENT These lines will ask the user if the file should
COMMENT be purged. User must respond Y or N.
COMMENT
    IF ("!YES_NO" = "Y") THEN
        ECHO    << Purging !QUALNAME >>
        SETVAR CIERROR,0
        PURGE !QUALNAME
COMMENT '
COMMENT User responded Y. Attempt to purge the file.
COMMENT
        IF CIERROR <> 0 THEN
            ECHO Purge of !QUALNAME failed.......
        ENDIF
COMMENT ^
COMMENT Attempt failed. Tell the user why and continue to
COMMENT next file.
COMMENT
    ENDIF
ENDIF
COMMENT * end of command file.
```

The next command file is STRIP. It will parse a variable value and remove ALL leading,trailing and imbedded blanks.

```
PARM VARNAME
SETVAR LITVAR,"!VARNAME"
SETVAR SAVEVAR,!VARNAME
WHILE POS( " ","!SAVEVAR") > 0
    SETVAR SAVEVAR,"!SAVEVAR" - " "
```

```
ENDWHILE
SETVAR !LITVAR,"!SAVEVAR"
DELETEVAR LITVAR
DELETEVAR SAVEVAR
```

Once we have this command file under our belt, we can really start to get fancy. With a little imagination, we can select files in our fileset by specific attributes, such as file code, file size, etc. The mechanics are not difficult, but are not included because of space considerations. However, I can include a command file to set the attributal variables related to a file. This command file is FILEATTR:

```
PARM LISTF_LINE
OPTION NOLIST
SETVAR FILE_CODE,STR(LISTF_LINE,11,6)
COMMENT "  file code of file (string)
SETVAR FILE_CODE_I,FINFO('!QUALNAME',-9)
COMMENT '  file code of file (integer)
SETVAR CREATOR,FINFO('!QUALNAME',4)
COMMENT '  file creator (string)
SETVAR DATE_CREATED_STR,FINFO('!QUALNAME',6)
COMMENT '  date created (string) ]
SETVAR REC_SIZE,STR(LISTF_LINE,17,5)
COMMENT '  record size (bytes or words,string)
SETVAR ASCII_BINARY,STR(LISTF_LINE,26,1)
COMMENT '  file format (ascii or binary,string)
SETVAR FIXED_VARIABLE,STR(LISTF_LINE,25,1)
COMMENT '  record format (fixed or variable,string)
SETVAR BYTES_WORDS,STR(LISTF_LINE,22,1)
COMMENT '  record units of measure (bytes or words,string)
SETVAR EOF,STR(LISTF_LINE,30,9)
COMMENT '  number of records in the file (string)
SETVAR FILE_LIMIT,STR(LISTF_LINE,40,10)
COMMENT    maximum number of records in file (string)
SETVAR FILE_LIMIT_I,FINFO('!QUALNAME',12)
COMMENT '  max records in file (integer)
SETVAR FOPTIONS,FINFO('!QUALNAME',13)
COMMENT "  file options (string)
SETVAR FOPTIONS_I,FINFO('!QUALNAME',-13)
COMMENT '  integer foptions
SETVAR LAST_MOD_DATE_YYYYMMDD,FINFO('!QUALNAME',-8)
COMMENT '  last modification date (integer)
SETVAR INT_1,LAST_MOD_DATE_YYYYMMDD / 1000000
SETVAR INT_2,INT_1 * 1000000
SETVAR LAST_MOD_DATE_YYMMDD,LAST_MOD_DATE_YYYYMMDD - INT_2
COMMENT ' last mod date YYMMDD
STRIP FOPTIONS
STRIP FILE_CODE
STRIP REC_SIZE
STRIP EOF
STRIP FILE_LIMIT
COMMENT ' Strip the blanks from these vars. Once STRIPped,
COMMENT   their variable class will change to integer.
```

X. MPE/XL as a Programming Language

MPE has been thought of by some as being capable of being a programming language. Previous versions of MPE fell short of that mark, but probably not by much.

By that same token, lets examine MPE/XL in comparison to both older versions of MPE and other programming languages.

Although MPE/XL does not have a compiler, it is not necessarily a prerequisite of a programming language to have a compiler. Recursive abilities are found in MPE/XL (WHILE..ENDWHILE) and other programming languages; storage of numeric and non-numeric literals (SETVAR) are also present in MPE/XL and other compilers. Examination of stored literals (IF statement with explicit variable dereferencing) and intermediate literals (IF statement with STR function of literal constant, for example) are present in MPE/XL as well as compiler languages. The ability to call programs written in other languages (RUN statements) or in the same language is characteristic of some programming languages. Programs written in MPE/XL are callable from programs written in other programming languages (HPCICOMMAND intrinsic is needed). In addition, MPE/XL lends itself well to structuring and development with modularity, a trademark of some programming languages. MPE/XL can also accept input external to the program (INPUT command), produce output (PRINT, ECHO) and can perform arithmetic operations (CALC command). Older versions of MPE had but a few of these features.

Indeed, there may now be little argument that MPE/XL is a programming language. However, to make MPE/XL a REALLY powerful programming language, I offer Hewlett-Packard a 'wish list'.

XI. Some Things to Make MPE/XL Really Neat

While MPE/XL is a vast improvement over MPE, I would like to see several things implemented to make MPE/XL a bona fide programming language.

The first would be the ability to call any intrinsic from a command file (except those that use procedure labels). All the new HP intrinsics (HPFOPEN, HPFCLOSE) could be called in the same manner as calling other command files. For example, to call HPFOPEN, you might use command syntax like this:

```
HPFOPEN filenum,filename,fileoptions
```

This seems like a natural extension to MPE/XL, and something that should follow all the progress made with MPE/XL.

Several additional functions that would be nice to have might be a leading/trailing spaces trimming' function similar to the 'STRIP' command file presented previously. A 'STUFF' function to place a string within another string would be great. How about a square root function for the FORTRAN programmers?

These 'wishes' are not to be taken as a criticism of MPE/XL because they do not have these 'goodies'. MPE/XL is an outstanding example of a natural software evolution, and the absence of these functions or structure in no way detracts from the total product.

Acknowledgments

Many thanks to the people of Collier-Jackson, Inc in Tampa, for allowing me the experience of programming on a SPECTRUM machine. Special thanks to Barry Lemrow at CJI for ideas and suggestions, as well as his review of this paper before publication.

Special thanks to Hugh McKee of Hewlett-Packard, who also reviewed this paper before publication, and urged many months ago, that it should be written.

Computer Assisted VIEW, IMAGE & SPL

Norman A. Hills
N.A.Hills Computing Services Limited
336 Piccadilly Street
London ON Canada N6A 1S7

Introduction

VIEW provides an effective user to terminal interface, IMAGE is an efficient Data Base for organizing data storage, and SPL is a versatile language with which to build a busness system. DATA ELEMENTS must be defined in each of VIEW, IMAGE and SPL conforming to the various rules which are different for each sub-system, as illustrated in Table I.

Table I

DATA ELEMENT COMPARISONS

Sub-System	VIEW	IMAGE	SPL
Data Element	FIELD NAME	ITEM PART	IDENTIFIER
Alpha first Char and A-Z, 0-9 or:	_	+-*/?'#%&@	'
Numeric Types	DIG NUMn IMPn	K1 I1 I2	LOGICAL INTEGER DOUBLE REAL LONG
Character	CHAR	Xn	BYTE (char) LOGICAL (words)

While the programmer must live with these differences, he must also maintain a conformity between the sub-systems so that the Data Elements do not become damaged or distorted during any transfer between the sub-systems. This need for intra-system conformity complicates the task for system maintenance particularly when there is a need to revise any of the characteritics of a data element.

The presence of the data element in THREE distinct systems requires that each data element be defined THREE TIMES. For anyone concerned with labour efficiency or cost effectiveness there is an obvious redundancy of effort associated with THREE definitions for the same data element.

There are many packages available that address in various ways this issue of programmer productivity. Our approach has been to make more effective use of the existing resources that are available in the standard HP3000 utilities.

VIEW

Most systems start with the development of VIEW screens as samples for the user to review his visible interface to the system.

```
RUN ENTRY.PUB.SYS
```

requires the user to identify the name of the FORMS file and a BATCH file. The potential user can then experience the Field Edits and the other features of VIEW that can be included in the design of the screens.

As a later step, when the database has also been created, the user can employ DBENTRY from the CSL libraries to interact between the VIEW screen and the IMAGE database and it can be very useful to have this type of practice prior to the development of any application programs.

The information about each of our Data Elements that has been stored in our VIEW forms file is of course very pertinent to the balance of the program development. It would be very helpful to be able to have the computer convert some of this information rather than have the programmer create manually the corresponding data elements for the other sub-systems.

Existing VIEW Intrinsics in Table II can be used to retrieve pertinent information relative to each Field Name.

Table II

VIEW INTRINSICS

INTRINSIC	WORD OF BUFFER	INFORMATION RETURNED
VGETFILEINFO	5	Number of Forms in File
VGETFORMINFO	3-10	Form Name
	12	Number of FIELDS in Form
VGETFIELDINFO	11-18	Field Name
	20	Field Number
	21	Field Length
	25-26	Data Type of Field

To obtain complete information from the Forms File, the number of forms from VGETFILEINFO establishes the number of iterations required for VGETFORMINFO, and the number of fields from each VGETFORMINFO establishes the number of iterations required for VGETFIELDINFO.

For this retreived VIEW information to be applicable to IMAGE and SPL we have replaced the screen design identifier in the FIELD MENU during forms design with a FIELD NAME that follows our particular conventions established for IMAGE naming of SETS and ITEMS.

- Each Field Name becomes Itemname_Setname in the VIEW formsfile. In View we are limited to the underscore as the only permissible joiner, and this has the unfortunate feature of becoming invisible in the forms file listings.

- Although the JOIN utility of QUERY uses the convention of Setname.Itemname, we find it much more convenient during program preparation and program maintenance to be able to have the Item as the orimary key of any sort, so we are using the Itemname first.

- Each ITEM and SET name will be a maximum of 5 characters.

This information from the Formsfile can be re-arranged to develop a significant start for the IMAGE database DBSCHEMA as:

- A sorted list of ITEM names.

- A sorted list of SET names.

- A list of ITEM names that are associated with each SET name.

- A highly probable identification of the ITEM Type which we would usually translate as follows:

Table III

DATA TYPE TRANSLATION

VIEW TYPE	FIELD LENGTH	IMAGE TYPE	SPL TYPE
DIG	< 6	K1	LOGICAL
		or I1	INTEGER
	> 5	I2	DOUBLE
NUMn	< 6	I1	INTEGER
	> 5	I2	DOUBLE
IMPn	< 6	I1	INTEGER
	> 5	I2	DOUBLE
CHAR	n	Xn	LOGICAL

- An extraction of the PROCESSING SPECIFICATION associated with each Field is not available through an existing intrinsic. Our only solutions so far to this desire, is to copy the formsfile listing to a disc file, scan it for the occurrence of the Field Name, and then extract the subsequent text of PROCESSING SPECIFICATION.

IMAGE DBSCHEMA

The skeleton of Itemnames and Setnames followed by Items of the set can be enhanced by the programmer to include comments and any additional items or sets that have not originated as a View Form Field.

For those who would like to write the DBSCHEMA as their first step, is is unfortunate that there are no intrinsics that will help to develop VIEW from the DBSCHEMA.

RUN DBSCHEMA.PUB.SYS is used in the conventional way to create the ROOT FILE for the Database.

RUN DBDERIVE.PUB.SYS is used to create the BASENAMEnn dataset files for the database, and at the same time create the files of declarations and code that can become part of the subsequent SPL programs via appropriate $INCLUDE statements.

This process is quite fully explained in the June '87 issue of Interact article titled DERIVATIONAL PROGRAM CODE. To avoid repetition, we will concentrate here on the aspects that have been incorporated subsequent to this reference publication.

In the above reference we descripe the contents and purpose of four files created by DBDERIVE:

basenameDC

basenameGL

basenamePC

basenameZX

We now have DBDERIVE create a fifth file named basenameSP which is a BTREE file containing the correct spelling for all the known expression elements that may be referenced in the source code, as follows:

- BasenameGL contents of identifiers

- INTRLIST file of valid SPL Intrinsic names

- All of the SPL Reserved words.

This BTREE file basenameSP acts as a DICTIONARY of valid words that may be included in the SPL Program Code.

EDITOR entry of SOURCE CODE

The HP EDIT3000 can be customized to execute up to 3 user interfaces by using the initiation command:

```
RUN EDITOR.PUB.SYS;PARM=16
```

These user interfaces can be located in an SL at either the SYSTEM ACCOUNT or GROUP level, and can be invoked at either:

- INITIALIZATION to set up files or processes

- COMMAND phase so that whenever any test is entered in response to a / prompt, the users' procedure will be executed and may execute user defined special commands.

- ADD phase will be executed whenever any text is entered in response to a line number prompt.

We make use of this feature of EDIT3000 by having INIT Activate a Process which will receive via a message file during the ADD phase, a copy of each line as it is entered.

This background process takes each word of the entered line and searches the BTREE spelling Dictionary file basenameSP to determine if the word is valid. If a line contains any word that is not in the BTREE dictionary, then the offending word is surrounded by the escape sequence for blinking inverse video and the whole line is returned directly to the $STDLIST screen.

By having the spelling check performed by a background process, it does not slow down the interactive response of the terminal to each terminating line feed. Only the offending lines are returned to the $STDLIST screen and although the offending lines may not appear until two or three lines after they were entered, this is still far more efficient and productive than wating until the first compile to be made aware of transpositions and other spelling errors.

If the user is presented with a blinking word that is correct, then this is an effective reminder that the user should have the word apprpriately added to the program DECLARATIONS, at either the LOCAL or GLOBAL level, and have the word added to the BTREE file basenameSP so that the background process in future will return only lines that contain genuine omissions.

SPL COMPILES

Now that we have been using these techniques for about two years, we have settled down to a few conventions of convenience.

PROGDEV user is the program developer, with AL,PH,MR capability, home is the program testing group PROGTEST and the source code for development versions is in the group TSTLIBRY. The programs for testing are compiled as appropriate:

1/ Following any database revision, by a JOB STREAM which includes all the required steps such as:

   CODIDENT to create the currently required contents for all of the $INCLUDE files that are part of the source code listings Since all the modules will require compiling, this stream will also PURGE USLname and BUILD USLname.

   SPL textname.TSTLIBRY, USLname, $NULL to compile all of the source code files into the USL file.

   PREP and Save the compiled program.

```
2/ UDCname textname
   SPL !textname.TSTLIBRY, USLname, $NULL
   PREP USLname, $newpass; MAXDATA=nnnn; CAP=IA. BA. PH, MR
   PURGE Progname
   SAVE $OLDPASS, Progname
```

UDCname textname as a UDC with only one PARM which can be used to compile any source text from TSTLIBRY to USLname for PREP and SAVE of each modification during testing. It is important to recognize that any program change that intriduces global variables not previously used, will require the Job Stream to perform a complete recompile with the expanded $INCLUDE of Global variables.

LBRARIAN user is the ACCOUNT LIBRARIAN, with AL,PH,MR capability, home is the group LIBRARY and the source code for production versions is in the group LIBRARY.

In addition to this, we have a group OLDLIBRY which will contain a copy of the most recently replaced production source code, and a group named OLDPROG which will contain the most recently replaced PROG code. The steps to be performed when a new version has completed its test and is ready for production include:

```
3/ REN textname
   RENAME !textname.LIBRARY, !textname.OLDLIBRY
   FCOPY FROM = !textname.TSTLIBRY; TO = !textname.LIBRARY
```

REN is a UDC for quickly renaming source codes files that are about to be replaced.

4/ An expanded version of the job stream in TSTLIBRY is maintained as a SYSTEM job so that it can include the commands to DEALLOCATE before and ALLOCATE after the complete SPL and PREP of the production version of the program code.

As each new module is created, it is added to both of the job streams so that the updating of any production version can be accomplished with a minimum of instruction to the computer.

DATA TRANSFERS

Every application of VIEW and IMAGE requires program code to effect a data transfer between the Form Field and itemname'setname. Character strings are a reltively simple one to one transfer, but each of the numeric fields require a translation between the character format of the form field contents and the binary format of the database itemname. We have simplified this transfer through the use of VGET'TYPE AND VPUT'TYPE intrinsics of our own creation. These intrinsics operate on three INTEGER ARRAYS.

- The first array contains the field numbers for each window that is to be transferred, and the process is termination by a '0' as the field number.

- The second array contains the same number of integer elements, and each integer identifies the type of conversion that is to be performed during the transfer.

- The third integer array again contains the same number of elements, and the element is the word address of the location on the stack for the database item.

The creation and keying into the program of these arrays is tedious and easily subject to error. Since all of the variables were identified during the run of these utilities, we can readily have the three integer arrays and their assigned values computer created and ready for inclusion in the programs by a $INCLUDE statement.

SUMMARY

Our approach has been the application of creative laziness to a more effective utilization of existing resources that are available in the standard HP3000 utilites. While the product of our efforts may be of some interest to other users of IMAGE with SPL or VIEW, we feel that telling the story of how our shortcuts have evolved, could be an inspiration to others who should be looking for opportunities to implement savings within their own particular environment.

Using COBOL II's Facilities

By: Patrick A. Lockwood

Orion Systems Technology, Inc.

1309 East Northern Ave., Suite 701

Phoenix, AZ 85020

INTRODUCTION

This paper is targeted at analysts/programmers who are familiar with COBOL, but who have not had much experience utilizing it on the HP3000.

HP's implementation of 1974 ANSI Standard COBOL provides the designer and the programmer with many tools to help develop robust systems, with techniques that rival the current crop of fourth generation languages for speed, and allow the use of concepts not usually associated with this high level language.

This paper will explore techniques used to quickly develop systems in COBOL II (most of which is upwardly compatible with the ANSI '85 compiler), and to accomplish this without sacrificing quality.

Some topics to be covered are:

- Use of MULTIPLE COPY LIBRARIES for both DATA and PROCEDURE divisions.

- Commonly (and not so commonly) used INTRINSICS called from COBOL II, and how they can help you.

- DECLARATIVES and I/O STATUS checking.

- PROCESS HANDLING vs DYNAMIC SUBPROGRAMS in on line menus.

Examples from real programs will be used throughout, with minor changes to protect both the innocent and the guilty.

COPY LIBRARIES

I haven't worked on a computer system that uses COBOL without some form of the COPY statement; however many make it inconvenient to implement.

HP has provided two facilities for managing copy libraries that provide the programmer with great flexibility, as well as making standardization easy to implement.

1. **MULTI-MEMBER** library files.

   On the HP3000, unlike many other systems, multiple members may reside in one copy library file. Normally, a copy library file is maintained as a KSAM file, which may be easily manipulated with the COBEDIT.PUB.SYS utility. This utility allows you to add new members, and to delete or edit existing members. Editing is handled by process handling to the EDITOR while still within the COBEDIT utility.

2. **MULTI-LIBRARY** COPY STATEMENTS in a program.

   Within one program, you may copy members from multiple library files, thus allowing you to maintain separate libraries of standardized routines used in any program, as well as libraries of members unique to a single application.

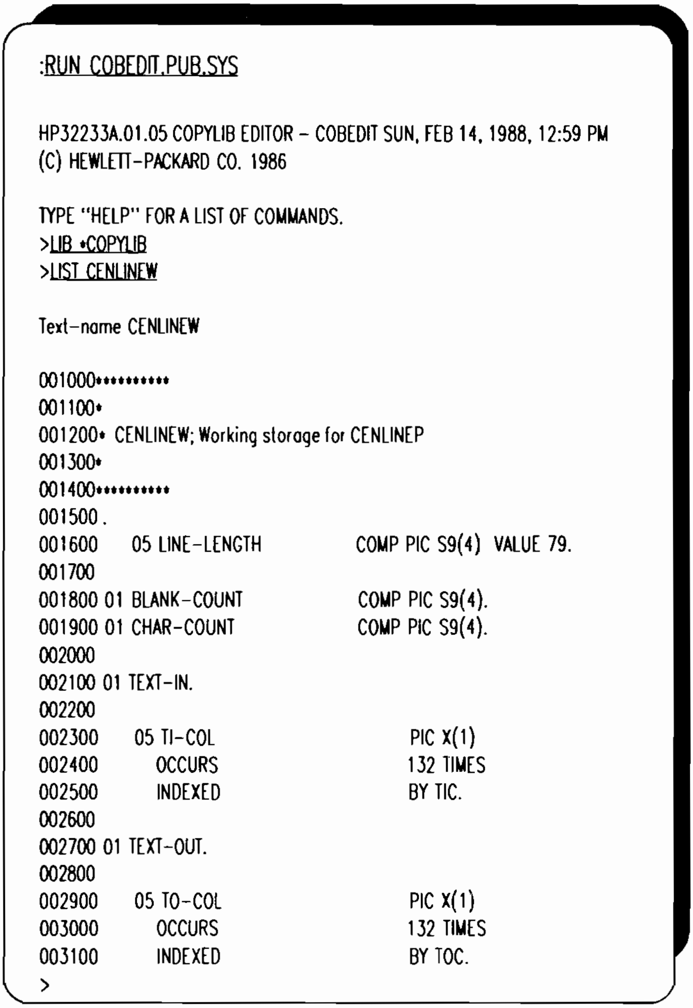
Figure 1

```
102.2   $PAGE ''Working Storage Copy Members''

102.3    01  CENLINEW     COPY CENLINEW IN COPYLIB NOLIST.
102.4    01  COMMONW      COPY COMMONW  IN COPYLIB NOLIST.
102.5    01  STDCALLW     COPY STDCALLW IN COPYLIB NOLIST.
102.6    01  VCALLW       COPY VCALLW   IN COPYLIB NOLIST.
102.7    01  PAUSEW       COPY PAUSEW   IN COPYLIB NOLIST.
102.8    01  STDPRTRW     COPY STDPRTRW IN COPYLIB NOLIST.
102.9
103      01  WSAPTCD      COPY WSAPTCD  IN GCCLIB  NOLIST.
103.1    01  WSBANK       COPY WSBANK   IN GCCLIB  NOLIST.
103.2    01  WSCTLREC     COPY WSCTLREC IN GCCLIB  NOLIST.
103.3    01  WSGLACCT     COPY WSGLACCT IN GCCLIB  NOLIST.
103.4    01  WSJOB        COPY WSJOB    IN GCCLIB  NOLIST.
103.5    01  WSLEDGER     COPY WSLEDGER IN GCCLIB  NOLIST.
103.6    01  WSOWNER      COPY WSOWNER  IN GCCLIB  NOLIST.
```

Figure 1 shows how these two facilities make it easy to combine data from multiple copy libraries into one program. Note that two libraries are specified; COPYLIB, which is the library of commonly used members, and GCCLIB, which contains record descriptions used only in the GCC applications. The NOLIST entry tells the COBOL compiler not to list the data being inserted into the program at compile time.

COPY LIBRARIES

The first copy library member listed in Figure 1 is CENLINEW, the commonly used working storage for centering text in any line up to 132 characters long. Using the COBEDIT utility, it's easy to list the data from a copy member to the terminal.

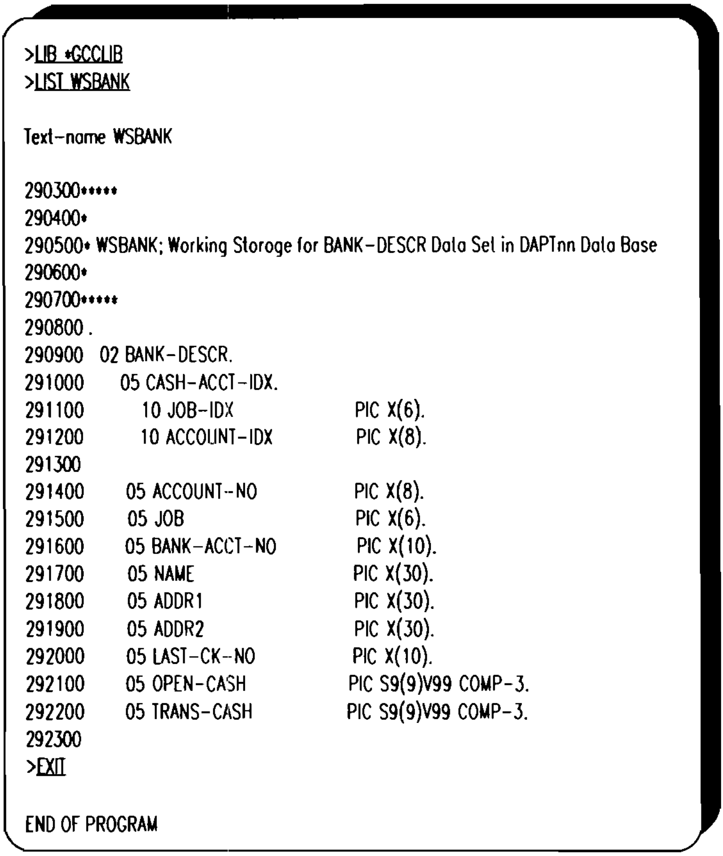


```
:RUN COBEDIT.PUB.SYS

HP32233A.01.05 COPYLIB EDITOR - COBEDIT SUN, FEB 14, 1988, 12:59 PM
(C) HEWLETT-PACKARD CO. 1986

TYPE "HELP" FOR A LIST OF COMMANDS.
>LIB *COPYLIB
>LIST CENLINEW

Text-name CENLINEW

001000***********
001100*
001200* CENLINEW; Working storage for CENLINEP
001300*
001400***********
001500 .
001600     05 LINE-LENGTH           COMP PIC S9(4)  VALUE 79.
001700
001800 01 BLANK-COUNT               COMP PIC S9(4).
001900 01 CHAR-COUNT                COMP PIC S9(4).
002000
002100 01 TEXT-IN.
002200
002300     05 TI-COL                        PIC X(1)
002400        OCCURS                        132 TIMES
002500        INDEXED                       BY TIC.
002600
002700 01 TEXT-OUT.
002800
002900     05 TO-COL                        PIC X(1)
003000        OCCURS                        132 TIMES
003100        INDEXED                       BY TOC.
>
```

COPY LIBRARIES

The COBEDIT utility allows you to switch between libraries; the following shows a listing of a member of GCCLIB, which contains application specific record descriptions.

NOTE that COBEDIT allows back-referenced file names for selecting the current library.

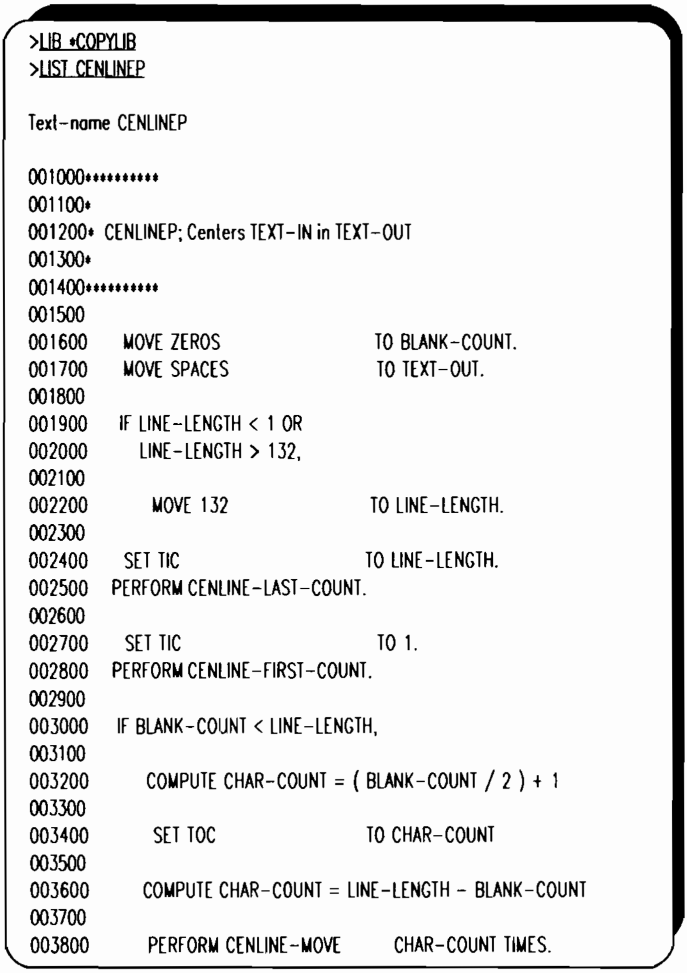


```
>LIB *GCCLIB
>LIST WSBANK

Text-name WSBANK

290300*****
290400*
290500* WSBANK; Working Storage for BANK-DESCR Data Set in DAPTnn Data Base
290600*
290700*****
290800.
290900  02 BANK-DESCR.
291000     05 CASH-ACCT-IDX.
291100        10 JOB-IDX               PIC X(6).
291200        10 ACCOUNT-IDX           PIC X(8).
291300
291400     05 ACCOUNT-NO               PIC X(8).
291500     05 JOB                      PIC X(6).
291600     05 BANK-ACCT-NO             PIC X(10).
291700     05 NAME                     PIC X(30).
291800     05 ADDR1                    PIC X(30).
291900     05 ADDR2                    PIC X(30).
292000     05 LAST-CK-NO               PIC X(10).
292100     05 OPEN-CASH                PIC S9(9)V99 COMP-3.
292200     05 TRANS-CASH               PIC S9(9)V99 COMP-3.
292300
>EXIT

END OF PROGRAM
```

Common routines may also be stored in copy libraries; once tested, they may be used easily by all members of the staff without worrying about re-inventing the wheel, and with assurance that they are not contributing towards bugs discovered during testing.

COPY LIBRARIES

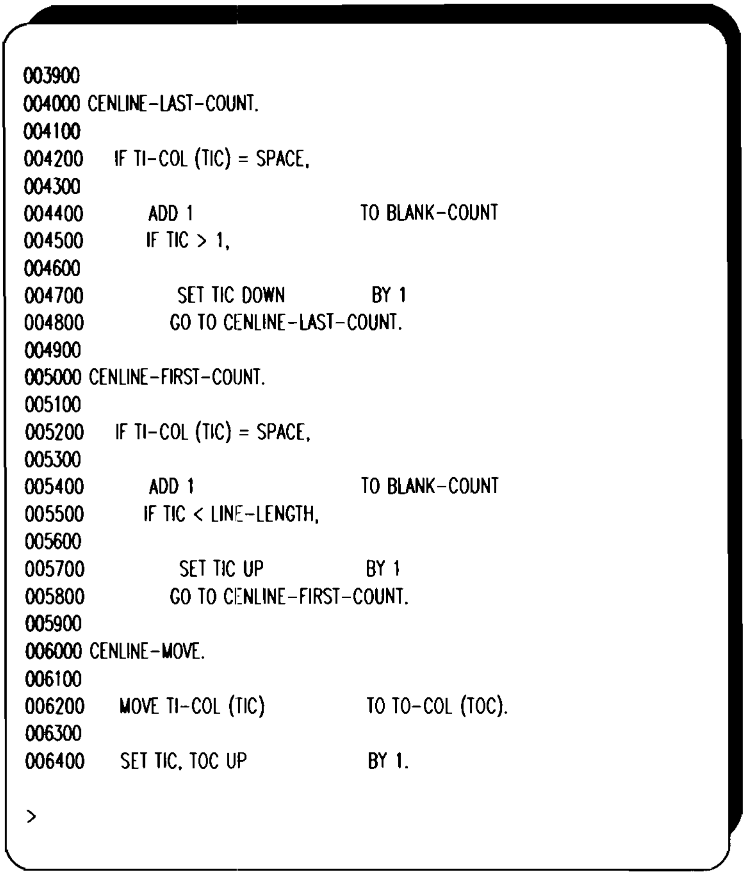
A common routine CENLINEP is stored in COPYLIB; many programs in different applications have occasional need to center text.



```
>LIB *COPYLIB
>LIST CENLINEP

Text-name CENLINEP

001000**********
001100*
001200* CENLINEP; Centers TEXT-IN in TEXT-OUT
001300*
001400**********
001500
001600      MOVE ZEROS                TO BLANK-COUNT.
001700      MOVE SPACES               TO TEXT-OUT.
001800
001900     IF LINE-LENGTH < 1 OR
002000        LINE-LENGTH > 132,
002100
002200          MOVE 132              TO LINE-LENGTH.
002300
002400      SET TIC                   TO LINE-LENGTH.
002500     PERFORM CENLINE-LAST-COUNT.
002600
002700      SET TIC                   TO 1.
002800     PERFORM CENLINE-FIRST-COUNT.
002900
003000     IF BLANK-COUNT < LINE-LENGTH,
003100
003200         COMPUTE CHAR-COUNT = ( BLANK-COUNT / 2 ) + 1
003300
003400          SET TOC               TO CHAR-COUNT
003500
003600         COMPUTE CHAR-COUNT = LINE-LENGTH - BLANK-COUNT
003700
003800          PERFORM CENLINE-MOVE      CHAR-COUNT TIMES.
```

Continued on Page 7 . . . .

COPY LIBRARIES



```
003900
004000 CENLINE-LAST-COUNT.
004100
004200    IF TI-COL (TIC) = SPACE,
004300
004400        ADD 1                     TO BLANK-COUNT
004500        IF TIC > 1,
004600
004700            SET TIC DOWN          BY 1
004800            GO TO CENLINE-LAST-COUNT.
004900
005000 CENLINE-FIRST-COUNT.
005100
005200    IF TI-COL (TIC) = SPACE,
005300
005400        ADD 1                     TO BLANK-COUNT
005500        IF TIC < LINE-LENGTH,
005600
005700            SET TIC UP            BY 1
005800            GO TO CENLINE-FIRST-COUNT.
005900
006000 CENLINE-MOVE.
006100
006200    MOVE TI-COL (TIC)             TO TO-COL (TOC).
006300
006400    SET TIC, TOC UP               BY 1.

>
```

The ability to have multiple libraries accessed within one COBOL program makes the use of common routines a 'common' occurrence in shops that rely on standardized techniques to develop programs quickly.

COPY LIBRARIES

Combining the copy members CENLINEW and CENLINEP from COPYLIB, and WSBANK from GCCLIB, we're able to use coding techniques like the following:

```
MOVE NAME IN WSBANK         TO TEXT-IN.
MOVE 30                     TO LINE-LENGTH.

PERFORM CENLINEP.

MOVE TEXT-OUT               TO HEADING-BANK-NAME.
```

NOTE that the use of the copy member name as a paragraph name (CENLINEP) is acceptable; the entry in COPYLIB actually has no paragraph name.

Similarly, by beginning the working storage copy members with a period (.), and having the '01' level be prior to the COPY statement, allows reference to the group item by its copy member name (WSBANK), as well as by the '02' level that corresponds to the data set name (BANK-DESCR).

The NOLIST convention for copy members is common in shops that make heavy use of copy libraries; typically, each programmer has a listing of the common library (COPYLIB) at his/her desk, as well as listings of those application dependent libraries (such as GCCLIB) that are frequently referenced. This makes compiled listings shorter, and for programmers experienced with the shop's conventions, easier to work with.

DECLARATIVES and I/O STATUS

You've seen it, the infamous TOMBSTONE printed by the file system when a COBOL program attempts an I/O operation that is unsuccessful, and for which there wasn't an appropriate error handling routine established.

Many programs check for AT END and INVALID KEY conditions, but are at a total loss if an OPEN fails, or if the INVALID KEY condition doesn't allow the program to adequately diagnose the problem, thereby preventing 'elegant' error handling.

Two features of COBOL II (and COBOL 85) provide the means to trap I/O errors and take the appropriate action based upon the actual condition that occurred.

DECLARATIVES.

This Section of the program, which must be the first Section within the Procedure Division, defines procedures to be used when the file system discovers an error or unusual condition.

FILE STATUS.

This entry in the SELECT filename clause defines a storage location in which the status of the most recent I/O operation for a file is returned.

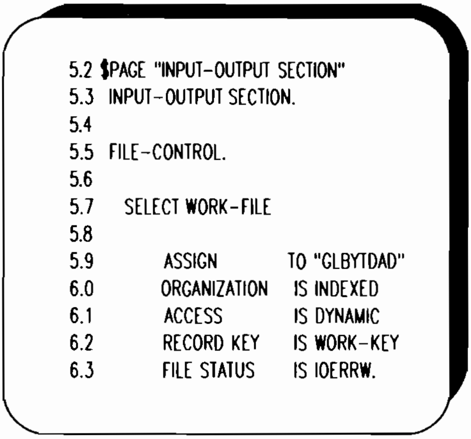
The two, working in combination, give the programmer complete control over error and exceptional condition processing for a file.

To see how these work together, we'll begin with some sample program code, beginning on Page 10 with a file select clause using the FILE STATUS option.

DECLARATIVES and I/O STATUS

The FILE STATUS item must be selected if you want to trap I/O errors and be able to determine the cause of the I/O failure. When an input or output operation has been performed on the file, the status item is updated with a two character code indicating the status of the operation.

If the first byte contains 0 (ZERO), the operation was basically successful. The second byte contains additional information further defining the status.



```
$PAGE "INPUT-OUTPUT SECTION"
 INPUT-OUTPUT SECTION.

 FILE-CONTROL.

   SELECT WORK-FILE

        ASSIGN          TO "GLBYTDAD"
        ORGANIZATION     IS INDEXED
        ACCESS           IS DYNAMIC
        RECORD KEY       IS WORK-KEY
        FILE STATUS      IS IOERRW.
```

The example above shows a FILE STATUS item of IOERRW. This is a two byte field defined as a commonly used member of COPYLIB.

Page 11 shows this copy member, which also includes an additional field used for interpreting the second byte of the status returned for I/O operations.

DECLARATIVES and I/O STATUS

Meanings of the first byte (IO-ERR-1) are:

- * 0 Successful completion
- * 1 At end, EOF has been reached
- * 2 Invalid key, duplicate for writes, or not found for reads
- * 3 Physical I/O error, or beyond EOF

```
**********
*
* IOERRW: Working Storage for FILE STATUS
*
**********
 .
    02 IOERRW-CHARS     PIC X(2) VALUE ``00''.

      88 IO-SUCCESSFUL           VALUE ``00''.
      88 IO-ALLOWED-DUPL         VALUE ``02''.
      88 IO-EOF                  VALUE ``10''.
      88 IO-SEQUENCE-ERR         VALUE ``21''.
      88 IO-DUPL-KEY             VALUE ``22''.
      88 IO-NOT-FOUND            VALUE ``23''.
      88 IO-BOUND-VIOL           VALUE ``24'',
                                       ``34''.
      88 IO-PERM-ERROR           VALUE ``30''.

    02  FILLER
        REDEFINES   IOERRW-CHARS.

      05 IO-ERR-1        PIC X(1).

         88 IO-OK              VALUE ``0''.
         88 IO-AT-END          VALUE ``1''.
         88 IO-INVALID-KEY     VALUE ``2''.
         88 IO-PERMANENT-ERROR VALUE ``3''.
         88 IO-MISC-ERROR      VALUE ``9''.

      05 IO-ERR-2       PIC X(1).
```

```
**********
*
* Used to convert the second byte of FILE
* STATUS (IOERRW) to a valid MPE file system
* error code (FSERR)
*
**********



 01 IOERR-CONVERT.

     05 IOERR-DUMMY     PIC X(1) VALUE
                                 LOW-VALUES.

     05 IOERR-CHARACTER   PIC X(1).

 01 IOERR-MPE-ERR-NUM
    REDEFINES  IOERR-CONVERT
                    COMP  PIC S9(4).

        88 IO-UNOBTAINABLE     VALUE 90,
                                     91.
        88 IO-FILE-NOT-FOUND   VALUE 52,
                                     53.
        88 IO-DEV-UNAVAILABLE VALUE 55.
        88 IO-DUPLICATE-FILE   VALUE 100,
                                     101.
```

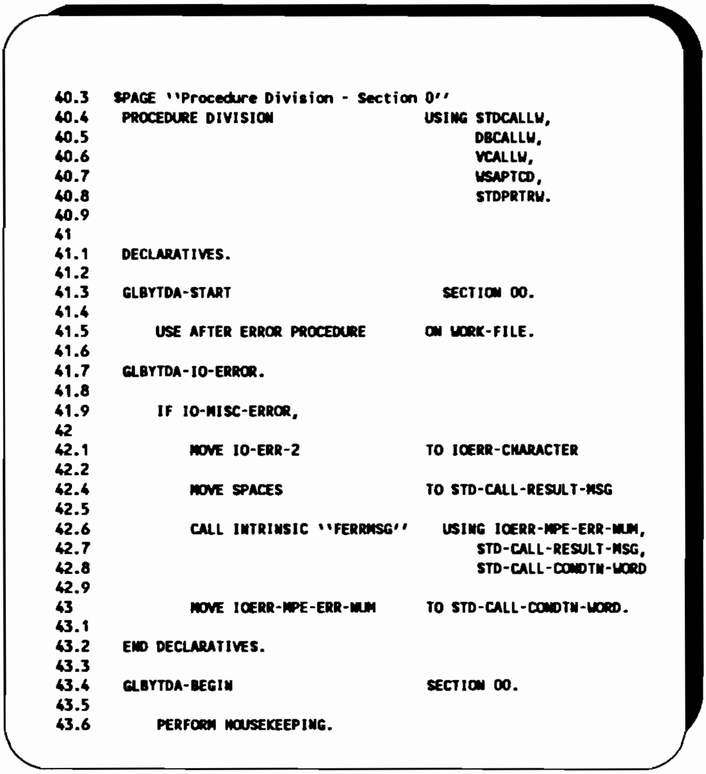
WARNING: these definitions [illegible] COBOL [illegible]

DECLARATIVES and I/O STATUS

The other feature that helps us handle I/O errors is a fairly simple routine inserted in the first part of the program; it works for main programs and subprograms.

The program must have a DECLARATIVES Section, which must be the first section in the program. NOTE that the use of a section for Declaratives requires a section name for the first paragraph of the normal procedure division, even if the program is not to be sectioned to create additional code segments.

A sample DECLARATIVES follows.



```
                          $PAGE ''Procedure Division - Section 0''
A dynamic subprogram       PROCEDURE DIVISION                   USING STDCALLW,
being called with five                                                DBCALLW,
     parameters . . .                                                 VCALLW,
     all copy members                                                 WSAPTCD,
                                                                      STDPRTRW.


Required statement to      DECLARATIVES.
   begin DECLARATIVES
SECTION NAME required      GLBYTDA-START                          SECTION 00.

        USE statement          USE AFTER ERROR PROCEDURE        ON WORK-FILE.

Followed by paragraph      GLBYTDA-IO-ERROR.
   name for procedure
                               IF IO-MISC-ERROR,

   Convert 2nd Byte of             MOVE IO-ERR-2                TO IOERR-CHARACTER
IOERRW to FSERR num.
                                   MOVE SPACES                  TO STD-CALL-RESULT-MSG

    Get FSERR message              CALL INTRINSIC ''FERRMSG''     USING IOERR-MPE-ERR-NUM,
                                                                      STD-CALL-RESULT-MSG,
                                                                      STD-CALL-CONDTN-WORD

   For return to caller            MOVE IOERR-MPE-ERR-NUM       TO STD-CALL-CONDTN-WORD.

   Required statement      END DECLARATIVES.

Begin normal program       GLBYTDA-BEGIN                        SECTION 00.
    with section name
                               PERFORM HOUSEKEEPING.
```

DECLARATIVES and I/O STATUS

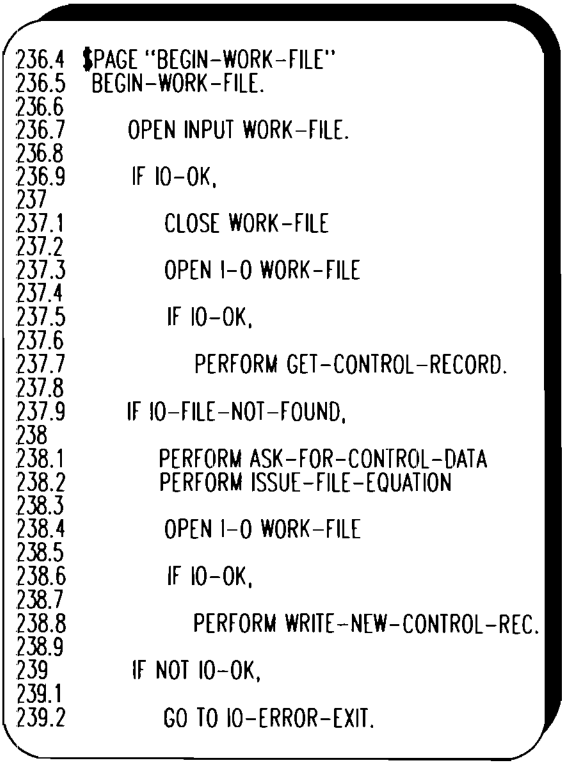
The documentation for COBOL provides the rules for precedence for FILE STATUS items and USE PROCEDURES (with a flow chart further explaining this in KPR# 4700245142 in the System Staus Bulletin); it really boils down to a simple statement . . . .*'If you use a FILE STATUS item, and have a USE PROCEDURE, your destiny is in your own hands'.*

The FILE STATUS item is updated for all of your I/O, and the USE PROCEDURE is executed for every exceptional condition. This allows the following type programming:

The file is first opened for input; just to see if it's there. If so, the control record is retrieved. If it doesn't exist, the user is asked for parameters for building a new file, and for data to be stored in the control record.

The change in processing based upon IO-FILE-NOT-FOUND (FSERR 52) is easy to handle; all other I/O errors are unexpected and cause an exit to the error handling routine.

The USE Procedure is invoked for all 'NOT IO-OK' situations, and the FILE STATUS item is set after each I/O.



```
$PAGE "BEGIN-WORK-FILE"
 BEGIN-WORK-FILE.

     OPEN INPUT WORK-FILE.

      IF IO-OK,

          CLOSE WORK-FILE

          OPEN I-O WORK-FILE

          IF IO-OK,

              PERFORM GET-CONTROL-RECORD.

      IF IO-FILE-NOT-FOUND,

          PERFORM ASK-FOR-CONTROL-DATA
          PERFORM ISSUE-FILE-EQUATION

          OPEN I-O WORK-FILE

          IF IO-OK,

              PERFORM WRITE-NEW-CONTROL-REC.

      IF NOT IO-OK,

          GO TO IO-ERROR-EXIT.
```

If the FILE STATUS contains a 9 in the first byte (IO-ERR-1), the second byte is moved to IOERR-CHARACTER, which is used, via the redefinition of IOERR-CONVERT, to call INTRINSIC "FERRMSG" to obtain an interpretation of the error condition to place in a message passed back to the caller.

DECLARATIVES and I/O STATUS

In the example shown on Page 13 only the IO-FILE-NOT-FOUND condition was specifically anticipated. But note how easy it would be to attempt the open, then if the condition IO-UNOBTAINABLE (Exclusive Violation) was found, the program could 'elegantly' let the user know that the file was in use by someone else, and request that a later retry would be appropriate.

The COBOL manual recommends calling "CKERROR" to convert the second byte of FILE STATUS to an ASCII number, however the simple move to a redefined integer (COMP) accomplishes the same thing, and that number is in the correct format for calls to "FERRMSG".

Of course, there are many other ways to accomplish the same logic that this little routine uses; it only points out one set of circumstances that make use of these techniques. Once mastered, and with key elements readily available in a COPYLIB, you'll find its flexibility to be helpful in complex applications.

Obtaining actual file system error codes for those conditions that do not begin with a 9 in the first byte is also possible. The intrinsic FCHECK applies to files on any device, and can be used simply. For example:

```
CALL INTRINSIC "FCHECK"  USING WORK-FILE,
                               IOERR-MPE-ERR-NUM,
                                \\, \\, \\.
```

This returns the file system error code for the last I/O for WORK-FILE into IOERR-MPE-ERR-NUM.

NOTE that the intrinsics manual asks for *filenum* for file intrinsics; COBOL programmers may substitute *filename*, as defined in a SELECT statement. The backslashes in the call above stand for 'null' parameters; optional parameters not required for a simple call just for the file system error number.

To easily change programs from COBOL 74 to COBOL 85, a new copy member can be created that contains the FILE STATUS error codes used with the newer compiler (and run time processing). The table on Page 15 provides a brief overview of the differences.

DECLARATIVES and I/O STATUS

COMPARISON of ANSI 85 vs ANSI 74 I-O STATUS CODES

ANSI85	ANSI74	MEANING
04	00	Read length of record doesn't match file.
05	00	Optional file not present; created.
07	00	File NOT a TAPE file as OPEN/CLOSE implies.
14	00	Relative record number larger than PICTURE of key descriptor.
24	24	Write beyond file boundary, or relative record number larger than PICTURE of key descriptor.
35	9x	Non-optional file not present; not created.
37	00	Open mode invalid for file type.
38	00	Attempted OPEN on file closed with lock.
39	00	Attribute conflict; file not opened.
41	9x	Attempted OPEN on file that is open.
42	9x	Attempted CLOSE on file that is not open.
43	9x/00	Attempted DELETE/REWRITE without prior READ.
44	00	Boundary violation or invalid record size.
46	10	Attempted READ after EOF or previously unsuccessful read.
47	9x/00	Attempted READ on file not open for input.
48	9x/00	Attempted WRITE on file not open for output (or I-O).
49	9x/00	Attempted REWRITE/DELETE on file not open for I-O.

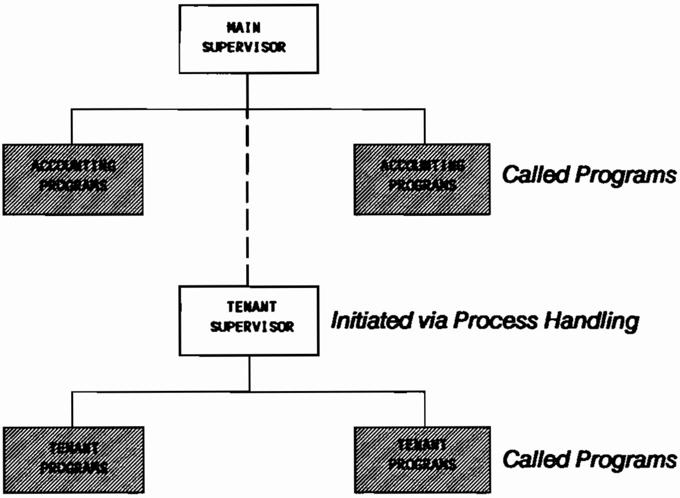
Creating a new copy library member incorporating the revisions to I-O Status makes upgrading to COBOL 85 an easier task.

SOME ADDITIONAL INTRINSICS

To introduce the use of some intrinsics that are fairly easy to use from COBOL II, I'll describe a situation that we ran into, and the solution that we used. Of course, there are always multiple solutions to any design/programming problem, but this will be fairly illustrative of the power available to COBOL programmers.

A system, written in COBOL 68, was being converted to COBOL 74. To those of you never exposed to the wonders of COBOL 68, it had no facility for calling HP intrinsics directly; any needed intrinsic calls were written in SPL. First the COBOL program called the SPL intrinsic handling routine, which called the instrinsic, then returned to the COBOL program. It was a little cumbersome, but it worked. However, it did discourage COBOL shops from heavy use of intrinsics. Naturally, as part of the conversion process, direct calls to intrinsics were substituted for the calls to SPL intrinsic handling routines.

The system being modified was structured with a MAIN SUPERVISOR, which called dynamic subprograms for various required functions. Due to the system table limits at the time the system was developed (CST entries, maximum code segments per process), it had to also initiate a second level supervisor for some functions, using process handling to accomplish this.



NOTE that parameters were passed between processes by entries in a data base.

SOME ADDITIONAL INTRINSICS

Our charter wasn't to totally redesign the system, but we were asked if we could speed the movement between the main supervisor and the secondary supervisor. The basic logic in use was:

MAIN SUPERVISOR

Post parameters to control data base
Close terminal
Call SPL routine to CREATE and ACTIVATE process

{ *Wait for Tenant Supervisor* }

Retrieve parameters from control data base
Open terminal
If parameters indicate a different production data base was opened by tenant supervisor, close initial data base and open data base that had been opened by tenant supervisor.

TENANT SUPERVISOR

Open control data base, and retrieve parameters
Open production data base
Open forms file
Open terminal

{ *Additional Processing* }

Post current data base name and other parameters to control data base, then close it.
Close production data base
Close terminal
Close forms file
STOP RUN

SOME ADDITIONAL INTRINSICS

After some analysis of the frequency of use of the Tenant Supervisor (how often it was initiated from within the Main Supervisor by a user in one session), we elected the following two concepts for reducing the transition time:

1. Eliminate the control data base as a means of passing parameters. Several options came to mind; we chose an extra data segment as a means of passing parameters between processes. They provide a fast means of sharing data between processes, and implementation in the system would be easy, utilizing the existing data structure (the control record layout) for parameter storage.

   This eliminated the data base open, GET/PUT I/O, and data base close in both the Main Supervisor as well as the Tenant Supervisor.

2. Because the frequency of use seemed to justify it, we elected to not terminate the Tenant Supervisor (STOP RUN) when its current processing was complete, but to retain it as a process that could be re-activated when next needed.

   This eliminated the repeated overhead of program loading, forms file open, and data base open (so long as the data base requested in the passed parameters was the same as the previously requested data base).

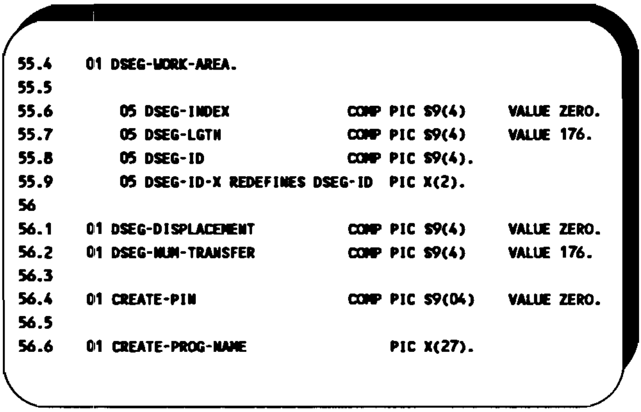
The results were quite acceptable to the users.; the transition time was reduced. On the first initiation of the Tenant Supervisor during any one session, there was still a noticeable delay, but not as long as previously.

The second, and subsequent initiations of the Tenant Supervisor were at a speed that gave no indication that another program was being started. The following intrinsics were used to accomplish this from within COBOL programs:

GETDSEG
DMOVOUT
CREATE
ACTIVATE
DEMOVIN
KILL
FREEDSEG

SOME ADDITIONAL INTRINSICS

To illustrate how these may be used from within COBOL programs, we'll start with the WORKING STORAGE used in the Main Supervisor.



```
                             55.4   01 DSEG-WORK-AREA.
                             55.5
  Assigned by MPE (GETDSEG)  55.6      05 DSEG-INDEX                COMP PIC S9(4)    VALUE ZERO.
     DSEG length (in words)  55.7      05 DSEG-LGTH                 COMP PIC S9(4)    VALUE 176.
      Program assigned name  55.8      05 DSEG-ID                   COMP PIC S9(4).
                             55.9      05 DSEG-ID-X REDEFINES DSEG-ID  PIC X(2).
                             56
          Starting Location  56.1   01 DSEG-DISPLACEMENT            COMP PIC S9(4)    VALUE ZERO.
    Words to DMOVIN/DMOVOUT  56.2   01 DSEG-NUM-TRANSFER            COMP PIC S9(4)    VALUE 176.
                             56.3
    PIN for created process  56.4   01 CREATE-PIN                   COMP PIC S9(04)   VALUE ZERO.
                             56.5
    Program to be initiated  56.6   01 CREATE-PROG-NAME                  PIC X(27).
```

Extra Data Segments are an additional segment of memory that a program is allowed to use for storage of data. They may be easily shared by multiple processes within the same process tree (father process and its sons). One advantage they have is that transfer of data is at memory to memory speeds; there is no disc I/O associated with their use, other than any required by MPE's memory manager.

The GETDSEG intrinsic is used to create a new Extra Data Segment, or to gain access to one that has been previously created. The ID is the name by which your program attempts to initially perform the GETDSEG, for the stated LENGTH.

The INDEX is a unique number assigned by MPE; once acquired, the index is used to obtain access to the data in the DSEG using the DMOVIN intrinsic (move data from extra data segment to your program's working storage) and the DMOVOUT intrinsic (move data from working storage to the extra data segment).

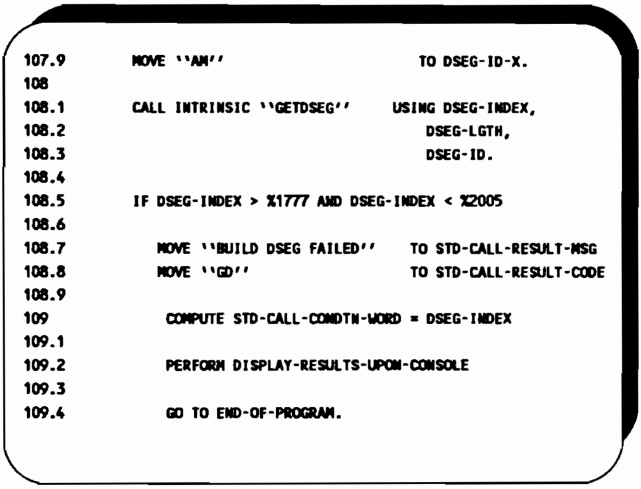
DISPLACEMENT is like a subscript or index, telling the DMOVIN and DMOVOUT intrinsics where in the Extra Data Segment to begin the move of data, using 0 (ZERO) as the first word. The size of the data string to be moved is stated in number of words.

SOME ADDITIONAL INTRINSICS

The first change to the Main Supervisor was to add a call to GETDSEG, done only once in the intialization logic. This establishes the data segment, and assigns a unique index number to it. This is the identifier by which it will be recognized by any program that is a member of this process.

The ID is the name by which other processes sharing this extra data segment will first obtain access to it.

The size, in number of words, is the size of the record used in the original data base parameter passing routine.



```
107.9        MOVE ''AM''                          TO DSEG-ID-X.
108
108.1        CALL INTRINSIC ''GETDSEG''      USING DSEG-INDEX,
108.2                                              DSEG-LGTH,
108.3                                              DSEG-ID.
108.4
108.5        IF DSEG-INDEX > %1777 AND DSEG-INDEX < %2005
108.6
108.7           MOVE ''BUILD DSEG FAILED''     TO STD-CALL-RESULT-MSG
108.8           MOVE ''GD''                    TO STD-CALL-RESULT-CODE
108.9
109              COMPUTE STD-CALL-CONDTN-WORD = DSEG-INDEX
109.1
109.2            PERFORM DISPLAY-RESULTS-UPON-CONSOLE
109.3
109.4            GO TO END-OF-PROGRAM.
```

Assign an ID to DSEG

Returned by MPE
Length desired
ID established above

Check for error conditions.
If found, establish
diagnostics and exit to
common error display
routine used by Main
Supervisor for all
"catastrophic" errors found
in Main or subprograms.

Once established, the Extra Data Segment may be used repeatedly. There is no significant time used in acquiring an Extra Data Segment; it is significantly less than the time used to open a data base.

The error conditions for which the test is done are items such as invalid length, you've attempted to exceed the maximum configured XDSEGS, etc.

SOME ADDITIONAL INTRINSICS

The end of program routine was modified to include a call to FREEDSEG; this releases the the data segment from the session. Perhaps not strictly required in this application; experience has shown that good housekeeping pays off.

The same is true for the call to the KILL intrinsic. This deletes the son process; that is the Tenant Supervisor, if it had been intitiated.

```
                                         END-OF-PROGRAM.

If Tenant Supervisor alive, kill it          IF CREATE-PIN NOT = ZERO,

                                                CALL INTRINSIC ''KILL''      USING CREATE-PIN.

If XDSEG was created, free it                IF DSEG-INDEX NOT = ZERO,

                                               CALL INTRINSIC ''FREEDSEG''  USING DSEG-INDEX,
                                                                                   DSEG-ID.

If production data base open,                IF DB-OPEN,
                     close it
                                                PERFORM DBCLSDBP.

Close the forms file                         PERFORM VCLOSEFORMF.

Close the terminal                           PERFORM VCLOSETERM.

Exit this program                            GOBACK.
```

Earlier, the use of copy members for commonly used functions was discussed. This routine includes performs of three commonly used copy members:

DBCLSDBP; closes the currently open data base
VCLOSEFORMF; closes the currently open VPLUS forms file
VCLOSETERM; closes the currently open terminal file used by VPLUS

SOME ADDITIONAL INTRINSICS

The routines to create and/or activate the Tenant Supervisor and pass parameters become easy after the preliminary work has been done.

Move parameters to XDSEG

Create/Activate another process; then wait for it to return

Move changed parameters back in

Use the INDEX assigned by GETDSEG

Starting location in target DSEG

Number of words to transfer

Source of data to be moved out

Check for errors, and exit if any found

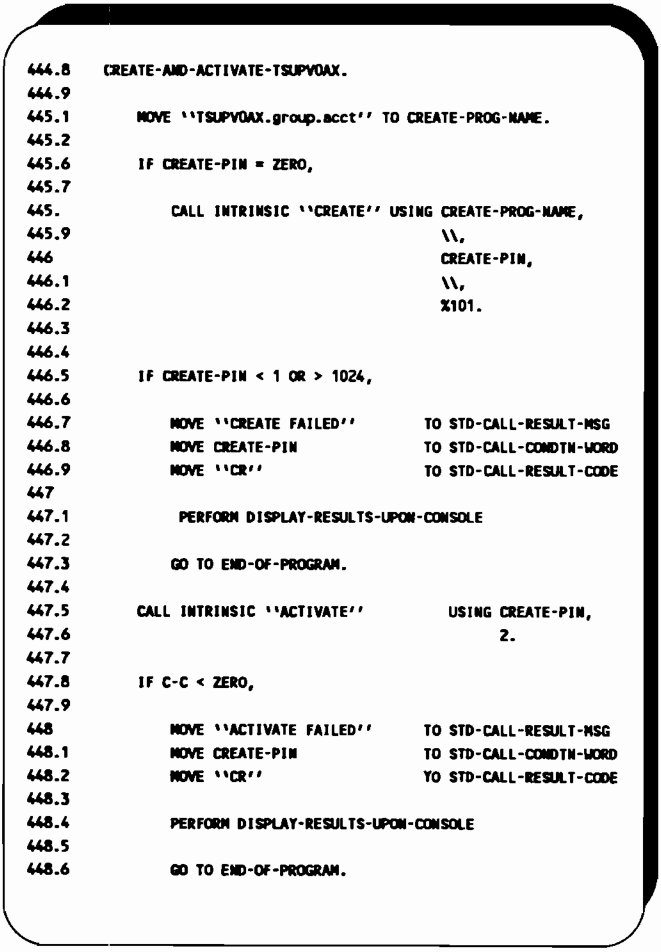
```
439.3   INITIATE-TENANT-SUPERVISOR.
440
440.1       PERFORM MOVEOUT-DSEG.
440.2
440.3       PERFORM CREATE-AND-ACTIVATE-TSUPVOAX.
440.4
440.5       PERFORM MOVEIN-DSEG.
440.6
440.9   * Continue processing
441.6
441.7   MOVEOUT-DSEG.
441.8
442.7*      Set up parameters here
443.3
443.4       CALL INTRINSIC ''DMOVOUT''   USING DSEG-INDEX,
443.5                                      DSEG-DISPLACEMENT,
443.6                                      DSEG-NUM-TRANSFER,
443.7                                      WSCTLREC.
443.8
443.9       IF C-C NOT = ZERO,
444
444.1         MOVE ''DM''            TO STD-CALL-RESULT-CODE
444.2         MOVE ''DMOVOUT FAIL"   TO STD-CALL-RESULT-MSG
444.3
444.4           PERFORM DISPLAY-RESULTS-UPON-CONSOLE
444.5
444.6           GO TO END-OF-PROGRAM.
```

NOTE that this process is suspended after the Tenant Supervisor is initiated (Line 440.3), so the next instruction will be executed as soon as control is returned to this process. The use of the Extra Data Segment is barely more difficult than a "CALL USING" when dealing with a subprogram.

Continued on Page 23 . . .

SOME ADDITIONAL INTRINSICS

Creating and/or activating a process is not difficult. NOTE that the %101 parameter (flags, as defined in the Intrinsics Manual) tells MPE that the created process should use the NOCB parameter; it has stack size problems and needs the space this frees.



Set up program name

Only if not previously created, will the process be created

NO entry point name

PIN number returned

No PARM= passed

NOCB & reactivate father when new process terminates

If errors found, exit to common diagnostic routine

Activate the Tenant Supervisor, expecting to be activated by it

If errors found, exit

```
CREATE-AND-ACTIVATE-TSUPVOAX.

    MOVE ''TSUPVOAX.group.acct'' TO CREATE-PROG-NAME.

    IF CREATE-PIN = ZERO,

        CALL INTRINSIC ''CREATE'' USING CREATE-PROG-NAME,
                                        \\,
                                        CREATE-PIN,
                                        \\,
                                        %101.


    IF CREATE-PIN < 1 OR > 1024,

        MOVE ''CREATE FAILED''       TO STD-CALL-RESULT-MSG
        MOVE CREATE-PIN              TO STD-CALL-CONDTN-WORD
        MOVE ''CR''                  TO STD-CALL-RESULT-CODE

         PERFORM DISPLAY-RESULTS-UPON-CONSOLE

        GO TO END-OF-PROGRAM.

    CALL INTRINSIC ''ACTIVATE''         USING CREATE-PIN,
                                               2.

    IF C-C < ZERO,

        MOVE ''ACTIVATE FAILED''     TO STD-CALL-RESULT-MSG
        MOVE CREATE-PIN              TO STD-CALL-CONDTN-WORD
        MOVE ''CR''                  YO STD-CALL-RESULT-CODE

        PERFORM DISPLAY-RESULTS-UPON-CONSOLE

        GO TO END-OF-PROGRAM.
```

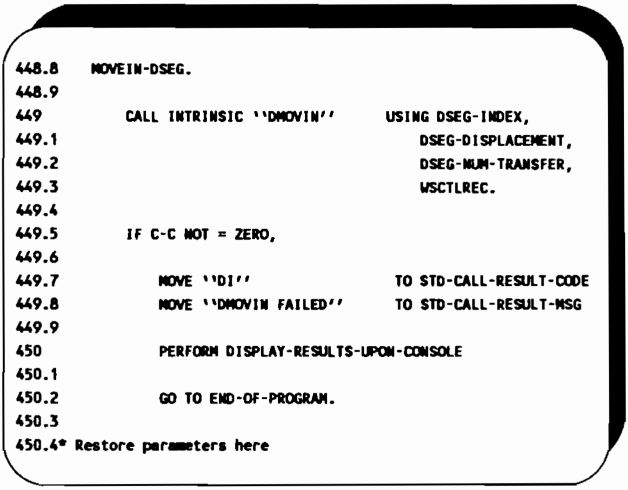
Continued on Page 24 . . .

SOME ADDITIONAL INTRINSICS

When theTenant Supervisor returns control to the Main Supervisor, the passed, and maybe changed, parameters are restored using DMOVIN. Its operation is just the reverse of the DMOVOUT intrinsic; it moves data from the Extra data Segment into the program's Working Storage.

Use INDEX from GETDSEG
Starting location in DSEG
Number of words to move
Target in working storage

If errors found, exit



```
448.8    MOVEIN-DSEG.
448.9
449          CALL INTRINSIC ''DMOVIN''      USING DSEG-INDEX,
449.1                                             DSEG-DISPLACEMENT,
449.2                                             DSEG-NUM-TRANSFER,
449.3                                             WSCTLREC.
449.4
449.5        IF C-C NOT = ZERO,
449.6
449.7            MOVE ''DI''                 TO STD-CALL-RESULT-CODE
449.8            MOVE ''DMOVIN FAILED''      TO STD-CALL-RESULT-MSG
449.9
450              PERFORM DISPLAY-RESULTS-UPON-CONSOLE
450.1
450.2            GO TO END-OF-PROGRAM.
450.3
450.4* Restore parameters here
```

The Main Supervisor code to replace control data base open, gets, and puts was easily replaced by the GETDSEG, DMOVOUT, DMOVIN, and FREEDSEG intrinsic calls.

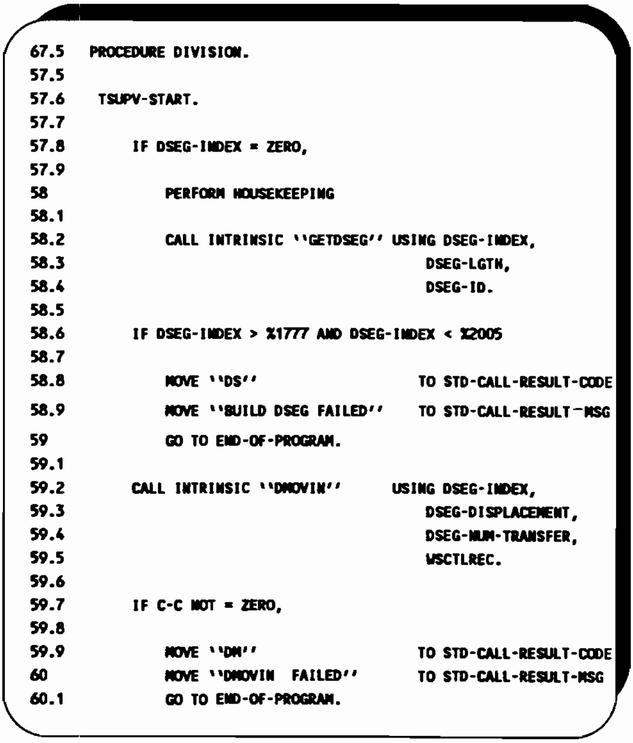
But what about the Tenant Supervisor ?
What changes did it require for an Extra Data Segment?
 And how could we eliminate its startup overhead?

SOME ADDITIONAL INTRINSICS

The initiated program needs to do some of the same things as the initiator. It must use GETDSEG to acquire access to the Extra Data Segment, and it uses DMOVIN and DMOVOUT to receive and return parameters in the Extra Data Segment.

However, to avoid startup overhead, it needs some slight modifications. First, it needs to have a way to suspend itself, rather than completely terminate. This allows it to be re-activated in the same state that it was in when it suspended. That means that any files open at the time it suspended will still be open when it is re-activated.

It must, therefore, be able to recognize whether the current activation is an initial activation, or a reactivation. This is an easy task, since the DSEG-INDEX itself becomes the switch; if non-zero, then the program was just re-initiated.



Check DSEG-INDEX for 'first time'; if so acquire DSEG and do other initialization tasks

If errors found, exit

Move in parameters

If errors found, exit

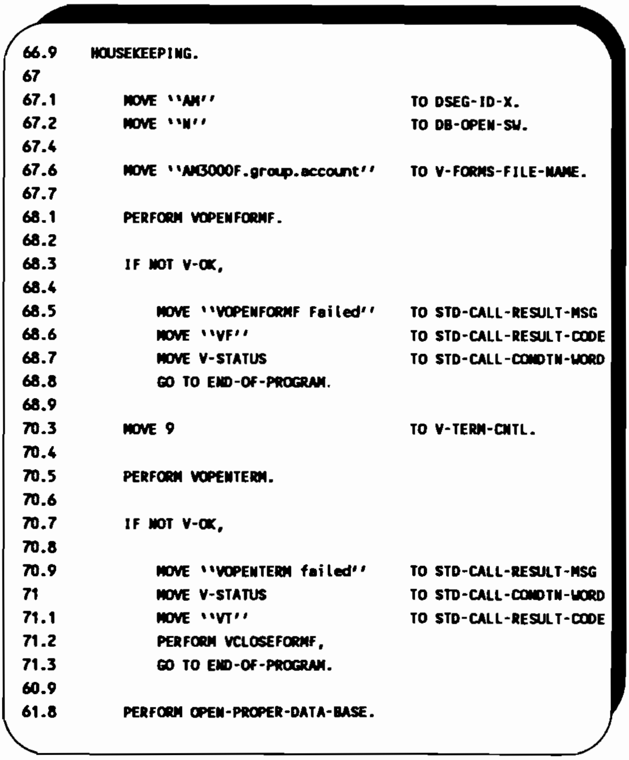
```
PROCEDURE DIVISION.

 TSUPV-START.

     IF DSEG-INDEX = ZERO,

         PERFORM HOUSEKEEPING

         CALL INTRINSIC ''GETDSEG'' USING DSEG-INDEX,
                                        DSEG-LGTH,
                                        DSEG-ID.

     IF DSEG-INDEX > %1777 AND DSEG-INDEX < %2005

         MOVE ''DS''                      TO STD-CALL-RESULT-CODE
         MOVE ''BUILD DSEG FAILED''       TO STD-CALL-RESULT-MSG
         GO TO END-OF-PROGRAM.

     CALL INTRINSIC ''DMOVIN''      USING DSEG-INDEX,
                                        DSEG-DISPLACEMENT,
                                        DSEG-NUM-TRANSFER,
                                        WSCTLREC.

     IF C-C NOT = ZERO,

         MOVE ''DM''                      TO STD-CALL-RESULT-CODE
         MOVE ''DMOVIN  FAILED''          TO STD-CALL-RESULT-MSG
         GO TO END-OF-PROGRAM.
```

SOME ADDITIONAL INTRINSICS

The housekeeping routines, which are only executed on the first activation of the program, include terminal and forms file opens, as well as a data base open.

On second, and subsequent initiations, these routines are bypassed. The GETDSEG isn't extremely time consuming, but the file opens create tremendous overhead.

The normal processing can now continue as if this were a dynamically called subprogram; if the currently open data base is the correct one (based upon the passed parameters in the DSEG), the program can proceed with the next VPLUS screen to be displayed to the user.



```
HOUSEKEEPING.

    MOVE ''AM''                          TO DSEG-ID-X.
    MOVE ''N''                           TO DB-OPEN-SW.

    MOVE ''AM3000F.group.account''       TO V-FORMS-FILE-NAME.

    PERFORM VOPENFORMF.

    IF NOT V-OK,

        MOVE ''VOPENFORMF Failed''       TO STD-CALL-RESULT-MSG
        MOVE ''VF''                      TO STD-CALL-RESULT-CODE
        MOVE V-STATUS                    TO STD-CALL-CONDTN-WORD
        GO TO END-OF-PROGRAM.

    MOVE 9                               TO V-TERM-CNTL.

    PERFORM VOPENTERM.

    IF NOT V-OK,

        MOVE ''VOPENTERM failed''        TO STD-CALL-RESULT-MSG
        MOVE V-STATUS                    TO STD-CALL-CONDTN-WORD
        MOVE ''VT''                      TO STD-CALL-RESULT-CODE
        PERFORM VCLOSEFORMF,
        GO TO END-OF-PROGRAM.

    PERFORM OPEN-PROPER-DATA-BASE.
```

There's one last thing we have to take care of; ensuring that when the program is ready to return control to the Main Supervisor, it suspends itself rather than completely terminating. This is handled through a small change to the end of program routine.

SOME ADDITIONAL INTRINSICS

The end of program routine has checks for errors discovered by processing routines, and a call to intrinsic ACTIVATE. Calling ACTIVATE with a PIN number of 0 (ZERO) indicates to MPE that you want to activate the Father of the current process.

```
                              $PAGE ''CLOSE ROUTINES''
                               END-OF-PROGRAM.

       Check for subroutine        IF ( NOT RESULTS-OK ),
                     errors
                                       PERFORM DISPLAY-RESULTS-UPON-CONSOLE.

   Set up return parameters        MOVE STD-CALL-RESULTS          TO TSUPV-STD-CALL-RESULTS.
                                   MOVE WSAPTCD                   TO CR-WSAPTCD.
                                   MOVE TSUPV-USER-PARMS          TO CR-USER-PARMS.

   If no DSEG errors, move         IF STD-CALL-RESULT-CODE NOT = ''DS'',
   them out into the DSEG
                                       CALL INTRINSIC ''DMOVOUT'' USING DSEG-INDEX,
                                                                        DSEG-DISPLACEMENT,
                                                                        DSEG-NUM-TRANSFER,
                                                                        WSCTLREC.

   If error in DMOVOUT, set        IF C-C NOT = ZERO,
      up parameters for
       diagnostic display              MOVE ''DS''                      TO STD-CALL-RESULT-CODE
                                       MOVE ''DMOVOUT FAILED''          TO STD-CALL-RESULT-MSG

                                       PERFORM DISPLAY-RESULTS-UPON-CONSOLE.

    Activate Father process        CALL INTRINSIC ''ACTIVATE''      USING 0,3.

           Check for errors        IF C-C NOT = ZERO,

                                       MOVE ''AF''                      TO STD-CALL-RESULT-CODE
                                       MOVE ''ACTIVATE FATHER FAIL'' TO STD-CALL-RESULT-MSG
                                       PERFORM DISPLAY-RESULTS-UPON-CONSOLE

                                       GOBACK.

    Go to start of program         GO TO TSUPV-START.
          when re-activated
```

The ACTIVATE of the Father suspends the current process; when re-activated, it continues with the next instruction, which takes it back to the start of the program.

SOME ADDITIONAL INTRINSICS

The following summarizes some of the key differences between called dynamic subprograms and created processes.

CALLED PROGRAMS	CREATED PROCESSES
Subprograms may reside in Segmented Libraries (SLs), or be prepped with the main program.	Programs are prepped as main programs.
Shared data bases and other files eliminate overhead associated with opens and closes.	Initial activation requires opening any required files. If not suspended upon completion, this is repeated for each creation/activation.
Parameter passing techniques are familiar to most programmers.	Parameter passing requires additional design work, but is relatively easy once mastered.
Programs execute serially; that is, the calling program suspends until the called program returns.	Created processes may execute serially, or may be executed in parallel with the creating process.

Each can be an effective technique when properly applied; the analyst must be familiar with multiple techniques to create applications that meet the user's requirements and effectively utilize the hardware/software environment of the HP3000.

SOME ADDITIONAL INTRINSICS

There are a few additional comments regarding these techniques:

1. The use of VALUE clauses in Working Storage of the initiated process can mislead you. If the process is re-activated, it begins processing in the state it was in at the time of its suspension. That means working Storage is NOT re-initialized for you by VALUE clauses.

2. Working with intrinsics requires that you check the MPE Condition Code. You must have an entry in the Special Names section to allow you to check this. A sample entry follows.

```
5.7    SPECIAL-NAMES.
5.8
5.9        CONDITION-CODE IS C-C.
```

3. Programs using Process Handling and Extra Data Segments must be prepped with these Special Capabilities indicated. To prep a program with PH and DS Capability, you must have these capabilities. The executable programs must reside in a group and account that has these capabilities.

As is true of so many design/programming techniques, the more you use them, the easier they become. And the more you learn, the more you find there is to learn.

Today's COBOL on the HP3000 provides many ways for the inventive analyst to achieve things that previously were reserved for 'Systems Programmers'.

SUMMARY

COPY LIBRARIES

COBOL II's Copy Library facility makes the use of common working storage and common procedure division routines easy. In addition to assisting in the development of error free programs, it enhances the speed of development.

The multiple library capability not only simplifies maintenance of libraries, but also eases the task of updating programs for new versions of compilers and operating systems. For example, changes to parameter sizes associated with the new intrinsics in the XL operating system can be easily accomodated in a new copy library, allowing programs to be compiled for either with minimum change.

FILE STATUS & DECLARATIVES.

The use of File Status items and the Declarative section give the programmer complete control of file system error handling. This, combined with the ability to call file system Intrinsics using the COBOL filename in place of the normal Intrinsics's *filenum* parameter allow for 'elegant' error handling, as well as provide access to many facilities previously considered too esoteric for COBOL programmers.

OTHER INTINSICS

Special Capabilities such as Process Handling and Extra Data segments can be easily utilized in COBOL II. A careful reading of the Intrinsics manual will open many doors for the creative analyst/programmer.

I hope these ideas have spurred your imagination. Hewlett Packard has given us a powerful tool for business programming in COBOL II. One of our tasks is to recognize the facilities available to us, and make use of them to provide quality systems to our users.

A Beginner's Guide to UDC's and JCW's:
How to Use Them to Your Benefit

David L. Largent
The N.G. Gilbert Corporation
P.O. Box 1032
700 South Council
Muncie, IN 47308-1032

1.0 Introduction - "Is this for me?"

How many times have you forgotten to issue a FILE command before running a program? How often have you typed a FILE command wrong? Do you ever get tired of typing the DELETESPOOLFILE command? Have you been looking for a way to make your job streams a little "smarter"? Would a way to automate some of your procedures be helpful? If you have any of these problems and have never considered using UDC's and/or JCW's, now is the time to do so.

Over the course of the following pages will be found answers to questions related to User Defined Commands (UDC's) and Job Control Words (JCW's) such as:

- What are they?
- How can they help me?
- How are they created?
- How are they used?
- What are some common problems that arise?
- Are they worth the effort of learning something new?

To find these answers, a number of example UDC's, job streams, and programs will be examined to see how and where UDC's and JCW's can be used and how they work.

This paper is directed at new users of an HP3000 who have not yet explored the world of UDC's and/or JCW's. However, any user may find some tidbits to put in their "toolbox" that will prove useful either now or in the future. This paper assumes the reader has a general knowledge of the HP3000; specifically, the following are "prerequisites" for this paper:

- Knowledge of the HP3000 accounting structure (i.e., File/Group/Account/User).
- Knowledge of how to use an ASCII text EDITOR (e.g., EDITOR/3000).
- Knowledge of many MPE commands and how to use them.
- Knowledge of job streams and how to use them.

One further note. The information provided in this paper is correct and up-to-date (to the best of my knowledge) and reflects the way UDC's and JCW's are used and work as of the G.02.04 version of MPE V/E (UB Delta 4). I am aware that changes have been made in MPE XL, but have not made any attempt to include that information in this paper.

For those of you who have chosen to stay with me, here is what you can expect. First we will examine User Defined Commands (UDC's); second, Job Control Words (JCW's) will be examined; third, we will take a look at how UDC's and JCW's can be used together; and finally, we will take a look at what we have learned and decide if it is worth the effort to put these powerful new tools to work. So. . .

2.0 "What are User Defined Commands (UDC's)?"

UDC is one of many acronyms used in the HP3000 world. This one comes from the phrase User Defined Command. A UDC is a command that is designed for some user's convenience. It is made up of standard MPE commands and/or other UDC's. A UDC can replace a single, long command or may replace a long sequence of many commands. They provide a short cut (a more precise way!?!) to accomplish some particular task. By entering the UDC name, the predefined commands are automatically executed. Think of them as just another MPE command.

2.1 "How can UDC's help me?"

A UDC may be used in most any situation where a standard MPE command may be used, even within another UDC. They can be used in both sessions and job streams. They cannot, however, be executed by using the COMMAND intrinsic or from within subsystems unless you are in BREAK. With that as background information, how can they help you?

UDC's are perfect solutions for many "problem" situations. A long sequence of MPE commands can be replaced by a single UDC name, thus decreasing the time it takes to accomplish the task (less typing), and also eliminating typing errors. A lengthy command like DELETESPOOLFILE can be replaced by a UDC name such as RPUR. A complicated sequence of commands, such as a long list of FILE commands before running a program, can be reduced to one UDC name such as PAY. Many of your boring, repetitive tasks can be automated by using UDC's. UDC's can also be used to protect your system from both the naive user and the too-knowledgeable user. They can also cause something to automatically happen at log on time. These all serve as examples of ways to make the HP3000 more "user-friendly".

By this point in time, you probably have half a dozen UDC ideas in your head just waiting to get out. So. . .

2.2 "How do I create a UDC?"

First of all, slow down! UDC's, as with most everything in life, are best created with PPP -- proper prior planning. If thought is given to each UDC you create, a lot of changes can be avoided later and you will likely end up with a more functional set of UDC's that are easier to use. For some people, creating a good UDC is an art, rather than a science -- they may put as much thought and care into a UDC as they put into developing a program.

Now that you have slowed down a bit (you have haven't you?!!), give some thought to what MPE commands get used often on your system. Are there particular tasks that always require the same sequence of commands? Are there commands or tasks that may not be difficult for us programmers to master, but may be intimidating for the user? Now take your list (you did write them down didn't you?) and evaluate each command or task. Some of them may be used so seldom that it is not worthwhile to create a UDC. (That can also be a reason to create a UDC for a task, so that the sequence of steps need not be remembered!) Some of the commands may be so short that creating a UDC for them would only save two or three key strokes. You must make a judgement call for each command or task as to whether the convenience of having the UDC available for use is worth the time and effort of creating it.

2.21 Defining the UDC.

Let us assume that you have chosen to create a UDC for the SHOWJOB command in its simplest form. You would like to be able to type the letter "J", and have the system behave as if you had typed "SHOWJOB". A UDC to do this could look like:

```
J                     Header section
OPTION LIST, HELP     Execution options section
SHOWJOB               Body section
**                    UDC separator section
```

As shown above, every UDC consists of four main sections: a header section, an optional execution options section, a body section, and a UDC separator section. The first line in this example is the header section, containing the actual command the user will type. The second line provides some execution options to the system. The third line makes up the body section, containing the actual command to be executed. The fourth line serves as a separator between UDC definitions in a UDC file.

2.211 The Header Section.

The Header Section of a UDC consists of the command name, parameters, and their optional default values. A UDC header may extend over more than one physical line or up to a maximum of 320 characters. Each line to be continued must have as its last nonblank character an ampersand (&).

The command name is what the user will actually type (along with any needed parameter values) to cause the UDC to be executed. It is composed of a maximum of sixteen alphabetic or numeric characters and must begin with an alphabetic character. The characters "RFA" may not be a UDC name nor may they be the first three characters of a UDC name, as this has been reserved by HP for internal use only. Use common sense when naming your UDC's. Use a derivative of the actual MPE command(s), or if naming a UDC for a particular task, give it a name that is descriptive of the task. It is a good idea to avoid single letter UDC names to reduce the chance of executing the command by mistake (at least for any that may be "destructive").

Parameters are variables that are assigned either a default value or a value provided by the user along with the command name. These variables are specified on the same line as the command name and may be used in the body of the UDC to make the UDC more flexible and less specific. You may have a maximum of sixteen parameters in one UDC.

Each parameter must have a name. A parameter name must begin with an alphabetic character; the remainder may be alphabetic or numeric characters. The maximum length for a parameter name is seventy characters. However, the actual maximum may be less -- a parameter name may not be split between physical lines. The same is true for default values for parameters. Further constraints on the length of the parameter name may apply because of the position where the parameter is used in the body section.

If a UDC parameter does not have a default value specified, then it is considered a required parameter and one must be provided when the user executes the UDC. If a value for a required parameter is not provided by the user, an appropriate error message is displayed and the UDC is not executed.

Let us change the requirements for our J UDC. We now want it to do one of three things:

```
     SHOWJOB JOB=@      (show all jobs)
     SHOWJOB JOB=@S     (show sessions only)
     SHOWJOB JOB=@J     (show batch jobs only)
```

This can be accomplished with one UDC if we use a parameter. The header section will now look like:

```
     J WHAT2SHOW = " "
```

The command name is still J. The character string "WHAT2SHOW" is the parameter name. A default value of " " has been specified. This means that if a user just types "J", a list of all jobs will be displayed. Alternatively, if the user types "J S" the list will show sessions only.

As can be seen in the example above, a default value for a parameter is specified by placing an equal sign (=) after the parameter name, and then following that with the default value. If the default value contains spaces or special characters, then it will need to be enclosed in quotation marks ("). If no spaces or special characters are needed in the default value, the quotation marks are optional. When a default value is not provided for a parameter, the only thing that appears in the header section is the parameter name.

If more than one parameter is used in a UDC, default values may be specified for all, some, or none of them. That is, some of the parameters can be required while others are optional. A comma (,) is used to separate a parameter name from the previous parameter name (or default value if one is specified).

From the user's point of view, there are two ways UDC parameters can be thought of: keyword and positional. When a user is executing a UDC, either approach may be used, but not both at the same time. A keyword parameter is one in which the parameter name is typed, followed by an equal sign (=), followed by the parameter value. In this way, values for parameters may be provided in any order. For example:

```
J WHAT2SHOW="S"
```

is the way the user would execute the J UDC using a keyword parameter. A positional parameter is one in which the value is specified for each parameter in the order they were defined in the UDC. If a new value is not provided for an optional parameter, its position must still be "held" by a comma (,). The J UDC executed with a positional parameter would look like:

```
J S
```

The concept of keyword and positional parameters is the same as for the standard MPE commands, except that only one approach may be used on a given execution of a UDC.

When providing numeric values for parameters (either the default values or the user's actual values), both decimal and octal numbers may be used. Octal numbers are indicated by preceding them with a percent sign (%). As you would expect, if the user provides a value for a parameter that has a default value, the user's value is the one that will be used. If the parameter value a user needs to provide contains spaces or special characters, that value will need to be enclosed in quotation marks (").

2.212 The Execution Options Section.

The execution options section of a UDC consists of choices from each of four pairs of options. Each of these four pairs has a default and, therefore, if the defaults are what is desired, this entire section can be left out. These options control the operation and use of the UDC. If the execution options section is provided, it must appear as the next line following the header section and will consist of a single line. The line must start with the word OPTION. The rest of the line contains the option or options you have chosen with a comma (,) between each one if you list more than one. The four option pairs are discussed in the following paragraphs.

LIST/NOLIST. The default is NOLIST. The LIST option will cause the text of the body section to be listed on the standard list device with the parameter values substituted as each line is executed. The NOLIST option will not list any of the text of the body section. It is a good idea to use the LIST option on UDC's that are replacements for standard MPE commands. This way the actual command is before you and you will be less likely to forget what the MPE command is when you need to use it on another system that doesn't have your UDC! The NOLIST option is generally a good choice for UDC's that are implementing user tasks.

BREAK/NOBREAK. The default is BREAK. If the NOBREAK option is chosen, the commands that make up the body of the UDC will be nonBREAKable. That is, pressing the BREAK key will not cause the command or program to stop. If the BREAK option is chosen, the commands that make up the body of the UDC will be BREAKable (if they are normally BREAKable as MPE commands). It is a good idea to use the NOBREAK option on any UDC that runs a VPLUS application. This eliminates the problems that occur when BREAK is pressed while in block mode. The use of the NOBREAK option will prohibit certain users from using any MPE commands and can serve as a way to discourage simple security breaches.

If the BREAK key is pressed during the execution of a UDC that is using the BREAK option, the following happens:

- If it was executing a nonprogram command, the UDC terminates.

- If it was executing a program, and the RESUME command is used, the UDC will also resume execution. An exception to this occurs if a SETCATALOG command is executed while in BREAK. In this case, the program is resumed but the remainder of the UDC will not be executed.

For you programmers, the NOBREAK option overrides the CAUSEBREAK intrinsic. A program containing CAUSEBREAK will not BREAK if it is executed from a UDC that has the NOBREAK option specified.

LOGON/NOLOGON. The default is NOLOGON. If the LOGON option is chosen, the commands specified in the body of the UDC will automatically be executed when the user logs on. Only one LOGON UDC at the user level will be executed. If more than one LOGON UDC exists for the user level, only the first one will execute. The LOGON option can be put to good use on user task UDC's. By using this option, the user can logon and automatically be put into a menu or application program. If the NOBREAK option is also used, you can keep a user from gaining direct access to MPE commands. An example of this will be shown later.

HELP/NOHELP. The default is HELP. If the HELP option is chosen, you may type HELP followed by the UDC name and see the definition of the UDC. If the NOHELP option is chosen, this possibility does not exist. More will be said later about the use of the MPE HELP subsystem. It is good to choose the HELP option unless you have a security sensitive situation. If you specify both the NOLIST and NOHELP options and an error occurs during execution of the UDC, the error will be reported but the line containing the error will not be listed.

2.213 The Body Section.

The body section of a UDC consists of one or more MPE commands and/or UDC's which will be executed when the user types the UDC name. Other noncommand text such as data for subsystems or application programs may not be included. A logical line of the body may extend over more than one physical line for up to a maximum of 320 characters. Each line to be continued must have as its last nonblank character an ampersand (&). The body section follows the execution options section, if any options have been listed. If the execution options section is not present, then the body section will follow the header section.

There are a few restrictions of which to be aware. The REDO command may not be used in a UDC. The DATA, JOB, and HELLO commands may be used, however, they will cause the current job or session to logoff and effectively terminate execution of the UDC; no new job or session will be initiated.

Let us go back to our last version of the J UDC. We wanted it to do one of three things:

```
SHOWJOB JOB=@
SHOWJOB JOB=@S
SHOWJOB JOB=@J
```

We last defined the header section as:

```
J WHAT2SHOW = " "
```

The body section needed to satisfy our requirements will consist of a single MPE command:

```
SHOWJOB JOB=@!WHAT2SHOW
```

In this example, all the characters preceding the exclamation point (!) will be left as shown. The character string "!WHAT2SHOW" will be replaced by either the default value of " ", or whatever the user types as a parameter value following "J". This revised command image is then what is sent to the MPE command interpreter for execution. Thus, if the user types:

```
J S
```

the command image that will be executed will be:

```
SHOWJOB JOB=@S
```

Alternatively, if the user just types:

```
J
```

and uses the default parameter value, the command image that will be executed will be:

```
SHOWJOB JOB=@
```

As is shown above, placing an exclamation point (!) immediately before a parameter name in the body section of a UDC will cause MPE to substitute the value of that parameter into that position of the command. If a parameter name does not follow an exclamation point, then an error will be reported to the user. If an exclamation point does not precede a parameter name, no substitution will be attempted, and the parameter name will be left in the command.

Normally, an exclamation point signifies a parameter name. If you want to use the exclamation point but not have MPE try to substitute a value, use an even number of exclamation points. An odd number causes MPE to attempt a substitution. Each pair of exclamation points will be translated into a single exclamation point. For example, consider the following UDC:

```
EXAMPLE PARM
OPTION LIST
COMMENT !!PARM
COMMENT !!!!!PARM
**
```

If the user was to execute it by typing

```
EXAMPLE TEST
```

the two comments would be displayed as:

```
COMMENT !PARM
COMMENT !!TEST
```

Sometimes you will need to insert a parameter into the middle of a character string in the UDC body section. As an example, the following UDC is presented:

```
PREPS PROGNAME, MAXDATASIZE=15000
OPTION LIST
PURGE !"PROGNAME"X
PREP !"PROGNAME"U,!"PROGNAME"X;MAXDATA=!MAXDATASIZE;RL=PRRL
SAVE !"PROGNAME"X
**
```

In this UDC, two parameters are used: PROGNAME and MAXDATASIZE. MAXDATASIZE has a default value of 15000. When the UDC is executed, it will purge a file, prepare a file, and then save the program file just created. The parameter name PROGNAME is enclosed in quotation marks (") to separate it from the letters "X" and "U" that follow. Without the quotation marks, there would appear to be invalid parameter names following the exclamation points. This is only a concern if the character immediately following a parameter name is a letter or number.

If the UDC was to be executed by typing:

```
PREPS PAY
```

the actual command images executed would be:

```
PURGE PAYX
PREP PAYU,PAYX;MAXDATA=15000;RL=PRRL
SAVE PAYX
```

The parameter value "PAY" corresponds to parameter PROGNAME because they are both the first item provided in the list. Since a second parameter value was not provided when the UDC was executed, the default value for MAXDATASIZE was used.

If an error is encountered as the commands of a UDC are executed, an error message will be displayed and the UDC will terminate. If the UDC is being executed in a job stream, the job stream will also terminate. Sometimes it is desirable for the UDC to continue with the next command regardless of whether the previous command resulted in an error. To accomplish this, a CONTINUE command must be inserted immediately preceding the command that may result in an error. Consider the following UDC:

```
REPORTS
OPTION LIST
CONTINUE
RUN PROG1X
RUN PROG2X
SHOWTIME
**
```

In this example, two programs are run and then the time of day is displayed. If program PROG2X results in an error (i.e., it aborts) when it is run, the remainder of the UDC will not be executed. That is, if program PROG2X aborts, nothing else will be done. Program PROG2X will always be run regardless of whether program PROG1X results in an error. The CONTINUE command tells MPE to continue with the next command, even if the current command fails. This "override" capability only applies to the command immediately following the CONTINUE command.

UDC's can perform tasks much more complex than those already covered by using sophisticated command sequences and/or referencing other UDC's from within the body section of the UDC. By using control characters and escape sequences, you can control the appearance and location of information as it is displayed by a UDC. The terminal's function keys can be "loaded" by a UDC, thus making your terminal a little more "user-friendly". Examples of these will be provided later.

As mentioned earlier, a UDC may reference or execute another UDC. This process is referred to as nesting. There is a limitation however: a UDC may only reference another UDC that is defined later in the UDC file (we will discuss UDC files in a few more paragraphs). As long as the reference occurs before the definition, UDC's can be written in a structured way.

Consider the following UDC's:

```
INFO
OPTION LIST
ME
J
T
**
J
OPTION LIST
SHOWJOB
**
ME
OPTION LIST
SHOWME
**
T
OPTION LIST
SHOWTIME
**
```

In this example, we have defined four UDC's: INFO, J, ME, and T. The INFO UDC will cause the J, ME, and T UDC's to be executed one after the other in the order listed in the INFO UDC. When the J UDC is executed, it will execute the SHOWJOB command; the ME UDC will execute the SHOWME command; and the T UDC will execute the

SHOWTIME command. So, by simply typing "INFO", the user will be provided information from all three of the MPE commands automatically.

2.214 The UDC Separator Section.

The last section of a UDC definition is the UDC separator. This consists of a single line that must start with an asterisk (*) in column one. The rest of the line is yours to do with as you wish. The purpose of this section is to separate one UDC from the next one in the UDC file and must be the last line of the UDC definition.

So, there you have it in ten words or less (give or take a few thousand). You now know how to define a UDC. But once you have a number of UDC's defined, what do you do with them? The answer to that question is. . .

2.22 Put Your UDC Definitions in a UDC File.

A UDC file is simply an ASCII text file that contains one or more UDC definitions. Any ASCII text editor may be used to create a UDC file. The record length of the UDC file may be any length; however, 72 character records is what MPE is expecting. If you set your record length less than 72, your UDC's likely will not execute properly. Using a record length larger than 72 characters will work; however, only the first 72 characters of each line will be read by MPE. A UDC file may contain any number of UDC definitions.

The UDC's should be entered in some logical sequence to make maintenance easier -- my suggestion is alphabetically by the UDC name. An exception: remember that if you are using nested UDC's, the definition of a UDC must be after its reference(s). So, either name them appropriately or include them in the file out of sequence. If you are using the escape character in your UDC's, an easy way to key it in is to type in some otherwise unused character (e.g., the ^ character) in place of the escape character. Then when you are finished entering the UDC definitions, globally change all occurrences of that character to the escape character. The reverse sequence can be used if you need to modify an existing UDC file. Just remember to change them back before you keep the file.

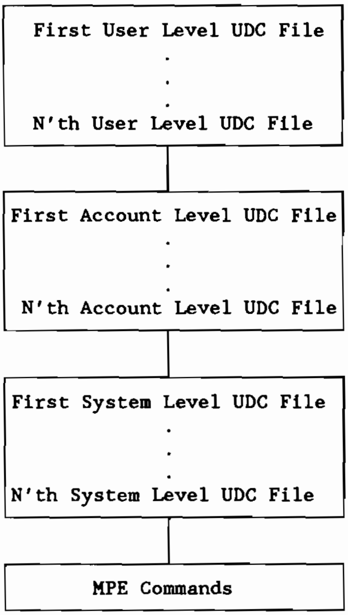
Once you have entered all of the UDC definitions using your text editor, save the file with some meaningful name. My preference is to use the letters "UDC" as the last three characters of the file name. Doing so makes it easy to locate UDC files with the LISTF command. You may keep your file with or without line numbers -- MPE does not care. You may create multiple UDC files; however, for performance reasons (which we will discuss later), it is best to limit the number of UDC files that exist for one user. If you wish, a lockword may be assigned to the UDC file.

Simply creating a UDC file with a text editor does not yet make the UDC's available for use. The next step is to tell MPE that you have a UDC file you wish to use. However, before we discuss that, let us take a look at account and system level UDC's.

2.23 Account and System Level UDC's.

Up to this point, we have been discussing UDC's in general. There are actually three different levels of UDC's: user, account, and system. A UDC file may be set up by a user for his/her own use. An account manager may create a set of UDC's for use by all users of that account. The system manager may create a set of UDC's for use by all users on the system. So now you have one more item to consider when you are creating a UDC: Who should have access to it?

When the MPE command interpreter is given a command (either an MPE command or a UDC), it follows a predefined hierarchy to decide what to do with the command in question. Shown below is that hierarchy:


First User Level UDC File
.
.
.
N'th User Level UDC File
First Account Level UDC File
.
.
.
N'th Account Level UDC File
First System Level UDC File
.
.
.
N'th System Level UDC File
MPE Commands

This hierarchy is followed regardless of whether it is trying to locate a command provided at a colon (:) prompt or a command that is coming from a UDC. This is the reason that when using nested

UDC's, the UDC definition must follow the reference(s) to that UDC.

There is an exception to this hierarchy: UDC's with LOGON listed as an option. When a user logs on, one logon UDC, at most, will be performed at each UDC level. The system level logon UDC (if present) will execute first, then the account level logon UDC (if present), and finally, the user level logon UDC (if present) will be executed. If more than one logon UDC exists at a level, only the first logon UDC at that level will be executed but will not affect the execution of other levels' logon UDC's.

Another implication of this hierarchy is that when a UDC at one level has an identical name to a UDC or command at another level, the following rules apply:

1. A UDC in user level UDC file 1 takes precedence over user level UDC file n.

2. A UDC in user level UDC file n takes precedence over account level UDC file 1.

3. A UDC in account level UDC file 1 takes precedence over account level UDC file n.

4. A UDC in account level UDC file n takes precedence over system level UDC file 1.

5. A UDC in system level UDC file 1 takes precedence over system level UDC file n.

6. A UDC in system level UDC file n takes precedence over the MPE commands.

What all of this means is that you can set up a UDC with exactly the same name as an MPE command (or UDC) and effectively redefine that command. For instance, if you wanted to keep a user from using the PURGE command, the following UDC could be established:

```
PURGE FILENAME - "$NULL"
OPTION LIST
COMMENT YOU HAVE NOT BEEN GIVEN ACCESS TO THIS COMMAND.
**
```

Unless you are specifically taking advantage of this capability, it is best to give all of your UDC's unique names. Here is another example of these rules and the hierarchy: if both a system level and a user level UDC have the same name and an account level UDC references a UDC by that name, then the system level UDC will be executed rather than the user level UDC because the system level UDC is "after" the account level UDC that referenced it, where as the user level UDC would be "before" that reference.

A user must have special capabilities to establish account or system level UDC's. To establish account level UDC's, the user must have the AM capability; to establish system level UDC's, the user must have the SM capability. Additionally, if system level UDC's are established, the system level UDC files must either have their access security released or have the access security such that all users will have READ and LOCK access to the file(s). (For security reasons, the latter is preferred.)

2.24 Telling the System About Your UDC File(s).

As stated earlier, simply creating a UDC file with a text editor does not yet make the UDC's available for use. The next step is to tell MPE that you have a UDC file you wish to use. The MPE command to do this is SETCATALOG. The simplest syntax of the SETCATALOG command is:

```
SETCATALOG udcfilename[/lockword]
```

where "udcfilename" is your UDC file name and "lockword" is an optional lockword assigned to the UDC file. This version will enable a UDC file for the user level only. The user executing the SETCATALOG command must have READ and LOCK access to the UDC file. Unless you have account or system manager capabilities, you must logon as the user that you want to enable the UDC file for. The SETCATALOG command may be issued from a session, job, or in BREAK. It may not be issued from a program, nor is it BREAKable.

When SETCATALOG is executed with a file name provided, three things occur. The command interpreter searches the UDC file for errors: problems like specifying an option that is not a valid option are caught; problems like syntax errors in the body section of the UDC are not caught. If no errors are found in a UDC, then an entry is established in a directory that will eventually contain entries for all UDC's in the UDC file. The UDC file name and optional lockword is stored in a system catalog of all UDC users. This system catalog is file COMMAND.PUB.SYS. The directory is stored in an extra data segment for that session or job. Since the lockword (if one is provided) is stored in COMMAND.PUB.SYS, a user does not need to know the lockword to logon or to access the UDC. In addition to the UDC file name (and lockword), the user name and account and the UDC level (user, account, system) are also stored in COMMAND.PUB.SYS.

As you could probably guess, the file COMMAND.PUB.SYS must exist before any SETCATALOG is attempted. If the file does not exist, the system manager or supervisor must build it. The file should be built with a record size of twenty words. The maximum number of records for the file can be determined by calculating the sum of:

1 for each user.account that will have user level UDC's; plus
1 for each user level UDC file to be used; plus

1 for each account that will have account level UDC's; plus
1 for each account level UDC file to be used; plus
1 if system level UDC's are to be used; plus
1 for each system level UDC file to be used.

So, the BUILD command used by the system manager might look like:

```
BUILD COMMAND.PUB;REC=20,32;DISC=500
```

To secure this file so that all users may utilize UDC's, but only the system manager may read or modify it (since there could be sensitive information stored there), the system manager should enter:

```
ALTSEC COMMAND.PUB;(X:ANY;R,L,W,A:CR)
```

Taking this last step will provide execute access to all users (which is all they need), and restrict all other access to the creator of the file (in this instance, the system manager).

If you have more than one UDC file that you wish to enable at the user level, the syntax of the SETCATALOG command becomes:

```
SETCATALOG udcfilename1[/lockword],udcfilename2[/lockword]...
```

The UDC file names are listed (with the optional lockwords) with a comma (,) inserted between each one. The position of a UDC in the directory determines which other UDC's it may reference (remember the hierarchy discussed a few paragraphs back?). The UDC files are opened and scanned in the order they appeared in the last SETCATALOG command for that UDC level. All user level UDC's are entered into the directory first, followed by all account level UDC's, and finally, all system level UDC's.

So, now you know how to "turn on" or enable user level UDC's. What if you want to disable or "turn off" user level UDC's? To accomplish this, you again use the SETCATALOG command, except this time do not provide a UDC file name:

```
SETCATALOG
```

When a UDC file name is not provided with the SETCATALOG command, all entries and references to any user level UDC files in the UDC directory and COMMAND.PUB.SYS that are currently enabled for the user issuing the command are eliminated. The UDC files themselves, however, are not purged. To re-enable your UDC files, you would simply type the SETCATALOG command, followed by your UDC file name(s) again. Disabling user level UDC's does not have an immediate affect on other job or sessions that are still logged on with the same user name. They may continue to use the disabled UDC's until they logoff. Conversely, if you enable some UDC's, other users will not have access to them until they logon again.

Now, suppose sometime previously you have typed:

```
     SETCATALOG UDC1,UDC2,UDC3
```

You, therefore, have the UDC's from three UDC files enabled (plus any account or system level UDC's!). For whatever reason, you no longer need the UDC's in file UDC2. To eliminate them, but still keep the others, you need to type:

```
     SETCATALOG UDC1,UDC3
```

Anytime the SETCATALOG command is executed, it has the effect of removing all user level UDC's for the user entering the command. Additionally, if a UDC file name(s) is provided, those UDC's are re-cataloged and available for use.

At some point in time (probably very soon after you start using UDC's), you will need to modify or add to an existing UDC file. There is not much more involved than simply using a text editor to make the changes or additions; however, you do need to know about a "gotcha". Because of the way UDC's are enabled, whenever a user is logged on to the system, any UDC file(s) associated with that user (including account and system level UDC files) are considered "open". What this means is that you may not alter that UDC file in any way as long as the user is logged on and has the UDC file enabled. You may not PURGE, RENAME, or (in EDITOR) modify and KEEP that UDC file. Any attempt to alter the file will result in the error message "EXCLUSIVE VIOLATION".

So, how do you ever make changes or additions?!? You have five options (take your pick -- they all work well under different situations):

1. Have all users (and jobs) that are using the UDC file log off (except you of course). You use the SETCATALOG command to disable the UDC file in question. Make your changes to the file, and keep it. Re-enable the UDC file by using the SETCATALOG command. Notify the users they may log back on.

2. Have all affected users (including yourself) disable the UDC file by using the SETCATALOG command. Make your changes to the file, and keep it. Have all users re-enable the UDC file by using the SETCATALOG command - or - you use the SETCATALOG command and have the other affected users logon again. Watch out for batch jobs when using this option.

3. Have all affected users (and jobs) logoff the system. You logon so that you can modify and keep the UDC file, but as a user that does not have this UDC file enabled. Make your changes to the file, and keep it. Notify the users they may log back on.

4. Logon so that you can modify and keep the UDC file. Make your changes to the file and keep it under a new file name. Later, after all affected users and jobs have logged off, disable the UDC file, PURGE the old UDC file, RENAME the new file to the old file, and re-enable the UDC file with the SETCATALOG command.

5. Any combination or variation of the above four options that you can get to work.

It is important to understand that all affected users must take some action (logoff/on or SETCATALOG) because the SETCATALOG command only takes effect for the session or job that issues it and for future logons. It does not affect other jobs or sessions currently logged on.

You now know everything (well almost everything) I know about using the SETCATALOG command with user level UDC files. But what about account and system level UDC files? How do they get enabled and disabled for use? The answer again is the SETCATALOG command! We add one more parameter to the previous syntax:

```
SETCATALOG udcfilenamel[/lockword][,udcfilename2/[lockword]]...;ACCOUNT
SETCATALOG udcfilenamel[/lockword][,udcfilename2/[lockword]]...;SYSTEM
```

The ACCOUNT parameter specifies that the UDC file(s) being enabled should be available for all users in the account. This parameter requires the account manager capability. Reading between the lines, you have probably realized that you must logon to the account that the UDC file is to be enabled for. For example, if you want to enable UDC file ACCTUDC for account XYZ, first logon as the account manager of account XYZ, then enter:

```
SETCATALOG ACCTUDC;ACCOUNT
```

If you wish to disable the account level UDC files, you would enter:

```
SETCATALOG;ACCOUNT
```

Again, you must have the account manager capability to use the ACCOUNT parameter. Taking this new information into account, all previous information pertaining to the SETCATALOG command and user level UDC files also applies to account level UDC files.

The SYSTEM parameter specifies that the UDC file(s) being enabled should be available for all users of the system. This parameter requires the system manager capability. For example, if you want to enable UDC file SYSUDC for all users of the system, just logon as the system manager, then enter:

```
SETCATALOG SYSUDC;SYSTEM
```

If you wish to disable the system level UDC files, you would enter:

```
SETCATALOG;SYSTEM
```

Again, you must have the system manager capability to use the SYSTEM parameter. Taking this new information into account, all previous information pertaining to the SETCATALOG command and user level UDC files also applies to system level UDC files.

If you have the account manager or system manager capability, you may use another parameter of the SETCATALOG command:

```
SETCATALOG udcfilename1[/lockword][,udcfilename2[/lockword]]...
           ;USER=user[.account]
```

where "user" is a user name and "account" is an account name. This allows you to enable a UDC file for a user other than the one you are logged on as. Account managers may enable UDC files for any user in their account. The system manager may enable UDC files for any user on the system. They will not take effect, however, until the next time the user logs on.

This parameter may also be used to disable UDC files for other users:

```
SETCATALOG;USER=user.[account]
```

The same restrictions and capabilities apply for account managers and the system manager, as were just discussed above.

One last parameter that you may wish to know about:

```
SETCATALOG ...any other parameters...;SHOW
```

This will list the UDC file names and UDC definitions in each of those files as they are scanned. This is helpful for locating where an error is occurring when you use the SETCATALOG command, but I don't suggest it for general use. It can take a while to list all of the UDC definitions in a large UDC file. Additionally, any account or system level UDC's are listed after the user level UDC's are scanned (remember -- they go into the user's directory too!).

2.25 What happens when I logon and have some UDC's enabled?

At logon time, any user that has one or more UDC file(s) enabled for his/her use (user, account, or system!) will cause a fair amount of CPU and disc utilization to occur. As a result of previous SETCATALOG's, the file COMMAND.PUB.SYS contains the UDC file names enabled for every user, as well as those enabled for each account and the system. At logon time, this file is searched to identify what UDC files are enabled for the user in question. As each one is identified, that UDC file is opened and the contents of it are read. As the UDC file is scanned, entries are created and placed in a directory of UDC's for the user. That directory is created in an extra data segment for the session or job.

When you enter a command or UDC name, the command interpreter searches the UDC directory (the extra data segment) for that which you typed in, starting at the beginning of the directory. Remember that entries are placed in the UDC directory in user level, account level, system level sequence. If the command is found in the UDC directory, the UDC body for that command is read one line at a time and parameters are substituted into the line where appropriate. The command interpreter is then re-entered at a special internal entry point to interpret the new expanded command string and goes through the same steps just mentioned, except that this time, the UDC directory scan begins with the directory entry that follows the UDC currently being executed. (Now do you understand why the definition of a UDC must be after its reference?) If the command interpreter fails to find a match for a command string in the UDC directory, it then checks to see if it is a valid MPE command. This cycle is repeated until the end of the UDC definition that is being executed.

2.26 Looking at what UDC's are available.

There is a handy MPE command which will list all of the UDC names enabled for your session or job:

```
SHOWCATALOG [listfile]
```

where "listfile" is the name of the file (disc or printer) you wish the list to be sent to. Unless directed elsewhere with a prior FILE command, if you specify "listfile", the listing is sent to device class LP. If "listfile" is not provided, the list of UDC names will be displayed on $STDLIST (your terminal for a session). This command may be issued from a session, job, or in BREAK. It may not be issued from a program. The SHOWCATALOG command is BREAKable (it aborts execution of the command).

The output from the SHOWCATALOG command lists the UDC's currently enabled for your use, the level at which they are defined (user, account, system), and the file name in which they reside. A sample execution follows:

```
:SHOWCATALOG
UDC1.GROUP.ACCT
     AA                  USER
     BB                  USER
UDC2.GROUP.ACCT
     CC                  USER
ACCTUDC.PUB.ACCT
     DD                  ACCOUNT
     SYSUDC.PUB.SYS
          EE                       SYSTEM
          FF                       SYSTEM
          GG                       SYSTEM
```

This example shows seven UDC's enabled from four different UDC files: three at the user level, one at the account level, and three more at the system level.

The SHOWCATALOG command has another parameter that can be useful, especially for account and system managers:

```
     SHOWCATALOG [listfile];USER=user[.account]
```

where "user" is the name of a user, and "account" is an account name. This parameter permits you to specify a user other than yourself for which you want the SHOWCATALOG done. The output when this parameter is used will consist of only enabled UDC file names and which level they were defined at. No UDC names will be listed.

For example if you want to know what UDC file(s) are enabled for a user, you could type:

```
     SHOWCATALOG;USER=DAVE
```

and the system would respond with:

```
     USER UDC CATALOG FILE NAMES:
          UDC1.GROUP.ACCT
          UDC2.GROUP.ACCT

     ACCOUNT UDC CATALOG FILE NAMES:
          ACCTUDC.PUB.ACCT

     SYSTEM UDC CATALOG FILE NAMES:
          SYSUDC.PUB.SYS
```

There are a few limitations when using the "USER=" parameter:

1. A user with neither account nor system manager capabilities may only specify his/her own user name. A user may not obtain information for any other user.

2. A user with account manager capability may specify any user in his/her account. Additionally, if "@" is used for the user name, only the account level UDC file name(s) will be displayed.

3. A user with system manager capability may specify any user on the system. Additionally, if "@" is used for the user name and an account name is specified, the account level UDC file name(s) will be displayed. Also,

if "@" is specified for both the user and account, only the system level UDC file name(s) will be displayed.

A way of getting more information about a particular UDC is to use the MPE HELP command. Unless the NOHELP execution option was specified when the UDC was created, if you type

```
HELP udcname
```

where "udcname" is the name of a UDC, you will receive a listing of the UDC definition. For example, if you typed:

```
HELP J
```

(and the UDC we discussed sometime back was enabled), the system would respond with:

```
USER DEFINED COMMAND

J WHAT2SHOW=" "
OPTION LIST
SHOWJOB JOB=@!WHAT2SHOW
```

If the NOHELP execution option is specified when a UDC is created, and the HELP command is used for that UDC, the system will respond with:

```
Can't find anything under this command or in the table of contents.
```

unless the UDC name is the same as an MPE command, in which case you will receive information about the MPE command. It should be noted that you may not enter the HELP subsystem itself and get information about your UDC's. Only by using the HELP command as described earlier will you receive this information.

2.3 How about some examples?

Good Idea! I'll present some here; also see section four for examples of using JCW's and UDC's together. NOTE: Anytime you see the characters "<esc>" in this paper, read it as an escape character. That is, for purposes of this paper, I have used the characters "<esc>" in place of the escape character. However, if you were to type in the UDC, you would need to use the escape character instead. Additionally, the control character will be represented by the characters "<ctrl>".

We have been at this for a while, so let's take a look at some games -- I mean UDC's for a games user. I have a variation of the following UDC file set up on my system:

```
StartGames
Option LogOn
Display " <esc>&dA******************************************* "
Display " <esc>&dA*                                         * "
Display " <esc>&dA*    WELCOME TO THE HP 3000 GAME ROOM     * "
Display " <esc>&dA*                                         * "
Display " <esc>&dA******************************************* "
Display " "
Display " Type LIST to get a listing of the available games   "
Display " "
**
Games
Display " ADVENT.... (Adventure) A game of exploration.       "
Display " AMORT..... Creates amortization (loan) tables.      "
...continued for other games available...
**
Advent
Continue
Run ADVENT
**
Amort
Display "<esc>&dBNOTE: If you want the loan schedule printed on paper,  "
Display "<esc>&dB      wait for this phrase to appear...                 "
Display "<esc>&dB        ENTER THE LISTING DEVICE (RETURN FOR $STDLIST)?"
Display "<esc>&dB      then press the BREAK key. You will get a colon(:)"
Display "<esc>&dB      Type AMORTPRINT, and press the RETURN key. The    "
Display "<esc>&dB      screen will now say...                            "
Display "<esc>&dB        READ pending                                    "
Display "<esc>&dB      Now type LP, press the RETURN key, and go on to   "
Display "<esc>&dB      answer the other questions.                       "
Display "<esc>&dB                                                        "
Display "<esc>&dB      If you just want the schedule to print on the     "
Display "<esc>&dB      screen, press the RETURN key instead of doing all"
Display "<esc>&dB      of the above.                                     "
Continue
Run AMORT
Reset List
List
**
AmortPrint
File LIST=AMORTRPT;Forms=8 1/2 by 11 INCH PAPER REQUIRED <ctrl>G.
Resume
**
...continued for other games available...
Stop
Abort
**
List
Display "  Available Games...                                   "
Display "      ADVENT      AMORT     ANIMAL     BIOSIN          "
Display "      BIOSINP1    BJ        CHESS      FIVEROW         "
Display "      FOOTBALL    GRIC      LANDERP    OTHELLO         "
Display "      SAHARA      STARTREC  TREK2640   UBOAT          "
Display "      ULTIMA
Display " "
Display "  If you would like a one line description of each of"
Display "  the games, type GAMES, and press the return key.   "
Display " "
**
```

Items of interest in the example UDC file above:

1. The LOGON execution option is used so the welcome message appears automatically.

2. No, there is not an MPE command DISPLAY! Display is a system level UDC I have set up to cause the parameter value provided to be displayed by using the COMMENT command. (We will take a look at the UDC definition for DISPLAY later.

3. Escape sequences are used to control the appearance of text on the screen. In the case of STARTGAMES, it will blink, and in AMORT, the text will be shown in inverse video.

4. The CONTINUE command is used in all of the individual game UDC's to guarantee that the LIST UDC will always be executed, even if the program aborts.

5. The RESUME command is used in AMORTPRINT to automatically put them back in the program after they press the BREAK key.

6. Notice that no options were provided in most UDC's. Since the default is NOLIST, the command(s) will not be displayed on the screen. However, in UDC DISPLAY, OPTION LIST is used so that the messages passed to it will be displayed.

7. The LIST UDC must be at the end after all of the individual game UDC's because each one of them ends with a reference to LIST.

8. The STOP UDC exists to provide the user a "graceful" way to end a game they do not wish to continue. The users know to press the BREAK key and type STOP if they find themselves in this situation.

O.K. We have had our fun with the games. Time to get back to some serious work. The next series of examples come from some of my application systems. These UDC's are generally user level UDC's.

First, a few UDC's from the accounts payable system. In UDC file APINUDC, I have the following UDC:

```
      Payable
      Option LogOn,NoBreak
      Continue
      Payable
      Bye
      **
```

In UDC file APUDC, the following UDC's exist (among others):

```
      Payable
      Option NoBreak
      File TODAT=TODAT.GLXEQ;Shr
      File AP830WRK=VENNAMES;Rec=-80,16,F,ASCII;Disc=32,1,Save
      Run PAYABLEX;Lib=G
      **
      VendorMaint
      Option NoBreak
      Display "Use this program with extreme care.<ctrl>G"
      Run AP429X;Lib=G
      **
```

Items of interest in the accounts payable examples:

1. There are two classes of users for most of the application systems: A "data entry" user who only needs access to one main menu-driven program (for example, PAYABLEX) and a user who needs access to that same program, as well as other utility programs (for example, AP429X) for the application system. I have implemented this by creating two UDC files. The "data entry" user has both UDC files enabled in the order listed above. However, the other user only has the second UDC file enabled. In this way, the UDC to actually run the program PAYABLEX only needs to be entered in the second UDC file (in this example that UDC is relatively short; however, some others have numerous FILE commands), with a reference made to it in the first, rather than needing to define it twice.

2. Even though the "data entry" user has both UDC files enabled for them, they effectively only have one thing they can do: RUN the PAYABLEX program. This is because of the LOGON execution option which automatically starts up the program at logon time, and logs them off (because of the BYE command) when the program stops running. Two more items are required to guarantee that this will work, however. The NOBREAK execution option must be specified to keep the user from pressing the BREAK key and gaining access to MPE. The CONTINUE command is also needed, so that if the PAYABLEX program should abort for some reason (I know - you write perfect programs that never abort - I have problems sometimes though!), the next command (in this case, BYE) will still execute.

Here is a neat idea for use with the SORT program (it works well with many others too!):

```
EquipmentSort
Purge EQPSORTD
Run SORT.PUB.SYS;StdIn=SDEQPSRT;StdList=$NULL
**
```

Most any program can have its standard input and output files redirected (i.e., provided from, or sent to, some place other than normal). See the RUN command for more details on this. In this UDC, the interactive input is coming from a disc file named SDEQPSRT, and the interactive output is discarded. File SDEQPSRT contains the commands that would normally have been provided to the SORT program (i.e., the names of the input and output files, and the key information).

Now, we will look at some UDC's from our job costing system. It has not been converted to an interactive system yet, and is still using the ENTRY.PUB.SYS program to create batch files which are then processed. The UDC's do provide some good examples of nesting UDC's, however.

```
EditJobList BFName
Option NoBreak
Reformat !BFName
Sort !BFName, SDEDTLST
File JCWRK,New;Temp
File CARD=SORTFILE,OldTemp
Run JC010X
File JCWRK,OldTemp
File JCMSTR,Old
File LIST=JC020LST;Cctl;Dev=LP
Run JC020X
**
EditWeeklyCost BFNAME
Option NoBreak
Reformat !BFName
Sort !BFName,SDEDTCST
File CARD=SORTFILE,OldTemp
Run JC110X
**
Reformat BFName
Purge !BFName,Temp
File REFFILE=JCREF.REFFILE,Old
File BATCH=!BFName.FCDATA,Old
File OUTFILE=!BFName,New;Temp
File REFLIST=$NULL
Run REFORMAT.PUB.SYS
**
```

```
SORT BFName,SDName
Purge SORTFILE,Temp
File INPUT=!BFName,OldTemp
File OUTPUT=SORTFILE,New;Temp
Run SORT.PUB.SYS;StdIn=!SDName;StdList=$NULL
**
```

Items of interest in these job costing UDC's:

1. UDC's EDITJOBLIST and EDITWEEKLYCOST both reference UDC's REFORMAT and SORT; however, different information is provided for the parameter values, so they process the files differently.

2. UDC REFORMAT only needs to know what the name of the batch file is that was created with ENTRY.PUB.SYS. The output from the REFORMAT program is stored in a temporary file with the original batch file name.

3. UDC SORT needs to know the batch file name and the name of the "sort data" file. The input data file for the sort is actually the temporary file created while reformatting. The output data file is always named SORTFILE, and is a temporary file. Temporary files are used for two reasons: first, they are automatically PURGEd when the session or job logs off; and secondly, they permit two sessions or jobs to do the same thing at the same time and eliminate the need to worry about permanent file usage conflicts. The "sort data" file contains information about how to sort the input file, and is specified by using the STDIN parameter of the RUN command (see the previous example for further discussion of this).

Here are some examples from the financial system that show how to execute the same program, but provide for different input and/or output.

```
BalanceSheet
BalShRun 11,";Forms=BLANK8 1/2 by 11 INCH PAPER REQ <ctrl>G."
**
BalanceSheetGB
BalShRun 2
**
BalShRun Copies,Forms=" "
File BALSHLST;Dev=LP,,!Copies;!Forms
Run BALSHX
**
CorporatePL
PrintPL C,12
**
DivisionPL
PrintPL 1,13
**
```

```
ClientPL
PrintPL 2,9
**
PrintPL FileCode,Copies=1
File PLFLA=PLFL!FileCode,Old
File PRPALLST=PRPL!"FileCode"LST;Dev=LP,9,!Copies;&
    Forms=BLANK 8 1/2 by 11 PAPER REQ <ctrl>G.
Run PRPALX
**
```

Here is what to look for in these examples:

1. There are seven UDC's, but only two actually RUN a program; the other five just reference those two.

2. In the case of the balance sheet program (BALSHX), we need to print the report on both blank paper and regular "green bar" paper. The program takes very little time to run, so we run it twice, once with each UDC, to create two different spool files with a different number of copies for each. In one case, the FORMS parameter and value is provided, and in the other case, the default value is used.

3. In the case of the profit and loss program (PRPALX), we need to print a different number of copies of each type of statement. Additionally, because of the way the program is set up, there is a different input file for each type of statement. There is actually only one character different in each file name, so the FILECODE parameter is used to provide that.

4. In UDC PRINTPL, notice the use of quotation marks (") around the FILECODE parameter. This is because it needs to be substituted in the middle of a "word".

5. In that same UDC, also notice the use of the ampersand character (&) to continue the logical line onto the next physical line of the UDC. Were all of that to have been typed on the same line, it would have gone beyond the seventy-two character limit for a line.

What if you have two different users that need to use the same program, but do not use the same "terminology"? Create two UDC's that RUN the same program. Our invoicing/accounts receivable system provides a good example. We have one set of users that work with the invoicing part of the system and a second set that works with the accounts receivable part of the system. It is really all one application system, but the two user groups tend to think of them as (related, but) separate entities. So. . .

```
Invoices
InvRecRun
**
ARec
InvRecRun
**
InvRecRun
Option NoBreak
File ENAME.PUB;Shr
File TODAT=TODAT.GLXEQ;Shr
Run INVRECX;Lib=G
**
```

In this example, each user group has a UDC that makes sense to them but both end up at exactly the same place!

Since most of you are probably programmers, let us take a look at some UDC's that you may find useful in your day-to-day existence. This list is not meant to be exhaustive, but rather meant to get you started:

```
DBLoad
Option List
Run DBLOAD.PUB.SYS
**
DBSchema SchemaFile
Option List
File DBSTEXT=!SchemaFile
File DBSLIST;Dev=LP
Run DBSCHEMA.PUB.SYS;Parm=3
Reset DBSTEXT
Reset DBSLIST
**
DBUnLoad
Option List
Run DBUNLOAD.PUB.SYS
**
DBUtil
Option List
Run DBUTIL.PUB.SYS
**
FormSpec
Option List
Run FORMSPEC.PUB.SYS
**
Preps ProgName,MaxDataSize=15000
Option List
Purge !"ProgName"X
Prep !"ProgName"U,"ProgName"X;MaxData=!MaxDataSize
Save !"ProgName"X
**
```

```
Query
Option List
Run QUERY.PUB.SYS
**
RunLG Program
Option List
Run !"Program"X;Lib-G
**
```

There is not a whole lot to explain here -- nothing fancy -- just some examples of commands and tasks that you probably do most every day. The only item I will mention is the use of OPTION LIST which displays the command as it is executed. This helps remind me that I am using a UDC and that some day, some where, I may need to type this longer command.

Now for some UDC's that the system operator would likely find useful:

```
AfterChecks
OptionList
DownLoad 6,VFCSTD6.PUB.SYS
DownLoad 6,Margin-1
StartSpool 6
**
BackUp
Option List
Limit 2,1
File BCKUPCNF.BACKUP-BCKUPCNF.PUB.SYS
Run BACKUP.HPUNSUP.SUPPORT;Info-" "
AbortJob HPTREND,MGR.TELESUP
Limit 1,1
**
BeforeChecks
Option List
Continue
StartSpool LP
StopSpool 6
HeadOff 6
**
JobF Priority-0
Option List
JobFence !Priority
**
LoadVFC VFCFileName-VFCSTD6
Option List
DownLoad 6,!VFCFileName.PUB.SYS
**
Margin LeftColumn-1
Option List
DownLoad 6,Margin-!LeftColumn
**
```

```
OffSites
Option List
Limit 2,1
Stream OFFSITEJ.PUB.SYS
AbortJob HPTREND,MGR.TELESUP
Limit 1,1
**
OutF Priority-1
Option List
OutFence !Priority
**
RCop FileNumber,Copies-1
Option List
AltSpoolFile #0!FileNumber;Copies-!Copies
**
RDef FileNumber
Option List
AltSpoolFile #0!FileNumber;Pri-0
**
RPri FileNumber,Priority-8
Option List
AltSpoolFile #0!FileNumber;Pri-!Priority
**
RPur FileNumber
Option List
DeleteSpoolFile #0!FileNumber
**
StopSp Device-6
Option List
StopSpool !Device;OpenQ
**
StrtSp Device-6
Option List
StartSpool !Device
**
SysDown
Option List
ShowJob
ShowOut Job=@;Sp
Console
ShowTime
**
SetOpKeys
SFK 1 0 " Reply  " "         " 6 "Reply "
SFK 2 0 "  Redo  " "         " 4 "Redo"
SFK 3 0 "  Show  " " Cache   " 9 "ShowCache"
SFK 4 0 " System " "   Up    " 5 "SysUp"
SFK 5 0 "  Show  " "Reports " 17 "ShowOut Job=@; Sp"
SFK 6 0 "  Run   " " Spook5 " 18 "Run Spook5.Pub.Sys"
SFK 7 0 " Spook5 " " Detail " 8 "S @.@;@0"
SFK 8 0 " System " "  Down   " 7 "SysDown"
UserKeys
**
```

Just the same as the last examples, this is by no means an exhaustive list, but rather a starting point for you as you consider what makes sense on your system. Now for some items of interest:

1. We print our checks "hot" to the printer. That is, we stop the spooler process for the printer itself. Two UDC's make life easier in this situation: BEFORECHECKS and AFTERCHECKS. Nothing fancy; just some steps that have to be done over and over -- and done CORRECTLY every time.

2. BACKUP and OFFSITES are two UDC's that assist with performing SYSDUMPs. The BACKUP program referenced is a program that used to exist in the HPUNSUP group of the SUPPORT account before the days of the TELESUP account. Its sole purpose in life is to create and stream a batch job that either performs a full or a partial backup based on the day of the week and the information stored in the file BCKUPCNF. In recent years, HP has provided us with the FULLBACKUP and PARTBACKUP commands which nearly eliminated my need for the BACKUP program. My problem is that we perform two full backups every week: one stays on site; the other goes off site. The PARTBACKUP command performs a partial SYSDUMP since the last full backup. However, since my second full backup is off site, I really want the partial backups done after it to still go back to the last on site full backup. The BACKUP program permits me to handle this, the HP commands do not. At any rate, each of the UDC's set the job and session limits low, stream an appropriate job stream, and then abort the HPTREND job stream so that it is not running during the backup.

3. In the SETOPKEYS UDC, the SFK UDC is referenced. This will be discussed later. For the time being, just understand that the SETOPKEYS UDC will cause the terminal function keys to be loaded with this information.

If you are getting tired of examples, feel free to go on to the next section. For those of you who want more examples, we will next look at some of the UDC's from my system level UDC file, and then finish up with some unique "goodies" that may prove useful to some of you:

```
AboJ JSNumber
Option List
AbortJob #!JSNumber
**
AltJ JobNumber,InPriority=8
Option List
AltJob #J!JobNumber;InPri=!InPriority
**
Con LDev=" "
Option List
Console !LDev
**
```

```
Ed
Option List
Editor
**
Entry
Option List,NoBreak
ListF BF@,0
Run ENTRY.PUB.SYS
**
Files
Option LogOn, List
File LP;Dev=LP
File T;Dev=TAPE
Display "The above File commands are in effect."
Display "The system UDC's are enabled."
**
J WHAT2SHOW=" "
Option List
ShowJob Job=@!WHAT2SHOW
**
L FileSet="@",Detail=2,ListFile=$STDLIST
Option List
ListF !FileSet,!Detail;!ListFile
**
LEq
Option List
ListEq
**
List FileName,ListFile=$STDLIST
Option List
FCopy From=!FileName;To=!ListFile
**
LT FileSet="@",Detail=2,ListFile=$STDLIST
Option List
ListFTemp !FileSet,!Detail;!ListFile
**
LUDC
Option List
ShowCatalog
**
Me
Option List
ShowMe
**
Out Items="Sp"
Option List
ShowOut Job=@;!Items
**
Print FileName,Copies=1,Priority=8
Option List
File LISTING;DEV=LP,!Priority,!Copies
FCopy From=!FileName;To=*LISTING
Reset LISTING
**
```

```
PScreen
Option List
Run PSCREEN.PUB.TELESUP
**
Purges FL1,FL2=$NULL,FL3=$NULL,FL4=$NULL,FL5=$NULL,FL6=$NULL
Option List
Purge !FL1
Purge !FL2
Purge !FL3
Purge !FL4
Purge !FL5
Purge !FL6
**
Res
Option List
Resume
**
RunPS Program
Option List
Run !Program.PUB.SYS
**
SetUDC File1=$NULL,File2=$NULL,File3=$NULL,File4=$NULL,File5=$NULL
Option List
SetCatalog !File1,!File2,!File3,!File4,!File5
**
ShC
Option List
ShowCache
**
ShD LDev=" "
Option List
ShowDev !LDev
**
Sort Input,Output
Option List
File INPUT=!Input
File OUTPUT=!Output
Reset LIST
Run SORT.PUB.SYS
**
Spook5
Option List
Run SPOOK5.PUB.SYS
**
Str JobName,Char="!"
Option List
Stream !JobName,!Char
**
StrAt JobName,Time,Char="!"
Option List
Stream !JobName,!Char;At=!Time
**
```

```
StrDay JobName,Day,Time="0:0",Char="!"
Option List
Stream !JobName,!Char;Day=!Day;At=!Time
**
StrIn JobName,Days=0,Hours=0,Minutes=0,Char="!"
Option List
Stream !JobName,!Char;In=!Days,!Hours,!Minutes
**
T
Option List
ShowTime
**
Display Message=" "
Option List
Comment <esc>M!Message
**
```

A few notes and points of interest:

1. In the few situations where we still use the ENTRY program to create batch files, we use the naming convention of always starting the batch file name with the letters "BF". In the ENTRY UDC, the purpose of the LISTF command is just to provide a list of existing batch files to the user before the program starts running.

2. When I want a quick listing of a file on my screen or on paper, I use the LIST or PRINT UDC's. Both use the FCOPY command to produce the listing. One slight inconvenience (besides the "extra garbage" displayed): unless the file record length is eighty (for LIST) or 132 (for PRINT) characters, FCOPY gives you a warning message that you must respond to.

3. The DISPLAY UDC is used to display the character string provided in the MESSAGE parameter. The escape M sequence causes the cursor to delete the line that the cursor is on, then display the message. The end result is that only the message is left on the screen. Note that this UDC is at the end of the UDC file so that all other UDC's may use it to display messages.

O.K. Now for those unique "goodies" I promised you. Have you been looking for a way to load information into your terminal's function keys automatically? If you have, keep reading. If you have not, perhaps you want to know why you would want to. Using the function keys to execute commands is a good alternative to UDC's because the overall overhead is usually less. Function keys also provide a one or two keystroke execution of commands. Why am I telling you about an alternative to UDC's?!? Because the solution is accomplished with UDC's.

My original source for this information was from the November 1987 issue of the Interact magazine. In that issue, Michael J. Parker and Lynn

Wilson of State Farm Insurance in Bloomington, Illinois, had a short article in the Users' Forum section of the magazine. I have taken their ideas and expanded them to work on all of the types of terminals I have (HP 2645A, HP 2392A, and HP 700/92), and, I believe, on any HP terminal. There are three parts to the solution: two UDC's that need to be defined once -- probably in a system level UDC, and one or more additional UDC's defined that use the first two. The two system level UDC's should be defined as:

```
SFK Key=1;Attr=0;Head1="        ";Head2="        ";&
    Length=40;Function="                                        "
Option List
Comment <esc>&f!"Attr"a!"Key"k!"Length"L!Function<esc>M<esc>A
Comment <esc>&f!"Key"k16d0L!Head1!Head2<esc>M<esc>A
**
UserKeys
Option List
Comment <esc>&jB<esc>M<esc>A
**
```

The SFK UDC accepts the information for one function key and "loads" that information by causing it to be displayed on the terminal with the COMMENT command. Be careful when you type this one in: upper and lower case makes a difference in how it will execute! The KEY parameter signifies which function key (1 through 8). The ATTR parameter indicates what should happen when the function key is pressed:

0 - (Normal) The defined string is displayed. To execute it, the user must press the RETURN key.
1 - (Local Only) The defined string is displayed; however, it may not be executed.
2 - (Transmit) The defined string is displayed and immediately executed.

The HEAD1 and HEAD2 parameters provide values to be placed in the labels on the screen (only for terminals that can "label" the function keys). LENGTH indicates how many characters are in the function string. And finally, the FUNCTION parameter provides the actual character string to be "loaded" into the function key.

I have used the COMMENT command twice in this UDC because we have a mixture of terminals and not all of them have the capability of labelling the function keys. The first (longer) COMMENT command will work on all of the terminals and will cause all of the information to be loaded except for the function key labels on the screen. The second COMMENT command provides the additional label information to those terminals that can accept it (the older model terminals just ignore it). It must be split in two steps; if combined into one, the older model terminals will not have any of the information loaded into the function keys. If all of your terminals have the function key labelling capability, you may combine them into one COMMENT. On the other hand, if none of your terminals have this capability, the second COMMENT could be left out and the HEAD1 and HEAD2 parameters could be eliminated.

The USERKEYS UDC simply causes the function key labels to be displayed on the terminal screen (if they are not already). Again, this is ignored by the older model terminals, but is needed for the newer ones. The "<esc>M<esc>A" sequence in both UDC's effectively "erases" the comments from the screen as the UDC executes. If you would like them left on the screen, that sequence could be left off the end of the line.

What might the UDC's that use these look like? Here is an example:

```
SetMainKeys
SFK 1,0,"        "," QUERY  ",,"Run QUERY.PUB.SYS"
SFK 2,0,"        "," DBUTIL ",,"Run DBUTIL.PUB.SYS"
SFK 3,2,"        ","SHOWJOB ",,"ShowJob"
SFK 4,0,"        ","FORMSPEC",,"Run FORMSPEC.PUB.SYS"
SKF 5,0,"        "," EDITOR ",,"EDITOR"
SFK 6,0,"        "," SPOOK  ",,"Run SPOOK5.PUB.SYS"
SFK 7,0,"        ","SEGMENTR",,"Run SEGMENTER.PUB.SYS"
SFK 8,0,"PrepSave","Program ",,"Preps "
UserKeys
**
```

If you want to take this a step further (although you do get back your saved UDC overhead), instead of RUNning each of the above programs, execute a UDC to do so. In the UDC for each program, execute a UDC before and after the RUN command to load the keys for the program about to be run and then reset them afterward. For example:

```
Query
SetQueryKeys
Run QUERY.PUB.SYS
SetMainKeys
**
DBUtil
SetDBUtilKeys
Run DBUTIL.PUB.SYS
SetMainKeys
**
```

Do remember to place these in the UDC file prior to the SETMAINKEYS and each of the SETxKEYS UDC's. By using this nesting technique, you can set up a "menu" system with only UDC's and terminal function keys. Neat, huh?!?

Now for "goodie" number two. Have you ever wanted to get rid of a set of files but did not really want to type PURGE over and over and did not have access to MPEX? Now you can (maybe)! This idea came from the March 1988 issue of The Chronicle newspaper. Victoria Shoemaker (of Taurus Software) in her :NEWUSER column, presented this novel solution to the problem:

```
     PurgeFS FileSet
     Option List
     Store !FileSet;$Null;Show;Purge
     **
```

This UDC STOREs the file set you specify to $NULL (which does nothing -- you do not even need to REPLY to a request!) and then PURGEs the files afterwards. This format of the STORE command is normally used to archive information and then remove it from the system -- we just happen to be archiving to the "bit bucket". Any file set that the STORE program will accept (including the "-" option) can be provided to this UDC. Be careful with this one -- it can be very powerful. Make sure you have a good backup before you type PURGEFS "@.@.@"!

"Goodie" number three. Do you need a way to provide different "welcome" and/or "news" messages for each user? If so, read on. This one comes from an article M.E. Kabay (of JINBU Corporation) wrote in the March 1988 issue of The Chronicle. Here's a system level UDC:

```
     SysMessage
     Option LogOn
     Run LIST.PUB.TELESUP;INFO="W ON;T OFF;L NEWS.PUB.SYS"
     **
```

If an account needs a special message or "news" file, set up an account level UDC for them:

```
     AcctMessage
     Option LogOn
     Run LIST.PUB.TELESUP;INFO="W ON;T OFF;L NEWS.PUB.ACCT"
     **
```

Special needs for a user? Try this user level UDC:

```
     UserMessage
     Option LogOn
     Run LIST.PUB.TELESUP;INFO="W ON;T OFF;L NEWS.GROUP.ACCT"
     **
```

The FCOPY program could be used instead of the LIST program, but I think you'll find the LIST program a little "nicer". By having all of these separate files, you can easily provide different information to different users or accounts by including it in their own news file. To update a file, simply use your favorite text editor.

One last "goodie". This one permits you to send a message or list the contents of an entire file on any terminal screen that is turned on but not logged on:

```
Send LDev,Source
File TERM;Dev=!LDev
Continue
FCopy From=!Source;To=*TERM
Reset TERM
**
```

This UDC, when executed, uses the FCOPY program to copy the specified file to the specified terminal. Again, the destination terminal must be turned on but logged off for the message to be displayed. If $STDIN is provided for the SOURCE parameter, then the user may type whatever he/she wishes at the time of the UDC execution. (This can be a little tricky though -- there is no prompt and you must type ":EOD" to end your message.)

A situation I find this UDC helpful in is after a backup has completed, and I need to let users know that they can logon. Consider the following:

```
SendToAll Source
Send 22,!Source
Send 23,!Source
Send 24,!Source
Send 25,!Source
Send 26,!Source
Send 27,!Source
Send 28,!Source
Send 29,!Source
Send 30,!Source
Send 31,!Source
Send 32,!Source
Send 33,!Source
Send 34,!Source
Send 35,!Source
**
```

If I type:

```
SENDTOALL SYSTEMUP.PUB.SYS
```

then it will attempt to transmit the contents of file SYSTEMUP.PUB.SYS to each terminal. If a particular terminal is not turned on, or is already logged on, that FCOPY will fail, but because of the CONTINUE command, the next one will still be attempted.

2.4 What are some problems I may have while using UDC's?

Throughout the paper, I have provided a number of warnings and "gotchas". Listed here (in somewhat random order) are a few worth repeating and a few not mentioned previously.

If a system and a user level UDC have identical names and an account level UDC references a UDC by this name, then the system level UDC will be executed because of the UDC hierarchy.

If an error or warning occurs as a UDC executes, MPE will:

1. Print an appropriate error message.

2. Unless NOHELP is specified, print a caret (^) pointing to the error.

3. Unless NOHELP and NOLIST are specified, print the line in which the error occurred.

Every time a user logs on, a UDC directory is created for that session or job. If an error occurs during this initialization, only the UDC level in which the error occurs will fail to be initialized. All others will still be enabled.

A UDC name may not be "RFA" or start with the letters "RFA". This is reserved for HP's internal use. Any UDC that is "RFA" or starts with "RFA" will not execute and will result in the error message:

```
UNKNOWN COMMAND NAME (CIERROR 975)
```

If the SETCATALOG is executed as part of a UDC, it will be the last command executed in the UDC body. Additionally, if the SETCATALOG was part of a nested UDC, all levels of UDC execution are terminated after completion of the SETCATALOG command.

If you execute a UDC that RUNs a program, you press BREAK, and then execute the SETCATALOG command while in BREAK, you may type RESUME, and continue with that program; however, any further execution of the UDC that issued the RUN command will be terminated.

UDC's are not always as secure as you might think. Certain programs and subsystems (e.g., SPOOK) allow users to enter MPE commands and RUN programs. So even if you have a UDC with LOGON and NOBREAK specified, the user can still gain access to MPE.

When you run a program, you no longer have access to your UDC's. The COMMAND intrinsic only can be used to execute MPE commands.

Consider using the CONTINUE command wherever possible. This will help prevent a program from aborting and terminating the UDC execution. Even though it may seem unlikely that a program will abort, it can be accomplished in many programs by typing :EOD when prompted for input. This causes an end of file condition on $STDIN and gives many programs problems.

Even though you have disallowed certain commands to a user (by redefining them with a UDC), be careful. If the user has access to the COMMAND intrinsic (e.g., through EDITOR), the user can still execute most MPE commands.

When modifying a UDC file, make sure that all users (sessions and jobs) accessing that file have either logged off or disabled UDC's with the SETCATALOG command. If you are working on a system level UDC file, that means every user on the system is affected!

If you get rid of a user (with the PURGEUSER command) that had UDC's enabled at the time, the entries are NOT removed from COMMAND.PUB.SYS. Always execute a SETCATALOG command to disable UDC's for the user before purging the user.

UDC's bring with them system resource overhead at logon time, and they use up entries in the DST table. The DST table entries are used because of the extra data segment used to store the UDC directory for each job or session. To reduce overhead and improve system performance when UDC's are used, do whatever you can to reduce the number of UDC files. This will reduce the number of DST's used, as well as reduce the number of FOPEN's.

Here is a question you have not asked yet: is there a maximum number of UDC's that may be enabled for a user? The answer is yes. Every UDC enabled for a job or session must have an entry placed in the UDC directory in an extra data segment. When that extra data segment becomes full, that is the maximum number of UDC's.

If you keep these potential problems and limitations in mind as you start your adventure into the wonderful world of UDC's, you should do well in avoiding most of the problems and pitfalls that may arise along the way.

2.5 Are we done with UDC's yet!?!

Yes! At least for the time being. And now for something completely different. . .

3.0 What are Job Control Words (JCW's)?

JCW is one of the many other acronyms used in the HP 3000 world. This one comes from the phrase Job Control Word. A JCW is MPE's way of permitting programs and commands to communicate with each other within a given job or session. JCW's are unsigned integer variables used at the operating system level with values ranging from zero through 65,535. Each JCW has a name and can be set and/or interrogated either by MPE commands and/or programs.

3.1 How can JCW's help me?
-or-
Why would I want a program to talk to my commands?

Good questions! A properly used JCW will permit you to create "smarter" job streams. They can help to automate some of the decision making process in procedures. They can even help you catch errors before they become a problem! All of this is to say: JCW's can help make the system more "user friendly".

By testing JCW's against specific values, the user can program conditional statements that take action(s) based on the results of the test. JCW's can be set to predetermined values to indicate completion of steps within a procedure. JCW's can be checked to determine if certain events (usually errors) have occurred within MPE.

3.2 O.K. I think I see how they could help me. So, how do I create and use a JCW?

First, some background information. Three classes of JCW's exist: user-defined JCW's; system-defined JCW's; and system-reserved JCW's. In some ways, they are exactly the same -- in other ways, they are completely different.

User-defined JCW's are named and assigned values solely by the user. MPE never changes the value of, or interrogates a user-defined JCW. The user creates and assigns a value to this class of JCW's with the SETJCW command or the PUTJCW intrinsic. The JCW name must begin with an alphabetic character and may consist of a maximum of 255 alphabetic or numeric characters. You may not begin a JCW name with the mnemonic names OK, WARN, FATAL, or SYSTEM except under very specific conditions. (If you want to know what they are, see the HP "commands" manual.) The value assigned to a user-defined JCW must be in the range of zero to 65,535 inclusive. User-defined JCW's may be interrogated by the user with the SHOWJCW command and the FINDJCW intrinsic. These new commands and intrinsics will be discussed later, so please be patient.

System defined JCW's are named by the system and assigned values by the system and/or by the user. Both the system and the user may interrogate system-defined JCW's. Only two system-defined JCW's exist: JCW and CIERROR. Both are created and set to zero at the beginning of every job or session. They will remain zero unless the user changes their value or an error occurs. The JCW named JCW has two special values:

%140000 (System 0)	Program aborted per user request.
a value greater than %140000	Program terminated in an error state.

The CIERROR JCW keeps track of the command interpreter (CI) errors. If a CI error occurs, CIERROR is set to reflect the most recent error number. Valid commands do not reset CIERROR to zero. Thus, it always contains the number of the last error that occurred, unless the user resets its value. Generally, it is best not to alter the values of the system-defined JCW's. If you need to control a JCW, it is best to use a user-defined JCW.

System-reserved JCW's are named and assigned values solely by the system. Users may not change the value of a system-reserved JCW. They may, however, interrogate it. There are six system-reserved JCW's: HPMINUTE, HPHOUR, HPDAY, HPDATE, HPMONTH, and HPYEAR. The following briefly explains what each is:

HPMINUTE	Minute of the hour: values are 0 through 59.
HPHOUR	Hour of the day: values are 0 through 23.
HPDAY	Day of the week: values are 1 through 7; Sunday = 1.
HPDATE	Day of the month: values are 1 through 31.
HPMONTH	Month of the year: values are 1 through 12; January = 1.
HPYEAR	Year of the century: values are 0 through 99.

3.21 JCW usage in jobs and/or sessions.

O.K. So now you know what JCW's are. You even know about the three classes of JCW's and what who can do to what. But, how do you look at or set their values? For jobs or sessions, the answer is: with the SHOWJCW and SETJCW commands.

The SHOWJCW command displays the current value of one or more JCW's. Its syntax is:

```
SHOWJCW [jcwname]
```

where "jcwname" is a valid JCW name (any class). If a name is provided, then only the value for that JCW will be displayed. If a name is not provided, then all system-defined and user-defined JCW's and their values are displayed -- system-reserved JCW's are not displayed. This command may be executed from a session, job, in BREAK, or from a program. It is BREAKable (it aborts execution of the command).

If no user-defined JCW's have been created and the user types:

```
SHOWJCW
```

the system will respond with

```
JCW=0
CIERROR=0
```

unless some error has occurred prior to this command.

If you wish to see the current value of a specific JCW, you might type:

```
SHOWJCW HPDAY
```

and the system would respond with:

```
HPDAY=3  SYSTEM RESERVED JCW
```

Or, if you typed:

```
SHOWJCW UPDATEERRORS
```

and UPDATEERRORS was a valid user-defined JCW name, the system would respond with:

```
UPDATEERRORS=2
```

Big deal, so you can look at the value of a JCW. So what?!? O.K., let me tell you how you can change or set the value of a JCW -- that is a little more productive. We need the SETJCW command to do this:

```
                         [+value]
SETJCW jcwname=value[-value]
```

where "jcwname" is the name of a new or existing user- or system-defined JCW and "value" represents one of the following:

1. An octal number between zero and %177777, inclusive.
2. A decimal number between zero and 65,535, inclusive.
3. An MPE-defined JCW value mnemonic (OK, WARN, FATAL, or SYSTEM)
4. The name of an existing JCW.

All values must be in the range of 0 to 65,535, inclusive. That is, if the "+" or "-" option is used, the result of the arithmetic must be in the range as well. (The equal sign following the "jcwname" may actually be one or more punctuation characters or spaces, except "%" and "-". If you prefer some other notation, feel free. . .) This command may be executed from a session, job, in BREAK, or from a program. It is not BREAKable.

A word or two about the four JCW value mnemonics. They are:

OK	value is zero
WARN	value is 16,384
FATAL	value is 32,768
SYSTEM	value is 49,152

These are strictly mnemonics for specific values -- they cannot be used as JCW names. You may use a combination of a mnemonic and a number to indicate a value between two mnemonics. If you specify:

```
FATAL32
```

for example, an implied addition takes place (32,768 + 32) and the value would be 32,800. The "+" and "-" option may also be used with mnemonics. For example:

```
FATAL - 768
```

would result in a value of 32,000. If the SHOWJCW command is used to display current JCW values, and a value is greater than one of the mnemonics, then the value will be displayed as the mnemonic plus the amount over. For example, a value of 16,386 will be displayed as:

```
WARN2
```

An exception to this is that any value less than 16,384 will be shown as the actual number.

When the SETJCW command is executed, it causes the MPE JCW table to be scanned for the name of the specified JCW. If the name is found, the JCW is set to the value provided. If the name is not found, it is added to the JCW table and then set to the value provided. Once a JCW is created, it exists for the duration of that session or job. There is no way to delete a JCW, short of logging off.

You still are not feeling very productive with JCW's yet, right!?! O.K. Here is the good stuff. JCW's are most often used to control the flow of batch jobs (they can also be used in UDC's and/or in sessions), taking various actions based on the results of previous steps. To do this, the IF/THEN, ELSE, and ENDIF commands are used. For purposes of this paper, I am going to assume you either know how these MPE commands work or can easily acquire the knowledge as we look at examples. Some examples will come later that should clear up some of your questions.

3.22 JCW usage in programs.

Now, for you programmer-type people, we will take a look at how to interrogate and set JCW's from within a program. My examples will be based on COBOLII usage; however, I will try to provide information in a general way.

The programmatic equivalent to the SHOWJCW command is the FINDJCW intrinsic. Its syntax (in COBOLII format) is:

```
CALL INTRINSIC "FINDJCW" USING jcwname,jcwvalue,status
```

where "jcwname" is an alphanumeric variable (byte array) containing the name of the JCW to be found, "jcwvalue" is an unsigned one-word integer variable (logical) to which the JCW value is returned and "status" is a signed one-word integer variable (integer) to which a value denoting the execution status of the intrinsic is returned. The "jcwname" parameter may contain up to 255 alphanumeric characters, starting with a letter and ending with a nonalphanumeric character, such as a blank. If the requested JCW is found, its value is returned to the program in the "jcwvalue" parameter; if not found, no change is made to this parameter. The "status" parameter will be returned with one of four possible values:

0 Successful execution; the JCW was found.
1 Error: "jcwname" is longer than 255 characters.
2 Error: The value of "jcwname" does not start with a letter.
3 Error: The JCW was not found in the JCW table.

The FINDJCW intrinsic may be used to return the value of any of the three classes of JCW's.

The SETJCW command's programmatic equivalent is the PUTJCW intrinsic. Its syntax (in COBOLII format) is:

```
CALL INTRINSIC "PUTJCW" USING jcwname,jcwvalue,status
```

where "jcwname" is an alphanumeric variable (byte array) containing the name of the JCW to be created or changed, "jcwvalue" is an unsigned one-word integer variable (logical) containing the value for the JCW, and "status" is a signed one-word integer variable (integer) to which a value denoting the execution status of the intrinsic is returned. The "jcwname" parameter may contain up to 255 alphanumeric characters, starting with a letter and ending with a nonalphanumeric character, such as a blank. If "@" is the value used, all JCW's will be set to the value provided. If the specified JCW already exists in the JCW table, its value is changed to the value provided in the "jcwvalue" parameter; if not there, an entry is created and then it is assigned the specified value. The "status" parameter will be returned with one of six possible values:

0 Successful execution; value entered in the JCW table.
1 Error: "jcwname" is longer than 255 characters.
2 Error: The value of "jcwname" does not start with a letter.
3 Error: JCW table overflow; no room to create this new JCW.
4 Error: Attempted to assign a value to a JCW value mnemonic.
5 Error: Attempted to assign a value to a system-reserved JCW.

The PUTJCW intrinsic may only be used to assign a value to a user-defined or system-defined JCW. As mentioned earlier, if the JCW named JCW is set to exactly %140000, then when the program stops running, the system will display:

```
PROGRAM ABORTED PER USER REQUEST (CIERR 989)
```

If it is set to any value greater than %140000, then the message displayed will be:

```
PROGRAM TERMINATED IN AN ERROR STATE (CIERR 976)
```

If the program is running in a batch job, the job will terminate unless a CONTINUE command precedes the RUN command.

Another possibility with JCW's is to use them to permit separate processes within the same job or session to communicate with each other. If a process were to set a JCW to a given value when a certain event occurred, then any other related process could check that JCW to find out when it occurred or what has occurred. Remember, however, that only numeric information may be assigned to JCW's.

Just so you can't say that I didn't tell you about them, two other intrinsics exist: GETJCW and SETJCW. They only permit you to interrogate and set, respectively, the value of the system-defined JCW named JCW. Since they have limited usability and you can use FINDJCW and PUTJCW to accomplish the same thing, I do not suggest learning about them.

3.3 How about some examples?

Once again, you have a good idea! First, we will look at some examples in batch jobs, then we will take a look at a couple example programs that use JCW's.

3.31 Batch Job Examples.

Since most of us are likely programmers, I will start with an example job stream I use when working with a program. Virtually all programs that are created at N.G. Gilbert Corporation are written in the PROTOS language. For those of you not familiar with PROTOS, it is a program generator whose output is a complete, structured, COBOLII program. Once the program has been written in PROTOS (and keyed into an editor file), the next step is to have PROTOS create the COBOLII source program. When PROTOS completes that task, then you need to compile the COBOLII source program. The final step is to prepare the object program to produce the program file.

Since I do not want to tie up a terminal (and the system) while all of this transpires, I use an "intelligent" job stream to handle the various tasks.

```
!Job prog,PRGRMR.NGG
!Comment This job stream performs a ProWrite, COBOLII compile,
!Comment PREPare, and SAVE of a program.  If any errors
!Comment are encountered along the way, a message is sent
!Comment to PRGRMR.NGG, and the job stream stops running.
!SetJCW CIError = OK
!Continue
!ProWrite progP,prog,$STDLIST,2000
!If CIERROR < > OK Then
!   Tell PRGRMR.NGG; ProWrite of prog aborted<ctrl>G.
!Else
!   If PROTOSError < > OK Then
!      Tell PRGRMR.NGG; Errors in ProWrite of prog<ctrl>G.
!   Else
!      Tell PRGRMR.NGG; ProWrite of prog done. Compile started.
!      File COPYLIB=PROCOPY
!      Continue
!      COBOLII prog, progU
!      If CIError < > OK Then
!         Tell PRGRMR.NGG;Complie of prog aborted <ctrl>G.
!      Else
!         Tell PRGRMR.NGG; Compile of prog done. Prepare started.
!         Purge progX
!         Continue
!         Prep progU,progX;MaxData-20000
!         If CIError < > OK Then
!            Tell PRGRMR.NGG;Prepare of prog aborted <ctrl>G.
!         Else
!            Save progX
!            Tell PRGRMR.NGG;Program prog is done <ctrl>G.
!         EndIf
!      EndIf
!   EndIf
!EndIf
!EOJ
```

This example demonstrates how you can structure a job stream to check for errors and take different actions based on the occurrence or non-occurrence of errors. Some points of interest:

1. JCW CIERROR is set to OK (i.e., zero) at the start of the job to guarantee that it starts at zero.

2. Even if one of the programs (PROWRITE, COBOLII, or PREP (SEGMENTER) aborts, the job stream will continue because of the CONTINUE commands. This permits us to check CIERROR after each one to see if it aborted. If the CONTINUE commands were not used, the job stream would abort before we could check the JCW's.

3. PROTOSERROR is a JCW that the PROWRITE program creates and sets equal to the number of errors found in your PROTOS program. By checking it, we can determine whether it is worthwhile to continue on with the compile and prepare steps.

4. Notice that you may nest the IF statements to create whatever logic that might be required.

Here is another variation of this same job stream. This version just tells the programmer that there is an error, but does not report in detail like the previous version:

```
!Job prog,PRGRMR.NGG
!SetJCW CIError=OK
!SetJCW False=1
!SetJCW NoProblems=OK
!Continue
!ProWrite progP,prog,$STDLIST,2000
!If CIError < > OK Then
!   SetJCW NoProblems=False
!Else
!   If PROTOSError < > OK Then
!      SetJCW NoProblems=False
!   Else
!      File COPYLIB=PROCOPY
!      Continue
!      COBOLII prog, progU
!      If CIError < > OK Then
!         SetJCW NoProblems=False
!      Else
!         Purge progX
!         Continue
!         Prep progU,progX;MaxData=20000
!         If CIError < > OK Then
!            SetJCW NoProblems=False
!         Else
!            Save progX
!         EndIf
!      EndIf
!   EndIf
!EndIf
!If NoProblems=False
!   Tell PRGRMR.NGG;Errors in program prog <ctrl>G.
!Else
!   Tell PRGRMR.NGG;Program prog is done.
!EndIF
!EOJ
```

In this version, JCW NOPROBLEMS is set to indicate when an error has occurred and then is checked at the end of the job stream to determine what to tell the programmer. (Note the creation of JCW FALSE.)

Here is an example of making use of the system-reserved JCW's. This is a modified version of the job stream we use to do our daily backups.

```
!Job BackUp,Operator,Sys
!SetJCW Monday=2
!ShowAllocate
!Run FREE5.PUB.SYS
!ShowCache
!Report
!If HPDay=Monday Then
!   FullBackUp
```

```
!Else
!   PartBackUp
!EndIf
!Stream JHPTREND.HP35136A.TELESUP
!If HPDay=Monday or HPDay=Monday+3 Then
!   Stream PREDICTJ.HP05093A.TELESUP
!EndIf
!EOJ
```

Points of interest in the backup job stream:

1. Notice the creation of the JCW MONDAY. This is not necessary, but it makes the IF command read a little nicer.

2. If the JCW HPDAY has a value of 2, then it is Monday and a full backup should be performed. Otherwise, a partial is done.

3. When the backup has completed, the HP trend job stream is restarted, and if it is Monday or Thursday (Monday + 3), then the HP predict job stream is also started.

Here is an example of using the IF command and JCW's to do something they were not designed for, but it works, so why not?!! The following could be included at any point in a job stream where JCW would not be equal to FATAL (which would be virtually anywhere):

```
!If JCW=Fatal Then
   From this point on (up to an ELSE or ENDIF command), you
   may type whatever you wish.  The lines do not even need to
   start with an exclamation point!  The reason this works is
   that when the condition in an IF command is false (which
   it is in this instance), all command lines are ignored
   until an ELSE or ENDIF command is read.  Thus, this
   provides an easy way to include comments without using
   the COMMENT command.
!EndIf
```

Two other sources of example job streams are the PREDICTJ and JREDUCE job streams from HP. They make extensive use of JCW's to control the flow of the job stream logic.

3.32 Programmatic Examples.

The following is a COBOLII subroutine (actually it was written in PROTOS) I created to call in any situation where I want to end a program and have:

```
PROGRAM ABORTED PER USER REQUEST (CIERR 989)
```

displayed afterwards (granted, this is not very often!). The sole purpose of this program is to set JCW to a value of %140000.

```
$CONTROL DYNAMIC,BOUNDS
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SETABORTJCW.
 AUTHOR. DAVID L LARGENT.
 DATE-WRITTEN. TUE, OCT 20, 1987,   3:41 P.M.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

******************************
*Sets JCW to "Program Abort".*
******************************

 01 JCW-NAME                                PIC X(4)
    VALUE "JCW " .
 01 JCW-STATUS                              PIC S9(4) COMP .
 01 JCW-VALUE                               PIC 9(4) COMP
    VALUE %140000 .

 PROCEDURE DIVISION.

 MAIN-LINE-SECTION SECTION.

 MAIN-LINE.

     MOVE ZERO                         TO JCW-STATUS.
     CALL INTRINSIC "PUTJCW" USING JCW-NAME JCW-VALUE
          JCW-STATUS.
     IF JCW-STATUS NOT = ZERO
          DISPLAY "Program SETABORTJCW:   JCW not set."
     .
     GOBACK.
```

This may not be a very useful program to you, as it is printed; however, it does show the basics of what needs to be done to create and/or set a particular JCW to a given value. If your JCW name is longer than three characters, make sure you increase the length of field JCW-NAME. Also, make sure you have at least one blank or some other non-alphanumeric character following your JCW name. The value for JCW-VALUE can be specified as a decimal number if you prefer.

Here is a trivial example of the FINDJCW intrinsic, but again it shows the basics of what needs to be done to retrieve the value of an existing JCW.

```
$CONTROL BOUNDS
 IDENTIFICATION DIVISION.
 PROGRAM-ID. DISPLAYJCW.
 AUTHOR. DAVID L LARGENT.
```

```
       DATE-WRITTEN. WED, APR 27, 1988,  4:24 A.M.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

      *****************************
      * Displays the Value of JCW *
      *****************************
        01 JCW-NAME                           PIC X(4)
           VALUE "JCW ".
        01 JCW-STATUS                         PIC S9(4) COMP.
        01 JCW-VALUE                          PIC 9(4) COMP.
        PROCEDURE DIVISION.

        MAIN-LINE-SECTION SECTION.

        MAIN-LINE.

           MOVE ZEROS                         TO JCW-STATUS
                                                 JCW-VALUE.

           CALL INTRINSIC "FINDJCW" USING JCW-NAME JCW-VALUE
                JCW-STATUS.
           IF JCW-STATUS NOT = ZERO
                DISPLAY "Program DISPLAYJCW: JCW not found."
           ELSE
                DISPLAY "JCW = " JCW-VALUE
           .
           STOP RUN.
```

Same comments as last time: make sure you set up JCW-NAME large enough to hold your JCW name and make sure you end it with at least one blank or other non-alphanumeric character.

3.4 What are some problems I may have while using JCW's?

Throughout the paper, I have provided a number of warnings and limitations. Listed here (in somewhat random order) are a few worth repeating.

A JCW name must start with a letter and may consist of a maximum of 255 alphabetic or numeric characters.

The value assigned (whether a number or a calculated value) must be in the range of zero to 65,535, inclusive.

The system-reserved JCW's may not be assigned a value by a program or the user.

If you want to check for program errors (e.g., program aborts), make sure to include a CONTINUE command before the RUN command.

Remember that the only thing that changes the values of CIERROR and JCW (except the user) is another error. That is, a valid command does not set them to zero!

Although JCW's can have any value from zero through 65,535, only values from zero through 16,383 will be displayed as numbers. Values larger than 16,383 will be displayed as a JCW value mnemonic and an offset beyond the mnemonic value.

4.0 How can I use UDC's and JCW's together?

UDC's and JCW's can be used together very effectively. JCW's can control the flow of logic within a UDC. A UDC (by way of its parameters) can gather information and use that information to set JCW's such that a program can then interrogate them and take appropriate actions. So, let us look at some examples of combining these two powerful capabilities.

To start off with, we will look at some logon UDC's. First, a simple addition to the system level logon UDC to make use of system-reserved JCW HPDAY easier:

```
SysLogOn
Option LogOn,NoBreak
SetJCW Sunday=1
SetJCW Monday=2
SetJCW Tuesday=3
SetJCW Wednesday=4
SetJCW Thursday=5
SetJCW Friday=6
SetJCW Saturday=7
**
```

By including this, these seven JCW's would always be available for use in IF commands. For example:

```
If HPDay=Monday Then
     FullBackUp
Else
     PartBackUp
EndIf
```

is easier to read and understood than "If HPDay=2...".

Next, a way to have one thing automatically happen if a session logs on and something else if a job logs on:

```
UserLogOn
Option Logon,NoBreak
SetJCW CIError=0
Continue
Resume
```

```
If CIError = 978 Then
     Run (batch program)
Else
     Run (online program)
     Bye
EndIf
**
```

The RESUME command is not allowed in job mode (thus, CI error 978), so we can use the result of its execution to determine if a session or a job is logging on. Also, note that a job could be set up to do other things after the "logon" program runs because there is not an automatic BYE like there is for the session.

Here is a way to control when people logon and play games:

```
GamesLogOn
Option LogOn,NoBreak
If HPDay = Sunday or HPDay = Saturday or &
     HPHour < 8 or HPHour > 17 or HPHour = 12 Then
     Display "Welcome to the Game Room"
Else
     Display "Sorry, the Game Room is closed."
     Display "Hours: Saturday and Sunday: all day"
     Display "       Monday-Friday: Before 8AM, After 5PM"
     Display "                    and Noon to 1PM"
     Bye
EndIf
**
```

If it is Saturday or Sunday, or before 8AM, after 5PM, or sometime during the noon hour, this UDC will let the user stay logged on. Any other time and the user will automatically be logged off. Note that if the user gets logged on during an "open" time, they may continue playing "forever" -- there is nothing to force them off when the game room "closes".

The following UDC is executed every morning by our operator or anytime the machine is restarted. It provides a quick, easy way of getting everything set to a known value:

```
SysUp
Option List
Streams 10
JobFence 0
JobPri CS,DS
JobSecurity Low
Continue
DiscRPS 3,Disable
Continue
StartCache 1
Continue
```

```
StartCache 2
SetJCW CIError=0
Continue
StartCache 3
If CIError=0 Then
     Stream JHPTREND.HP35136A.TELESUP
Else
     Display "<esc>&dJ The HPtrend job stream should be"
     Display "<esc>&dJ running.  If it is not,         "
     Display "<esc>&dJ STREAM JHPTREND.HP35136A.TELESUP"
EndIf
ShowJob JOB=@J
OutFence 1
HeadOff 6
Continue
StartSpool 6
Limit 3,16
SetOpKeys
**
```

A few points of interest:

1. Since we leave disc caching turned on virtually all of the time, this UDC assumes that if the "StartCache 3" command succeeds, the system must have just been started and, therefore, the trend job stream needs to be initiated.

2. The last thing that is done is to load the function keys on the system console with the SETOPKEYS UDC that was discussed earlier.

Here are a couple of UDC's to make things more convenient:

```
FindRun Program,Parm=0
SetJCW CIError=0
Continue
Run !Program.Group1;Parm=!Parm
If CIError=622 Then
     SetJCW CIError=0
     Continue
     Run !Program.Group2;Parm=!Parm
     If CIError=622 Then
          SetJCW CIError=0
          Continue
          Run !Program.Group3;Parm=!Parm
          If CIError=622 Then
               Display "This program was not found in"
               Display "group1, group2 or group 3."
          EndIf
     EndIf
EndIf
**
```

```
     List FromFile=$StdIn,ToFile=$StdList
     SetJCW CIError=0
     Continue
     File PRINT;Dev=!ToFile
     If CIError=301 or CIError=344 Then
          File PRINT=!ToFile
     EndIf
     FCopy From=!FromFile;To=*Print
     **
```

The FINDRUN UDC will try RUNning your program from three different groups before it "gives up". This could be changed to be different accounts as well, and could have more nesting added to try in more locations if you wish.

The LIST UDC provides a lot of flexibility. The "to" file can be specified by providing a device class name, a logical device number, or a file name! To list a file on your terminal, you would type:

```
     LIST file
```

To print a file on the line printer, you could type either one of these:

```
     LIST file,LP
     LIST file,6
```

To list a file on a terminal with a logical device number of 27 (the terminal would need to be turned on, but logged off), you would type:

```
     LIST file,27
```

To copy a file to an existing disc file, you would type:

```
     LIST file,file2
```

It is not possible to create a new disc file. Another option available with this UDC is to use the default "FromFile" value of $STDIN, which will accept input from your terminal keyboard, thus permitting you to "create" a file as you go.

Now let us take a look at a UDC in which information is passed to a program by way of JCW's. This example is from the UDC's that PROTOS Software Company provides with their PROTOS system:

```
     PROWRITE F,C="C",L=$NULL,S=1023,R=ROOTDB,Q=0
     SETJCW PROBUILDWRITE=2
     SETJCW QEDITOUT=!Q
     FILE SEMDOPE=SEMDOP01.PROTOS.PROTOS;SHR
     FILE SEMPASS=SEMPAS01.PROTOS.PROTOS;SHR
     FILE SSERR=SSERR.PROTOS.PROTOS;SHR
```

```
FILE SEMTEMP-SEMTMP01.PROTOS.PROTOS;SHR
FILE ATNIN1-!F
FILE ATNOUT2 - !L
PURGE !C
BUILD !C;REC--80,16,F,ASCII;CODE-EDTCT;DISC-!S
FILE COBOLOUT-!C
FILE ROOTDB-!R
IF QEDITOUT-1 THEN
     PURGE QECOBOUT,TEMP
     FILE QECOBOUT;REC-256;DISC-!S;TEMP
ENDIF
RUN PROTOS.PROTOS.PROTOS;LIB-G
IF QEDITOUT - 1 THEN
     PURGE !C
     RENAME QECOBOUT,!C,TEMP
     SAVE !C
ENDIF
RESET SEMDOPE
RESET SEMPASS
RESET SSERR
RESET SEMTEMP
RESET ATNIN1
RESET ATNOUT2
RESET COBOLOUT
RESET ROOTDB
**
```

In this example, there are two JCW's used to pass information to the PROTOS program: PROBUILDWRITE and QEDITOUT. The QEDITOUT JCW gets its value from the Q parameter of the UDC. Based on the QEDITOUT JCW, different parts of the UDC are executed.

JCW's may only be assigned numeric values. So how do you make use of character string information from a UDC parameter? Here is one way:

```
CopyW2ToTape CharSet-X
SetJCW A-0
SetJCW E-0
SetJCW !CharSet-1
If A-0 and E-0 Then
  Display "An 'A' or an 'E' must follow the copy command.<ctrl>G"
Else
  File W2TAPE;Dev-TAPE;Rec--276,25,F,ASCII
  If A-1 Then
    FCopy From-PR997TAP;To-*W2TAPE
  Else
    FCopy From-PR997TAP;To-*W2TAPE;EBCDICOUT
  EndIf
  ListF PR997TAP,1
  Display "<esc>dJ Check the number of records copied.<ctrl>G"
EndIf
**
```

The purpose of this UDC is to copy a disc file to magnetic tape. The catch is that we need the option of copying it in EBCDIC format; thus, the CHARSET parameter. There are two "correct" responses: A and E. At the beginning of the UDC, two JCW's (A and E) are created and set to zero, then the user's choice is set to one. By having a default value that is not correct, if the user does not provide a value, the UDC will provide the message to the user and "remind" them of the correct values.

5.0 Closing Thoughts -- Is it worth the effort of learning something new?

We have looked at User Defined Commands and Job Control Words. Numerous examples have been explored to see how they work and how they can be used. They are a very powerful feature of the HP 3000. There is overhead associated with them (especially UDC's), yet, my feeling is that the convenience and "user-friendliness" gained outweighs that overhead. They must be controlled however -- people can get carried away when creating UDC's.

So, is it worth the effort? My answer is a (qualified) resounding YES! The qualification is that UDC's and JCW's must be carefully planned and monitored to reap the greatest benefit, but, oh, what a benefit it is: increased operator, programmer, and user productivity and a computer system that is easier to use overall.

Acknowledgements

A special thanks go to the following people who assisted in creating this paper in some (tremendous) way -- all of their efforts (and tolerance!) are greatly appreciated:

Dan Hinds
Jonathan Largent
Lois Largent
Karen Morgan
Dawn Thomas
Martha Walsh

Bibliography

Fisher, Sharon	"Setting up UCDs", Interact, July, 1984, page 48ff.
Griffin, Brad	"Another Way to SetJCWs as Part of a Log-on UDC", SuperGroup Association Magazine, June, 1986, page 12.
Hewlett Packard Company	Hewlett Packard Response Center Questions & Answers, August 1, 1986, page 1.
Hewlett Packard Company	MPE V Commands Reference Manual, First Edition, Update 1, 1986, Chapters 2 and 3.
Hewlett Packard Company	MPE V Intrinsics Reference Manual, Second Edition, Update 1, 1986, Chapters 2 and 5.
Hewlett Packard Company	System Operation and Resource Management Reference Manual, Second Edition, Update 1, 1986, Chapter 6.
Hewlett Packard Company Response Center (specifically B.A.)	Calls to the Atlanta HP Response Center during April, 1988.
Kabay, M.E.	"Making the Most of Your WELCOME Message", The Chronicle, March, 1988, page 54.
Largent, David L.	"Function Key Labelling", Interact, February, 1988, page 15.
Lund, Robert A.	"UDCs: A Primer", Interact, April, 1987, page 71ff.
Parker, Michael J. and Wilson, Lynn	"Labelling F-keys", Interact, November, 1987, page 22ff.
PROTOS Software Company	PROTOS Documentation, version 861231.
Shoemaker, Victoria	"UDCs: Marvelous and Misunderstood", The Chronicle, March, 1988, page 34.
Volokh, Eugene	"Burn Before Reading - HP3000 Security and You", Thoughts and Discourses on HP3000 Software, Third Edition, 1987, page 19ff.
Volokh, Eugene	"Conditional Execution -:IF, :ELSE, :ENDIF, ET. AL.", SuperGroup Association Magazine, February, 1986, page 28ff.

ADDING MULTI-PLANT FEATURES TO

A LARGE, INTEGRATED MANUFACTURING PACKAGE

by Terry H. Floyd and Mehrdad Laghaeian

BLANKET RESOURCES

Houston, TX 77450

Whether buying a new software package or maintaining an already purchased system, program and database modifications to a third party vendor's application are a serious decision. Manufacturing software systems like Hewlett-Packard's MM/PM or ASK's MANMAN are large and complex, trying to be all things to all customers (almost). Even if a company buys software that "fits" its applications, the complexities of the changing environment require some small changes within the vendor's lead time. But really complicated modification projects are expensive to maintain as new enhanced "releases" become available.

If the vendor is definitely not going in a given customer's direction, that company must either modify the package and abandon new releases, change to another package or prepare to spend substantial funds and resources for on-going maintenance of the modifications within the vendor's new releases.

If planned modifications are something the vendor intends to do themselves (someday), then it may be okay to do drastic changes. The key to successful modification in anticipation of the vendor's moves is a very close contact with the vendor and its users groups.

This presentation is the story of one company's modifications to ASK's MANMAN system. It covers adding "Multi-Plant" features which were two releases ahead of the vendor, but which hopefully will "go away" when the vendor implements its more generic solution in a couple of years. It also shows how a two or three man team can leverage the thousand man years work of the vendor's R&D department, moving fast to solve unique problems while the vendor and its many customers agree on the best solution for all.

ASK released MANMAN/MFG on the HP3000 in the summer of 1978. MANMAN was originally written in IMAGE/Fortran on the HP2100/1000. During 1979, as the Financial software packages (named FINMAN I,II,and III)were produced, four updates to MANMAN/MFG were sent out to the tiny customer base. Anyone who had modified any programs found out quickly how to migrate through new releases. If ASK had kept up the three month release cycle, who knows where we'd be? But they didn't; they went to a twelve month release cycle by 1982.

Cameron Iron Works of Houston, Texas purchased the ASK MANMAN software in June, 1987. The Company was replacing large IBM mainframes running custom in-house manufacturing/financial software. Three hundred plus terminals in sites all over the oil producing states were to be replaced with "dumb PC's" networked to an HP3000.

ASK's software/hardware price and performance/fit were excellent for Cameron in most areas. Two major concerns were lack of adequate Multi-Plant features in the applications software and doubts about response time with 300 terminals. Cameron cut the number of terminals in half to solve the latter problem. Blanket Resources, also of Houston, was contracted to enhance the applications software.

Cameron and Blanket Resources worked closely with ASK, at the local Houston office and at Corporate, to learn which features would be added to the packages and when. Visits by Cameron and/or Blanket Resources to the ASK Users Group "Multi-Plant Common Interest Group" meeting in St. Louis in December, 1987 and "Multi-Plant Roadshow Seminar" in Chicago in September,1987 gave insights about what other ASK users wanted and when ASK would respond. Sessions at the MANMAN conference in San Jose in March, 1988 brought the Multi-Plant CIG up to date on the progress at Cameron.

Cameron Iron Works "went live" on all major aspects of MANMAN MFG, OMAR, A/P, and G/L on March 1, 1988. Here is a description of the environment and Modifications:

A 'multi-plant' environment can be described as a group of manufacturing plants with a common 'thread'. Any attempt to be more specific about the meaning of the term would make it invalid for any number of 'multi-plant' situations currently operating in the business world. The 'common thread' could be a holding company (fixed assets), a managing entity (general ledger), a financial group (accounts payable, accounts receivable), a production management center (order management, purchasing, production planning), or a supplier (a plant and multiple distribution warehouses). It will take an enormous effort to satisfy every combination that could possibly exist.

The standard ASK package has been able to provide a common view of the financial state of a company with multiple plants through a single GL. The problem has been the fact that different plants had to lead separate lives. There was no means of centralizing any other function within the company. This created book keeping problems and made production planning and management very difficult. The material transfer between different plants had to be managed in a roundabout way. Centralized order management and production planning had to be done manually. It was not possible to order the same part from the same vendor for different plants and take advantage of quantity discounts. And the list went on and on.

The task to turn the ASK's manufacturing and financial packages into a Multiplant package was made possible due to the fact that ASK is getting ready to have multi-plant capabilities. There are already 'inactivated hooks' throughout the system which just needed to be switched on. It also helped to have a specific company which had an already working definition of what the multi-plant system should look like. This limited definition enabled the project team to spend minimum amount of time studying the design requirements. The program modification and development started within 30 days after the start of the project.

The overriding guideline for the entire project was to stay within the realm of future ASK multi-plant features defined by ASK's own project documentation and the undocumented program hooks buried inside the seemingly 'single plant' packages. All designs had to fit within the framework of ASK's upcoming product enhancements so that the custom modifications could be phased out as they become vendor supported functions.

The modified system can have a single or multiple financial groups managing multiple manufacturing plants. Material transfer is possible and is tracked automatically through intercompany clearing accounts. Order entry is possible at the local plant level and also through centralized order management. Vendors are maintained in one database and paid for purchases by different plants with a single check. An overall view of bookings and shipments is possible making it feasible to observe product trends and forecast accordingly. All this was possible without any changes to the transactions and utilities supplied by the vendor.

Under this scheme there are plants that manufacture and warehouses under these plants which act as distribution points. As far as the user is concerned, the visible differences from the unmodified ASK package are as follows:

A. A multi warehouse user is prompted for the warehouse number at the start of each command.

B. A multi-plant user is prompted for the plant when initially starting a session.

A single warehouse user will never be prompted for the above information. The only other difference is the interplant material transfer transactions which had to be developed in their entirety since there are no such functions currently offered by the software vendor. All the warehouse and plant switching is done by the software in the background freeing the user from determining which database they should be accessing.

The project was completed in shortest possible time due in part to the somewhat unique approach which enabled the users to interact with the designer/programmer directly. The completed programs were submitted to the requesting user for testing. Any bugs or enhancements were explained to the original programmer who also made the necessary changes. There was one project leader for design and implementation who was involved in all aspects of the decision making. There was also a project coordinator in charge of user/design team/operations team interface. There were at most seven individuals in the project team including the user testing group.

Originally it was decided that the documentation would be completed before any code modification was started, but it quickly became apparent that, since the users had the final word, any attempt at documentation would lead to hundreds of pages of obsolete information by the completion of the project. The users, not being familiar with the new package, would alter the specification upon the first execution of a modified or developed transaction. This method goes against the prevailing and accepted method of product development and modification. In practice, however, this approach was abandoned after a 40 page document had to be discarded before it was even submitted to the client company. The time required to produce the documentation was deemed better spent completing the requested program and submitting that to the user committee.

The Cameron/ASK/Blanket Resources MANMAN implementation is still in progress while this paper is being written in April, 1988. The speech to be delivered in Orlando in August will provide more details.

OUTLINE

I. Introduction

A. History of ASK's MANMAN Software Releases

B. History of Case Study Company

II. Fitting the Company to the Package - the Compromises

A. Multi-Plant Environment

B. The Standard ASK Package in Multi-Plant Environment

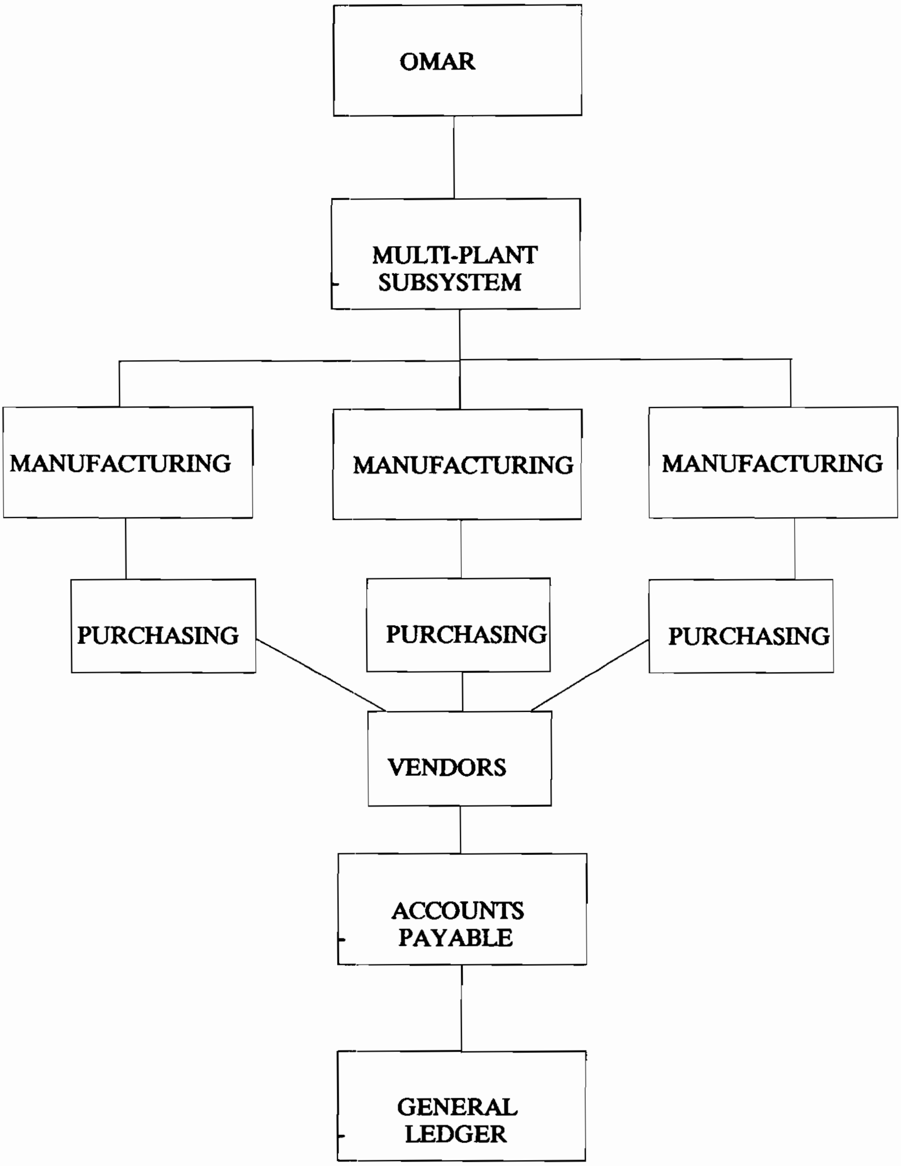
C. The Modifications

D. Implementation Events

III. Conclusions

A. The Team Approach for Success

B. Eleventh Hour Delivery Service


OMAR
MULTI-PLANT SUBSYSTEM
MANUFACTURING
MANUFACTURING
MANUFACTURING
PURCHASING
PURCHASING
PURCHASING
VENDORS
ACCOUNTS PAYABLE
GENERAL LEDGER

HP Portability: RAM/ROM vs. Disk-Based Approach

Hal Goldstein
Personalized Software
Fairfield, Iowa 52556

I am the publisher and editor of The Portable Paper, a bi-monthly magazine devoted exclusively to Hewlett-Packard (HP) Portable computers. We review products that run on the Portables and provide in-depth tutorials, tips, and tricks on how to make best use of HP Portables. Our company, Personalized Software also sells many software and hardware products that enhance HP Portables.

We know from our extensive customer service how people use and feel about HP Portables. We are intimately familiar with the machines' strengths and weaknesses. From this vantage point, we will compare the two main versions of computer "portability" in the marketplace today: the RAM/ROM disk-based portable and the mechanical disk drive portable. In particular, we will compare the Portable Plus/HP110 with the new HP Portable Vectra.

First we'll look at how HP RAM/ROM portables work, at their advantages and disadvantages, and recent advancements. Then we'll look at how the mechanical disk drive portables work and their advantages and disadvantages.

RAM/ROM MACHINES

To understand what we mean by a RAM/ROM electronic disk-based portable, we will examine the Hewlett-Packard Portable Plus and the HP110. Both these machines contain two electronic "pseudo" disks: a RAM-based A drive disk and a ROM-based B drive disk. These two disks are not physical disks in the usual sense -- they are electronic simulations of disks. From both the user's and the the operating system's point of view these RAM and ROM disks look the same as a floppy or hard disk.

The RAM-based A drive disk stores data and programs that can be created, modified, or deleted, in the same way one would manipulate data on a floppy or hard disk. Similary, the ROM-based B drive disk functions like a write-protected disk. A user can access any data or program on the ROM disk. However, no information cannot be added to, deleted from, or modified on the ROM disk.

Computer users may be familiar with "RAM disks." Usually, these can be created by software and function as a disk as long as the computer is on. In contrast the HP Portable electronic disks are "non-volatile" RAM. Unlike volatile RAM, the electronic disk stores and maintains files and programs even after the computer is "turned off." The internal battery supplies enough power to keep these files and programs alive for weeks and even months.

Further, this non-volatile RAM in the HP110 and Portable Plus actually serves two functions. The user divides his RAM between internal memory and an electronic disk according to his requirments. For example, on the 512K Plus, a user can allot 128K to internal memory and 384K to the electronic disk drive. That means he has 128K for program and temporary data space. When he wants to save his work, he saves it to the permanent 384K electronic A disk space. The B disk stores ROM software programs or data and is never used directly to save files.

We will now examine the HP110 and Portable Plus more closely.

THE HP110

The first HP RAM/ROM portable, the HP110, emerged from one simple idea: to create a lightweight, powerful, rugged portable computer that meets 95 percent of a portable computer user's needs.

Built into the ROM B disk of the HP110 is the full implementation of Lotus 1A, an easy-to-use word-processor called MemoMaker, communications software for the 300 baud modem, and MS-DOS 2.11. It also has 272K bytes of non-volatile RAM for electronic disk storage and internal memory.

The default and most common division of that 272K of RAM between internal memory and electronic disk is 96K internal memory and 176K RAM disk. With this configuration an HP110 can hold, for example, on its electronic disk: Turbo Pascal, a text formatting program, 110K for data files -- and still have enough internal memory to run many Lotus applications.

HP no longer makes this 16-line HP110. (However, Personalized Software and other sources sell used HP110's for between $700 and $1000, the bargain of the year.)

The HP110 does have its limitations. Some users want a bigger screen, larger memory, and IBM compatibility. They also want a choice of ROM applications and a higher-speed modem. To satisfy these desires, Hewlett-Packard created the Portable Plus.

THE PORTABLE PLUS

The Portable Plus' 25-line screen, greater memory capacity, user configurable ROM drawer, and 300/1200 baud modem make it more powerful than the HP110. It is more IBM compatible than the HP110, although it is not a true compatible.

The nine-pound Portable Plus comes with a standard 512K of internal memory. Until recently, the memory could only be expanded by 384K of RAM, but now it can be expanded to as much as

4.5 Megabytes (4500K). The standard Plus sells for $2195 (or $2700 with an internal 1200 baud modem).

Users may then purchase one or two additional drawers, depending on their own requirements. One drawer, usually reserved for ROM software, costs $160 plus the price of the ROM software (e.g., Lotus, $495). There are twelve sockets in the ROM drawer -- room for 8 to 15 applications. The other drawer is usually reserved for additional RAM. This flexibility permits users to build their own system by purchasing ROM chips and disk-based software for the electronic RAM disk.

Recent developments have not just enhanced the Portable Plus, they have made it a brand new machine. The $995 HP 1 megabyte RAM drawer as well as the Personalized Software/SoftWord 2 megabyte drawer and 1 megabyte RAM/8-socket ROM drawer greatly expand the computer's memory and capability.

These larger RAM drawers allow full-blown RAM software packages such as WordPerfect (word processor), Condor 3 (relational data base manager), and T/Master (spreadsheet, word-processor, communications, graphics, database), to fit in the Plus' RAM.

RAM/ROM ADVANTAGES

What are the advantages to RAM/ROM disk-based portables? First of all, they are rugged. Since there are no moving parts other than the keys, the portables can be dropped without damaging the unit.

Second, the electronic loading makes input and output fast. For example, it takes only a few seconds to load Lotus from ROM. Third, with built-in disks, users don't have to carry disks on the road or worry about copy protection (e.g., Lotus). Greater memory capacity means users can pack more in less space at lower cost.

Perhaps most importantly, the briefcase size and nine-pound weight make these computers truly portable.

ROM software comes in small chips that can be easily inserted into the Portable Plus. RAM software comes on 3 1/2" or 5 1/4" disks that can be run or transferred onto the electronic RAM disk from an external HP9114 disk drive or an IBM PC.

RAM/ROM DISADVANTAGES

One disadvantage to RAM/ROM portables is the separate disk drive required for loading, storing, and backing up files and for making room on the electronic disk. The external disk drive, which makes use of the HP-IL interface protocol, provides slow access.

The Portable Plus is not a true IBM compatible and, even with recent price reductions, is still fairly expensive. However, it certainly should be possible to create an IBM compatible RAM/ROM machine. As memory prices fall, it should be possible to create a less expensive machine.

MECHANICAL DISK DRIVE BASED COMPUTERS

Unlike a RAM/ROM disk drive, a mechanical disk drive magnetically stores information on a flexible disk (or some other medium). While a motor spins the disk, a delicate, electronic device moves in very close to read the disk and write on it. These moving parts make the computer susceptible to damage if dropped. They also significantly increase the computer's weight.

ENTER THE PORTABLE VECTRA

The new Vectra CS computers (the dual floppy Portable Vectra CS and the hard disk Portable Vectra CS Model 20) function as both portable and desktop computers. And they are true IBM compatible computers!

The 17.6-pound, briefcase-size Vectra offers four internal I/O expansion slots for adding expanded memory, serial ports, a 2400-baud modem, and other devices, without increasing the computer's size. The Portable Vectra comes standard with an expansion card that takes up one of the I/O slots and that contains a parallel centronix port and a monitor interface. That means the Portable Vectra can be easily connected to a printer and a monochrome, CGA, or EGA color monitor.

Other features include: a large, 12-inch, liquid crystal display based on new "supertwist" technology; a full-size keypad with 12 function keys; a 1.44-Megabyte (1440K), 3 1/2 " flexible disk drive capacity; and a standard 640 Kbytes of user memory.

Users can also add up to six Megabytes of EMS RAM to the dual floppy Vectra and up to four Megabytes to the 20 Megabyte hard disk Portable Vectra Model using the I/O expansion slots. The NEC V30 CPU speed is twice as fast as the HP110, the Portable Plus, or the IBM XT. For certain applications it runs as fast or faster than an IBM AT.

The basic Vectra, with its dual-floppy system costs $2495. The Vectra Model 20, with its 20 Megabyte system, is a little heavier than the basic unit and costs $3595.

MECHANICAL DISK DRIVE DISADVANTAGES

Despite a clamshell, closed-case design and extra shock mounting for its mechanical disk drive, the Vectra is not as sturdy as a RAM/ROM computer and its electronic disk drive.

It is less convenient than the HP110 and Portable Plus; it is twice as heavy, requires disks, and, although its central processor (CPU) is faster, its input and output is much slower.

SUMMARY COMPARISON CHART OF TWO APPROACHES

The first section of the following chart compares the RAM/ROM vs. mechanical disk drive approach to portable computing given the state of today's technology and costs. The second section compares the Portable Plus to the Portable Vectra in particular. The differences in the second section reflect design decisions rather than characteristics inherent in RAM/ROM or mechanical disk drive machines.

Characteristic	Electronic Disk Drives	Mechanical Disk Drives
Ruggedness	More	Less
I/O (Save/Retrieve)	Faster	Slower
Weight	Lighter	Heavier
Run off battery	Longer	Shorter
Backup	Harder	Easier
Storage capacity	Less	More
Cost	More	Less
Self-containment:		
Short trips	More	Less
Long trips	Less	More

Portable Plus Vs. Portable Vectra:

Keyboard	Inferior	Superior
Screen	Smaller	Larger
IBM Compatibility	Some	Complete
Expandability	2 RAM or ROM drawers	4 Adapter slots
CPU	Slower	Faster
Internal memory	512K Max	4 Meg with EMS'g

CONCLUSION

The marketplace has not yet caught on to the true convenience and portability of RAM/ROM-based computers. Rather, most users still view the ideal portable as a scaled-down version of their desktop machines, mechanical disk drives and all. The more a portable resembles a desktop, the better the portable.

Given this criteria, the new HP Portable Vectra succeeds admirably. In fact, its ability to hold expansion cards, its full keyboard, and its 1.44 Megabyte floppy capacity make it unique. If an external monitor is attached to the Portable Vectra, the Portable Vectra is virtually indistinguishable from a desktop computer. When portability is required, it takes only seconds to detach the monitor and snap on the LCD screen.

However, for people who travel a lot and really need portability, RAM/ROM machines are the way to go. Mechanical disk drive-based portables can never match the light weight, ruggedness, and long battery life of RAM/ROM machines. Hopefully, HP will take the best of the two technologies to produce an IBM Compatible RAM/ROM computer with detachable mechanical disk drives.

With its Portable Plus and Portable Vectra, HP has created two exceptional machines. Unfortunately, although HP has invested millions of dollars in research and development funds to create these computers, it has not been willing to spend the money to educate the marketplace as to the practical value of the two machines.

Both computers look different from the "typical" portable in the market. Consequently, neither of the computers has been especially well-received in the computer press or at the retail computer dealer level.

Many traveling professionals will find that a 1.5 Megabyte Portable Plus with Lotus, a word processor, spelling checker, and communications package all in ROM is the most useful, practical powerful portable computer in the marketplace. On the other hand the Portable Vectra with an external monitor is both a full-featured desktop and a portable computer. It is ideal for the executive who wants a powerful, ergonomic, full-featured, compact IBM compatible computer for his office that he can easily take home at night.

My strong hope is that HP will not throw away its leadership role in the Portable industry by abandoning its ROM and RAM disk-based technology. Whether or not HP stays in this RAM/ROM arena, you can be sure that, eventually, other Portable manufacturers will pick up the idea and produce IBM PC-compatible RAM- and ROM-based machines. It is just too good of an approach to Portable computing not to catch on in the general marketplace.

The Seven Wonders of TERMDSM

Dennis Heidner
Boeing Aerospace

Abstract

TERMDSM (*Terminal On-line Diagnostic/Support Monitor*) is an HP-written utility program, distributed as part of the Fundamental Operating System (FOS) and commonly used to fix "broken" ports on the HP3000. Most system managers never fully realize the potential of this wondrous program which understands the internal system tables (PPDIT, HWDIT, etc). This paper documents more fully the workings of TERMDSM, and how the system manager (or data communications manager) may use it more effectively.

The paper emphasizes the DISPLAY and DUMP commands and explains the various system tables which may be viewed through their use. (These tables are not explained in either the TERMDSM manual or the System Tables manual.) This paper is applicable only to MPE V.

Introduction

TERMDSM is available in the PUB group of the SYS account. TERMDSM was written to provide SEs (and later the system manager) a means of correcting port lock ups without needing a shutdown and restart. This program provides simple commands which allows the user to determine the hardware status of a port, the internal MPE software status of the port, and how the port is really configured (default port or a customized port).

Why study TERMDSM? The best reason is that it is in your own self interest! The official view from HP is expressed on two consecutive pages in the TERMDSM manual. First: *Generally, the system manager has many troubleshooting responsibilities and is required to collect and analyze a great deal of information. It is often helpful to keep a journal that documents information about past and present problems. But most importantly, the system manager must know when to call an HP service representative. When an HP service representative is called, he or she should be provided with all the pertinent information that could aid in the problem resolution process.*[1] This does not necessarily mean that HP would like all of the system managers to be handy with a wire cutters, hammer and screw driver. In fact HP is very quick to point out that: *It is Hewlett-Packard's responsibility to resolve problems that arise from the use of products under the warranty of a customer service contract. When support personnel are presented with complete information, the problem resolution process is made easier for all concerned.* [2]

The intent of this paper is to provide only a general overview of the tables used by MPE as information moves to and from users' terminals. **The contents of this paper should not be used to design privileged mode programs which access or modify the tables!**

MPE Data Communication Principles

The connection between the real world of the user and the HP3000 Central Processing Unit (CPU) is typically through either an Asynchronous Data Communications Controller (ADCC), an Advanced Terminal Processor (ATP), or a Terminal Interrupt Controller (TIC - the Micro HP3000 equivalent of an ATP). These devices are the port controllers (with many ports to each card). The ATP-type port controller is capable of "running" independently of the main CPU, transferring the contents of its buffers to and from the user's terminal. A special control program sent from the CPU to each port controller contains a complete description of the next task the controller is to perform. This task could be as simple as "remain idle - waiting for a key to be depressed" or as complicated as detecting a BREAK or modem disconnect. Unlike the ATP, ADCCs are not "intelligent" and require the intervention of the main CPU for EVERY character transferred to and from the terminals! ADCCs do not use a control program but instead a "channel" program. Because of the performance limitations of the ADCCs, HP has been gently nudging the user community to migrate away from ADCCs to the newer ATPs.

The control program which is sent to the port controller is "compiled" by the CPU after referencing a number of special "DEVICE INFORMATION TABLES" or DITs. These low level DITs are called the Hardware DIT (HWDIT), Protocol and Data Manager DIT (PDDIT), and Port Protocol DIT (PPDIT). In addition to these special tables MPE also maintains Terminal Buffers (TBUFs), VFC buffers, Interrupt Linkage Tables, and Logical Device Tables. Confused? Lost? Rightfully so! Deciphering what does what and who talks to whom while doing terminal I/O is much like the problem that a blind-folded tourist encounters when he is "dumped" into a foreign country and cannot speak the local language!

Let me clarify how terminal I/O is performed by personifying the tables and processes used. Let's set the scene by saying we have been called to solve a hideous crime. The inspector general (also known as the system manager) has observed that agent 99 (also known as LDEV 99) has been HUNG! The inspector is fit to be tied. The last several weeks some mysterious criminal element has been wreaking havoc. The whole country's (system's) network of agents (LDEVs) is concerned that they will be next. You are presented with a thick dossier of the agents and suspects (the tables). With the assistance of a new crime-stoppers' tool (TERMDSM) you must identify the most likely suspects so they can be then tracked down by INTERPOL (your local HP CE & SE). ***(In order to make the paper more readable and entertaining I deliberately blur the distinction between processes and tables!)***

The first entry on the inspector's list is a suspect called the Terminal Data Segment alias TDS. TDS is quite nosy; he has maintained a record of the other suspects' (DITs') locations and movements. It also appears that TDS may have a split personality!

The second suspect is called the Logical Monitor DIT alias MONDIT. Observation of the MONDIT indicates that he gets a lot of action. MONDIT appears to represent the "go-between". He has been edgy lately (perhaps due to the lack of sleep). MONDIT has been seen associating with IOQ, HWDIT, PPDIT, etc. The inspector does not believe that MONDIT is the culprit although we might be able to squeeze him for some information.

The third suspect is Protocol and Data Manager DIT alias PDDIT. PDDIT is an ambassador, he likes strict adherence to protocol. PDDIT will often take documents passed to him and add additional information to them. PDDIT is very demanding and complains that the others never worry about time.

Our fourth suspect is called Physical Driver-Hardware Device Information Table alias HWDIT. HWDIT is our mechanical expert, he spends most of his time working with something called "hardware". The other DITs view him as a social introvert! His only close friends are CNTRLPROG and CHANPROG (suspects five and six). CNTRLPROG - Control Program - is a quick and adaptable talker. CHANPROG - Channel Program - has a much more limited vocabulary and requires constant supervision.

Suspect number seven's name is Terminal Buffer alias TBUF. TBUF has what appears to be a photographic memory, unless she is overloaded with work. TBUF likes to be asked to remember bits and pieces of data. TBUF maintains a running total of all the requests for work, the fact TBUF has always been handy, and that she has a number of relatives waiting to help.

Our eighth suspect is quite crafty. Most of the other DITs are convinced that he has never done anything useful in his life. He appears to take orders from something called "ATTACHIO" and after asking for advice from the other DITs he passes the work on to some other worker. (Sounds like a manager to me!) This suspect uses the name of Input/Output Queue; IOQ is the nickname he prefers. The inspector has observed that shortly after IOQ gets into the act, the process which MONDIT belongs to is often awakened!

Next up we have fraternal quadruplets. Although related, these suspects have unique personalities. They are the Interrupt Linkage Tables (ILT & ILTX), Driver Linkage Table (DLT), and Device Reference Table (DRT). The inspector is sure that these tables are loyal and not the cause. The inspector's analysis is based on the knowledge that if any one of these suspects were compromised, the country would have already fallen (system failure)!

The next two suspects are the Logical-to-Physical Device Table (LPDT) and the Logical Device Table (LDT). The LPDT is the psychoanalyst in the gang. LPDT always knows whether the agents (LDEVs) are duplicative, or can accept jobs or data. The LPDT can immediately tell whether a port is feeling good (UP) or blue (DOWN). The LDT is a certified public accountant. LDT feels that items such as record length, printer header & trailers, and access count are not given enough mention.

The two suspects who appear to have most at stake are the Process Control Block twins, aliases PCB and LDTPCB (process control block associated with LDT). These two tables appear to be the origin of all input/output on the system. When an agent working for them hangs, they hang! Conversely if something happens to the PCB then the agent is left waiting and wondering when more data will come.

The second-to-last suspect's name is Vertical Format Control alias the VFC table. This suspect only visits the scene of port hangs when the port was configured as a spooled device. It is unlikely that a bad VFC table will cause a hung agent (LDEV); however, bad VFCs are known to cause erratic and strange evidence (e.g., top-of-form when it was not called for.)

The last suspect, Port Protocol DIT alias PPDIT, often looks and acts very suspicious before and after agent hangings (LDEV problems). The inspector has noted that the PPDIT appears to be wishy-washy, constantly changing his story when a different user is working with our agents. Rumor has it that PPDIT has undergone a "face lift" from something called a "Workstation Configurator".

Before we begin our actual investigation of the crime scene, some additional background information is necessary. Virtually all of the forenamed suspects are involved with I/O between the CPU and the agent (LDEV). The suspects have been greatly influenced by something called the "SYSTEM CONFIGURATION". Each suspect has been read its rights by another computer program called the "Initiator". Perhaps you've seen the Initiator repeat the rights to all the tables, it looks something like:

```
          DIRECTORY MAINTENANCE COMPLETED
          PART 1 OF 6 COMPLETED - MEMORY RESIDENT TABLES SET UP
          PART 2 OF 6 COMPLETED - SL BINDING
          PART 3 OF 6 COMPLETED - SYSTEM I/O PROCESS CREATION
          PART 4 OF 6 COMPLETED - DRIVER LOADING
          PART 5 OF 6 COMPLETED - DISC RESIDENT TABLES SET UP
          PART 6 OF 6 COMPLETED - SYSTEM PROCESS CREATION
```

After the Initiator has completed, each table will retain the basic characteristics specified in the system configuration until one of two events happens: the port is allocated, or an FCONTROL is issued requesting changes in the port characteristics. Allocation means that either a user has entered a [RETURN] to logon or the port has been FOPENed. **Once a port's characteristics have changed they will remain in this new state until either the port is de-allocated or another FCONTROL is issued.** So what? Well, it is possible to write a very simple "user mode" program that alters the port's characteristics so that it will work incorrectly for the next program or make the port appear to be hung. (I know. I have accidentally done so.) TERMDSM allows you to determine very quickly the characteristics of any port.

Troubleshooting Tips

There are three basic items that are required for fast and effective troubleshooting. The first is knowledge. Fortunately for the new system manager, there have been experimenters who have preceded you. The work of Ross Scroggs is still a classic and should be required reading. In order to troubleshoot, you must be able to distinguish what appears to be normal (or correct) from what is wrong. Take the time to read the reference material I have listed. Make sure that you keep good notes when you encounter problems, including what the causes, effects and cures were. The second basic "thing" that you need are tools. This does not mean that you must spend several thousand dollars for data communication analyzers. First learn to use effectively the free tools that exist on your system. These tools range from the very basic :SHOWJOB, :SHOWOUT, and :SHOWDEV commands to the more sophisticated TERMDSM program. The third basic item is a troubleshooting methodology. By this I mean knowing where to start looking and the sequence of problem areas to check.

HP's *Fundamental Data Communications Handbook* contains an entire chapter devoted to troubleshooting. You should become familar with it. But even before starting the steps HP lists, I do the following:

1 - If the port is a terminal then :SHOWJOB. Look for an asterisk ("*") in the state column; it means that MPE has not yet completed the logon sequence for the session. The JOBNUM and LDEV (JIN/JLIST) will be required for later tracking.

```
:SHOWJOB

JOBNUM  STATE IPRI JIN  JLIST    INTRODUCED  JOB NAME

#S154   EXEC        20  20       MON  4:11P  OPERATOR.SYS
#J971   EXEC QUIET  10S LP       WED  5:51P  REV03I,IDLE.SYS
#S646   EXEC        46  46       SAT 11:52A  DENNIS,TEM.TEIMS

3 JOBS:
    0 INTRO
    0 WAIT; INCL 0 DEFERRED
    3 EXEC; INCL 2 SESSIONS
    0 SUSP
JOBFENCE= 0; JLIMIT= 5; SLIMIT= 30
```

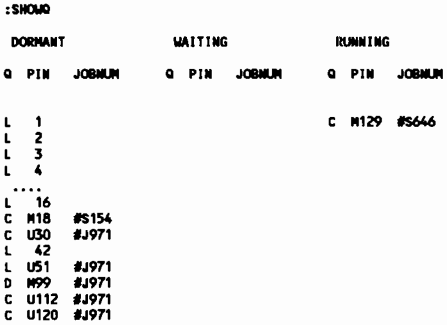
If the port is a spooled device (plotter or printer), use the :SHOWOUT command to check the status of the spooler. If the spooler shows it is active but there is nothing being printed, check for messages on the operator's console. Make a mental note of the STATE. (Remember, only READY files print). Also make sure that the priority of your output exceeds the OUTFENCE.

```
:SHOWOUT

DEV/CL   DFID     JOBNUM  FNAME    STATE FRM SPACE RANK PRI #C
LP       #02671   #J959   LP       READY       248    D 3   1
LP       #02989   #J1064  $STDLIST READY       104    D 2   1
20       #0648    #S154   $STDLIST OPENED
..        ...      ...      ...     ...
46       #03017   #S646   $STDLIST OPENED

29 FILES:
    0 ACTIVE
    26 READY; INCLUDING 26 SPOOFLES, 26 DEFERRED
    3 OPENED; INCLUDING 1 SPOOFLES
    0 LOCKED; INCLUDING 0 SPOOFLES
    27 SPOOFLES:  3888 SECTORS
OUTFENCE = 3
```

2 - If you have passed step 1, then have the operator use the :SHOWDEV command. Check to see if the port is available ("AVAIL") and job accepting ("A"). If they are not, find out why and take corrective action. A printer can become "owned" by a user's program if the system spooler process for the device was stopped. Once a printer is privately owned, other users' requests for it will be denied with a file system error 55 - "device unavailable". If the port is privately owned, the pin number of the owner will be displayed in the "OWNERSHIP" column. Jot down the offending pin and look it up with the :SHOWQ command. A port may also be "DOWNed" by the operator or by the security monitor. One other interesting item to look for is the number of files open at each LDEV. An interactive session waiting at the ":" prompt will only have two files open. If you see more, it means that the user is running a program.

```
:SHOWDEV
LDEV  AVAIL           OWNERSHIP          VOLID         DEN    ASSOCIATION

   1  DISC            6 FILES
   2  DISC  (RPS)     22 FILES
   6  SPOOLED         SPOOLER OUT
   7  UNAVAIL         SYS #1             LOGTAP(ANSI)
   8  AVAIL
   9  AVAIL
  10 A AVAIL
  20 A UNAVAIL        #S154: 2 FILES
  21 A AVAIL
  22  SPOOLED         SPOOLER OUT
  23 A AVAIL
  .....
  39 A AVAIL
  40  SPOOLED         SPOOLER OUT
  41 A AVAIL
  42 A AVAIL
  43 A AVAIL
  44 A AVAIL
  45  SPOOLED         SPOOLER OUT
  46 A UNAVAIL        #S646: 2 FILES
```

3 - Use the :SHOWQ command to check if the process is able to run or to obtain the Job/Session number of a pin that is holding a printer. Pin numbers which are preceded with "M" are typically waiting at the ":" prompt. Pin numbers which are preceded with a "U" are application programs. System processes (like spoolers) will not have any letter in front of them.



```
:SHOWQ

 DORMANT             WAITING             RUNNING

Q  PIN   JOBNUM      Q  PIN   JOBNUM     Q  PIN   JOBNUM


L   1                                    C  M129  #S646
L   2
L   3
L   4
 ....
L   16
C  M18   #S154
C  U30   #J971
L   42
L  U51   #J971
D  M99   #J971
C  U112  #J971
C  U120  #J971
```

After you have performed the easy and simple checks it may finally be time to use TERMDSM. This program has seven wonderful commands that we will cover in this paper. They are ABORTIO, ABORTJOB, BROKEN, DISPLAY, DUMP, RESET, and EXIT. (Not covered, but available, are diagnostics for ATP, ASNP, and DMI ports).

Running TERMDSM

TERMDSM may only be run by a user with SM or OP capability. Although the program is safe (and should not cause system failures), it does give the user the ability to abort I/O or jobs, and view terminal buffers. The latter should be of special interest to the security-conscious, since it allows all passwords to be viewed as they are typed in (even with the new security product installed).

The prompt for TERMDSM is "->". TERMDSM is accessed by entering

```
:RUN TERMDSM.PUB.SYS

HP32196G.05.04 - TERMDSM - Terminal Diagnostics (C) Hewlett-Packard
Co.

ADCC software version - G.51.24

Type HELP for aid
```

HELP

TERMDSM has built into it a help function. In addition to the "high level" help, each command also will provide additional help.

```
-> HELP
Valid input at this point is any one of the following:
  DIAGnostics - to enter dialog for running diagnostics on
                one or more ATP/ASNP ports
  DMIdiag     - to enter dialog for running diagnostics on
                one or more DMI ports
  ABORTJOB    - to enter dialog for aborting one or more
                jobs
  ABORTIO     - to enter dialog for aborting I/O pending on
                an ATP/ADCC/ASNP/DMI port
  RESET       - to enter dialog for resetting one or more
                ATP/ADCC/ASNP/DMI ports and associated tables
  DISplay     - to enter dialog for displaying ATP/ADCC/ASNP/DMI
                tables and terminal buffers
  DUmp        - to enter dialog for dumping ATP/ADCC/ASNP/DMI
                tables and terminal buffers
  Broken      - to obtain a list of ATP/ADCC/ASNP/DMI
                considered broken and/or unfixable
                by the driver software
  Exit        - to exit TERMDSM
The capital letters indicate abbreviated valid input.  The
message output when you enter HELP will change depending on
where you are in what dialog.

Anytime // is entered, TERMDSM will terminate.
```

BROKEN

The command BROKEN is used as a quick check for ports which have encountered some unusual condition which caused them to "break". Once you have identified a broken port you may use the TERMDSM RESET command to fix the port. However not all broken ports can be fixed. If the port is unfixable TERMDSM will indicate so. Unfixable ports can only be reset by shutting the system down and restarting with either a WARMSTART, COOLSTART or a COLDSTART. An example of the BROKEN command is:

```
-> BROKEN
BROKEN PORTS

       LDEV#   BROKEN   UNFIXABLE
       27        *
       33        *          *
```

DISPLAY

TERMDSM allows the user to display active information for virtually all of the tables used by the system for terminal I/O. In fact TERMDSM is the only HP-supported program which will allow the system manager to find out the terminal type and termtype file defined for a specific port. With this command you get only what you ask for. A sample dialogue with the DISPLAY command is:

```
-> DISPLAY
DISPLAY

Enter table name or ldev number:  HELP
Enter one of LPDT, LDT, TDS, MONDIT, HWDIT, IOQ,
DLT, ILT, DRT, PCB, LDTPCB, PDDIT (ATP/ADCC/DMI),
PPDIT (ATP/ADCC/DMI), VFC (ATP/ADCC/DMI), TBUF (ATP/ADCC/DMI),
CNTLPROG (ATP/ASNP/DMI), CHANPROG (ADCC only), an
ATP/ADCC/ASNP/DMI device LDEV number or just a carriage
return.

There are two sets of tables associated with each
device:  queues of tables and single tables.  IOQs and TBUFs
are queues of tables while the rest of the tables are individuals.
If IOQ or TBUF is entered, the first table in the queue will
be displayed and you will be prompted whether to continue
with the next.  If any other table name is entered, it will
be displayed.

If an ATP/ADCC/ASNP/DMI LDEV number is entered, it is now the
device whose tables are to be displayed, not the device
whose LDEV number was input previously.

If just a carriage return is input, TERMDSM returns to the
outer block.

Enter table name or ldev number: 46
Enter table name or ldev number: VFC

***** VFC Table SIR's *****

 OWNER'PCB      = 0       HEAD'PCB       = 0       TAIL'PCB       = 0
 QUEUE'LENGTH   = 0

***** VFC Table SIR's *****

WORD
0    000000 000000 000000 000000

No VFC entry associated with LDEV #46
```

A more complete explanation of the tables will be given later in the paper.

DUMP

The DUMP command will format "snapshots" of active systems taken by MPE immediately after a port breaks. Unlike the DISPLAY command DUMP can be used to RESET ports or format broken ports. Because it will reset both good and broken ports, *the DUMP command should be used with CAUTION!* For ATP-type controllers the user may request that the PCC memory also be dumped. The catch is that in the process of dumping the PCC memory, the port will also be reset! TERMDSM will ask whether or not you want the PCC memory included in the dump. If the port is already broken then you should say yes; if not and the port is in use, think twice! The sample tables that are explained later in the paper were dumped from an ADCC device that was active (ADCCs do not contain PCC memory). TERMDSM will dump and format the table information for the specified port to a file called TERMnnn, where nnn is the port logical device number. The file will contain carriage control characters for "top-of-form". It is best printed by using FCOPY. The dump command used to format the tables for the paper follows:

```
-> DUMP
DUMP
 Enter ldev number: HELP
Valid input is an ATP/ADCC/ASNP/DMI device LDEV number or just a
carriage return to terminate input.  As long as just the carriage
return is not entered, the prompt is repeated until all the
devices that are to be dumped have been input.

 Enter ldev number: 46
 Do you want to include a message? Y
   message-> MODEM CARRIER LOSS, BUT SESSION STILL ACTIVE
   message->
 Data dumped into file TERM46.
 Enter ldev number:
```

RESET

The RESET command will cause TERMDSM to abort sessions that were using the device, reset the hardware, and rebuild the internal tables related to the device. (Please note: If the device was unfixable, (see BROKEN) even RESET cannot fix it!) I have included a sample dialogue for fixing a typical port. Users who have the DMI interface should refer to HP's *Terminal Online Diagnostic/Support Monitor (TERMDSM)* reference manual.

```
-> RESET
RESET

Enter ldev number: HELP
Valid input is an ATP/ADCC/ASNP/DMI device LDEV number or just a
carriage  return  to  terminate  input.    As  long  as  a  valid
ATP/ADCC/ASNP/DMI device LDEV number is entered, the prompt will
be repeated until all the devices you want to reset are reset.

Enter ldev number: 46
The device entered is currently owned.  Resetting this device will
abort the session associated with it.  Be sure that you have the
correct logical device number!  If you wish to continue with the
reset process, respond with "Y", "N" or [RETURN]. y
```

```
The DEVICE DRIVER does not consider this device broken.  However
this does not exclude the possibility of a hung port.  If you wish
to continue with the reset process respond with "Y", "N" or
[RETURN] y
```

EXIT

When you are done and want to leave TERMDSM enter "EXIT" or "//".

```
-> EXIT
```

SAMPLE TABLE DUMP

Perhaps one of the largest drawback to using TERMDSM for our criminal investigation is simply information overload! If we format the tables for a specific port, somewhere between 15 and 25 related tables will be printed! For neophytes this is generally enough to cause immediate panic. Fortunately the information that we are really after is only a small subset. (This paper is not intended to teach you how to write device drivers, only how to troubleshoot common problems.) I have annotated the tables that you should be able to skip with the phrase "beyond the scope of this paper". For the rest of the dumps (or displays) we must play the part of the master detective Sherlock Holmes.

The Banner

The DUMP command will automatically include a banner at the beginning of the formatted file.

```
HP32196G.05.05 - TERMDSM - Terminal Diagnostics (C) Hewlett-Packard Co. 1983

MODEM CARRIER LOSS, BUT SESSION IS STILL ACTIVE!

DUMP OF LDEV#46 ON SAT, APR  9, 1988,  9:59 AM
```

Terminal Data Segment (TDS)

Terminal Data Segments contain the Hardware, Protocol, Data Manager DITs, the control (or channel) program and Vertical Format Control tables. MPE will automatically create the correct number of TDSs required. For systems which contain ATPs there will be one or two TDSs for each Device Reference Table (DRT) channel. Because ADCCs are much smaller, only one TDS is required for each DRT channel.

```
***** ADCC Terminal Data Segment Header *****

 MDDIT'P         =%000043  PORTAREA'P     =%000647  VFCAREA'P      =%000247
 HIGHLDEV'WRD    = 58      PORTDUMP'P     =%017752  WAITHEAD'P     =%000000
 LOWLDEV'WRD     = 20      TBUFTBL'P      =%022376  WAITTAIL'P     =%177777
 MSGTBL'P        =%017677  TDS'VER        = 2

***** Software Versions *****

SOFTWARE LEVEL  G.51.32
ADCCDRIVER      G.04.44
ADCCINIT        G.04.18
IHANDLER        G.04.36
IMANAGER        G.04.71
LPMON           G.04.19
TDS'VER         2
TERMDSM         G.05.05.0
TERMMON         G.04.28
TERMUTIL        G.04.27
```

Now back to our detective role. The TDS maintains pointers to the other DITs required for the port. If the pointers were bad the port would indeed hang, but more likely is a system failure! The version numbers are primarily for reference. Unfortunately our suspect's nosy habits are of little use to us in this case. After viewing the TDS, we can't hold this suspect; the verdict would be innocent.

Terminal Monitor DIT (MONDIT)

The MONDIT contains flags which indicate whether or not a device has been "DOWNed" or placed in diagnostics (:SHOWDEV will also display this information). MONDIT also contains the flags for pre-spacing and post-spacing (FCONTROL 1 with parm equal %100 or %101). The MONDIT is also the place to look to see if the user is "in BREAK", running an interactive session or job.

```
***** Terminal Monitor DIT *****

 DL'ACK'TO       = 0       DL'LDEV        =%000056  DL'READ'TIME   = 0
 DL'ACTIVE       = 0       DL'LOG         =%000000  DL'READ'TO     = 0
 DL'BINARY       = 0       DL'LOG1        =%000000  DL'READ'TVAL   = 0
 DL'BRK'MOD      = 0       DL'LOGN'TRLX   =%000000  DL'REQUEST     = 0
 DL'BROKEN       = 0       DL'LOGON'TO    = 0       DL'RESET       = 0
 DL'CFAIL'TO     = 0       DL'LOGON'TYP   = 1       DL'SAVE'PF     = 0
 DL'CONSOLE      = 0       DL'MCC'STAT    =%000000  DL'SPD'SNS     = 0
 DL'CONTROLLER   = 2       DL'MCC'VER     = 0       DL'SSTO        = 0
 DL'DATE'CODE    =%000000  DL'MISC        =%000103  DL'START'SS    =!0000
 DL'DEVTYPE      = 16      DL'MISC3       =%040020  DL'TBUFAVAIL   = 0
 DL'DLTP         =%072300  DL'MSC'STAT    =%000000  DL'TEMP        =%000000
 DL'DMI'STATE    =!0000    DL'MSC'VER     = 0       DL'TERM        = 1
 DL'DONT'CAT     = 0       DL'NEXT        =%000000  DL'TICK        =%000000
 DL'ERROR'CODE   = 0       DL'PCC'STAT    =%000000  DL'TIME        =%000000
 DL'FLAGS        =%140000  DL'PCC'VER     = 0       DL'TIME'FLAG   = 0
 DL'FLUSH        = 0       DL'PD'DITP     =%012503  DL'TRUEUNIT    = 0
 DL'HANGUP'TO    = 0       DL'PF'REC      = 0       DL'UNFIXABLE   = 0
 DL'ILTP         =%062704  DL'PREEMPT     = 0       DL'UNIT        =%000000
 DL'INT'MAN      =!0000    DL'PRESPACE    = 0       DL'UP          = 1
 DL'INT1         = 0       DL'PRP'LEVEL   = 3       DL'VERSION     = 0
 DL'IOQP         =%005464  DL'QPARM       = 3       DL'WAIT'RSN    = 0
```

And now a few words of explanation of MONDIT's street slang, so you can follow the progress of the questioning of our informer:

```
DL'TERM              1 - device is a terminal
```

DL'UP	0 - device not in use 1 - device has been speed-sensed or FOPENed
DL'BROKEN	1 - Port is broken
DL'LDEV	nnn - Logical device number for port
DL'PRESPACE	0 - Pre-spacing not enabled 1 - Pre-spacing is enabled
DL'BINARY	0 - Not Binary Mode 1 - Binary Mode
DL'SPD'SNS	1 - Device was speed-sensed
DL'FLUSH	1 - BREAK was seen, and has been processed
DL'LOGON'TYP	0 - :DATA 1 - :HELLO 3 - :JOB
DL'DATE'CODE	Contains the ATP hardware date code (ADCCs not used)
DL'DEVTYPE	The driver type, 16 for terminals
DL'SSTO	0 - If logon was not successful and the configured logon timeout is reached then disconnect the port. 1 - If logon was not successful and the configured logon timeout is reached, then reset and start new speed-sense process.
DL'BRK'MODE	0 - NOT IN BREAK 1 - IN BREAK
DL'ERROR'CODE	ATP failure code
DL'UNFIXABLE	1 - port is unfixable, WARMSTART is required

While interrogating the MONDIT, the detective made the following observations. The first is that the port was not broken and it was "up". The logical device for this port is %56 (decimal 46). Pre-spacing was not enabled and a BREAK had not been seen or processed. The device type is 16, just what we would expect for a terminal. Most of the suspect's ramblings (other fields) were considered not to be relevant to the investigation. Verdict: innocent.

Port Protocol DIT

The characteristics displayed in the PPDIT can be altered by using HP's Workstation Configurator. Always check the current termtype number and if a termtype file was specified. (Termtype can be specified with FCONTROL or at logon by entering `:HELLO user.acct;TERM=nn` or `TERM=termtype filename`). After the PPDIT fields have been formatted, TERMDSM will also show what special characters are in effect. If the special characters have been modified (via FCONTROL), it is possible that many programs will not operate as expected.

```
***** Port Protocol DIT *****

Last termtype file name : none
Current term type : 10
 PP'2631B'FIX    = 0        PP'ENQACK       = 1        PP'ODD'ENAB     = 0
 PP'ACKCHAR      = ACK      PP'ENQBLOCK     = 80       PP'ODD'PARITY   = 0
 PP'BLOCK'TRIG   = DC1      PP'ENQCHAR      = ENQ      PP'PARITY'ENAB  = 0
 PP'BLOCKMODE    = 3        PP'EVEN'ENAB    = 1        PP'STAT'WAIT    = 0
 PP'BSRESP       = 1        PP'EVEN'PARITY  = 2        PP'STATUS'RETRY= 0
 PP'CHARSIZE     = 6        PP'FF'NEWCHAR   = NUL      PP'SW'XONXOFF   = 0
 PP'CONS'STRIP   = 0        PP'FFOK         = 1        PP'TAILTBUF     =%000000
 PP'DC3'CCTL     = 0        PP'FOPEN'PARITY= 0         PP'TBUFS'IN'USE= 0
 PP'DELAY        = 0        PP'HEADTBUF     =%000000   PP'TRIGGER'CHAR= DC1
 PP'DELAYCR      = 0        PP'INIT'DEV     = 0        PP'VFC          =%000000
 PP'DELAYFF      = 0        PP'LAST'SSBRK   = EM       PP'VFC'OK       = 0
 PP'DELAYLF      = 0        PP'MIN'SR       = 0        PP'WRITESTATUS  = 0
 PP'DO'XON'TIMER= 0         PP'NAME'VALID   = 0        PP'XFLOW        = 1
 PP'DTR'LOW'TIME= 0         PP'NETWRK'DEV   = 0        PP'XOFF'DC1     = 0
 PP'ECHO         = 1        PP'NETWRK'WAIT  = 0        PP'XON'TIME     = 60
 PP'EMSTRIP      = 1        PP'NOACKACTION  = 1        PP'XSTRIP       = 1
 PP'ENQ          =%050005

*** Special Characters ***

Console attention      : SOH
Cancel 1 character     : BS
Linefeed               : LF
Type 1 EOR             : CR
Block-mode alert       : DC2
Cancel line            : CAN
Subsystem break        : EM
Strip and ignore       : NUL, DEL
```

Here are the inspector's notes from the questioning of this many-faced suspect:

```
PP'ECHO             0 - echo disabled
                    1 - echo enabled

PP'ENQACK           0 - disable ENQ/ACK
                    1 - enable ENQ/ACK

PP'DELAY            0 - disable delays
                    1 - enable delays after CR, LF, FF

PP'XFLOW            0 - disable XON/XOFF
                    1 - enable XON/XOFF
```

PP'XSTRIP	0 - do not strip XON/XOFF from read data 1 - remove XON/XOFF from read data
PP'EMSTRIP	0 - do not strip CTRL-Y from read data 1 - strip CTRL-Y from read data
PP'CONS'STRIP	0 - do not strip CTRL-A from read data 1 - strip CTRL-A from read data
PP'FFOK	0 - replace form feed (FF) with contents of PP'FF'NEWCHAR before output 1 - allow FF
PP'DC3'CONTROL	0 - do nothing 1 - Append DC3 to line after every CR, LF
PP'BLOCKMODE	0 - do not start read when DC2 is seen 1 - Line blockmode 2 - Page blockmode 3 - either line or page blockmode
PP'DO'XON'TIMER	0 - do not limit XOFF flow control 1 - start XON timer (see PP'XON'TIME)
PP'VFC'OK	1 - There is a VFC file for the device
PP'NAME'VALID	0 - Use HP FOS termtype files 1 - Use custom termtype file
PP'DELAYCR	time in .1 second for delay after CR
PP'DELAYLF	time in .1 second for delay after LF
PP'DELAYFF	time in .1 second for delay after FF
PP'ENQBLOCK	Size in characters of the ENQ/ACK block
PP'ENQCHAR	ENQ character
PP'ACKCHAR	ACK character
PP'BLOCK'TRIG	DC1 - read trigger character
PP'TRIGGER'CHAR	DC1 - normal character-mode trigger

```
PP'BSRESP           Action that will be taken for a BackSpace
                    1 - Nothing
                    2 - Send End-Of-Medium
                    3 - Send LF
                    4 - Send /
                    5 - Erase the character

PP'PARITY'ENAB      0 - Parity checking is disabled
                    1 - Enable parity checking

PP'FOPEN'PARITY     The parity which should be used
                    0 - Space
                    1 - Mark
                    2 - Even
                    3 - Odd

PP'ODD'ENAB         1 - If odd parity was sensed, parity
                        checking is enabled

PP'EVEN'ENAB        1 - If even parity was sensed, parity
                        checking is enabled

PP'EVEN'PARITY      Indicates what type of parity should be
PP'ODD'PARITY       generated if that parity was sensed.
                    0 - Space
                    1 - Mark
                    2 - Even
                    3 - Odd

PP'XON'TIME         The amount of time to wait for XON.
                    (in seconds)
```

The detective's analysis of the PPDIT revealed nothing unusual. The port was configured as type 10, typical for HP terminals. The special character table was what is considered normal. The trigger character PP'TRIGGER'CHAR was a DC1 as expected. I have included PPDIT displays for a spooled Laserjet II and a HP7750 plotter. Can you see any differences?

PPDIT for Laserjet II

```
***** Port Protocol DIT *****

Termtype filename     : TTPCL18.PUB.SYS
 PP'2631B'FIX   = 0      PP'ENQACK      = 0      PP'ODD'ENAB    = 0
 PP'ACKCHAR     = ACK    PP'ENQBLOCK    = 0      PP'ODD'PARITY  = 0
 PP'BLOCK'TRIG  = DC1    PP'ENQCHAR     = ENQ    PP'PARITY'ENAB = 0
 PP'BLOCKMODE   = 0      PP'EVEN'ENAB   = 1      PP'STAT'WAIT   = 0
 PP'BSRESP      = 1      PP'EVEN'PARITY = 2      PP'STATUS'RETRY= 0
 PP'CHARSIZE    = 7      PP'FF'NEWCHAR  = NUL    PP'SW'XONXOFF  = 0
 PP'CONS'STRIP  = 0      PP'FFOK        = 1      PP'TAILTBUF    =%000000
 PP'DC3'CCTL    = 0      PP'FOPEN'PARITY= 0      PP'TBUFS'IN'USE= 0
 PP'DELAY       = 0      PP'HEADTBUF    =%000000 PP'TRIGGER'CHAR= NUL
 PP'DELAYCR     = 0      PP'INIT'DEV    = 1      PP'VFC         =%000247
 PP'DELAYFF     = 0      PP'LAST'SSBRK  = NUL    PP'VFC'OK      = 1
 PP'DELAYLF     = 0      PP'MIN'SR      = 0      PP'WRITESTATUS = 0
 PP'DO'XON'TIMER= 1      PP'NAME'VALID  = 1      PP'XFLOW       = 1
 PP'DTR'LOW'TIME= 0      PP'NETWRK'DEV  = 0      PP'XOFF'DC1    = 0
 PP'ECHO        = 1      PP'NETWRK'WAIT = 0      PP'XON'TIME    = 60
 PP'EMSTRIP     = 1      PP'NOACKACTION = 1      PP'XSTRIP      = 1
 PP'ENQ         =%000005

*** Special Characters ***

Console attention     : SOH
Cancel 1 character    : BS
Linefeed              : LF
Type 1 EOR            : CR
Cancel line           : CAN
Subsystem break       : EM
Strip and ignore      : NUL, DEL
```

PPDIT for spooled 7550

```
***** Port Protocol DIT *****

Termtype filename     : TT7550.PUB.SYS
 PP'2631B'FIX   = 0      PP'ENQACK      = 0      PP'ODD'ENAB    = 0
 PP'ACKCHAR     = ACK    PP'ENQBLOCK    = 0      PP'ODD'PARITY  = 0
 PP'BLOCK'TRIG  = DC1    PP'ENQCHAR     = ENQ    PP'PARITY'ENAB = 0
 PP'BLOCKMODE   = 0      PP'EVEN'ENAB   = 1      PP'STAT'WAIT   = 0
 PP'BSRESP      = 1      PP'EVEN'PARITY = 2      PP'STATUS'RETRY= 0
 PP'CHARSIZE    = 6      PP'FF'NEWCHAR  = NUL    PP'SW'XONXOFF  = 1
 PP'CONS'STRIP  = 0      PP'FFOK        = 1      PP'TAILTBUF    =%000000
 PP'DC3'CCTL    = 0      PP'FOPEN'PARITY= 0      PP'TBUFS'IN'USE= 0
 PP'DELAY       = 0      PP'HEADTBUF    =%000000 PP'TRIGGER'CHAR= NUL
 PP'DELAYCR     = 0      PP'INIT'DEV    = 1      PP'VFC         =%000267
 PP'DELAYFF     = 0      PP'LAST'SSBRK  = NUL    PP'VFC'OK      = 0
 PP'DELAYLF     = 0      PP'MIN'SR      = 0      PP'WRITESTATUS = 0
 PP'DO'XON'TIMER= 0      PP'NAME'VALID  = 1      PP'XFLOW       = 1
 PP'DTR'LOW'TIME= 0      PP'NETWRK'DEV  = 0      PP'XOFF'DC1    = 0
 PP'ECHO        = 1      PP'NETWRK'WAIT = 0      PP'XON'TIME    = 60
 PP'EMSTRIP     = 1      PP'NOACKACTION = 1      PP'XSTRIP      = 1
 PP'ENQ         =%000005

*** Special Characters ***

Console attention     : SOH
Cancel 1 character    : BS
Linefeed              : LF
Type 1 EOR            : CR
Cancel line           : CAN
Subsystem break       : EM
Strip and ignore      : NUL, DEL
```

VFC TABLE

VFC tables are used for spooled devices. The dump in this case was of a device configured as a terminal.

```
No VFC entry associated with LDEV #46
```

VFC claims not that he was not anywhere near the scene of the crime. His alibi holds up. Terminals do not associate with VFCs. I have therefore taken the opportunity to include additional VFCs for a Laserjet II and an HP7550.

VFC entry for Laserjet II

```
***** VFC Entry *****

VFC Filename : VFCPCL.PUB.SYS

 VFC'DATABUF0    =%022515  VFC'INITBUF    =%022410  VFC'USE'COUNT  = 1
 VFC'DATABUF1    =%022622

***** VFC Initialization Buffer *****
WORD
0     000000 140000 015532 015505 015446 066061 046000 000000   .....Z.E.&l1L...
10    000000 000000 *** SAME TO 77 ***                          ................
100   000000 000000 000000 000000 000000                        ..........

***** VFC Data Buffer *****
WORD
0     000000 015446 066060 030526 000000 000000 000000 000000   ...&l01V........
10    000000 015446 066060 031126 000000 000000 000000 000000   ...&l02V........
            ***  VFC DATA BUFFER DISPLAY SHORTENED ***
```

VFC for spooled 7550

```
***** VFC Entry *****

VFC Filename : VFC7550.PUB.SYS

 VFC'DATABUF0    =%023034  VFC'INITBUF    =%022727  VFC'USE'COUNT  = 1
 VFC'DATABUF1    =%023141

***** VFC Initialization Buffer *****
WORD
0     000000 100000 015456 024033 027116 032473 030471 035461   ......(..N5;19;1
10    034472 015456 044470 030073 035461 033473 030467 035111   9:..180;;17;17:I
20    047073 000000 000000 000000 000000 000000 000000 000000   N;..............
              *** SAME TO 77 ***
100   000000 000000 000000 000000 000000                        ..........


***** VFC Data Buffer *****
WORD
0     000000 000000 000000 000000 000000 000000 000000 000000   ................
              *** VFC DATA BUFFER = 0 ***
```

Protocol and Data Manager DIT (PDDIT)

The PDDIT brings us closer to the actual port hardware. In the PDDIT we are finally able to see if a "powerfail" has been encountered, whether or not we are dealing with a modem, and if so, what the modem status is. The investigator can easily determine the port speed. If the terminal is processing a V/3000 read, that too is visible.

```
Protocol and Data level DIT *****

***** Fixed Area *****

 PD'ALL'PARITY  =%000000  PD'DSRTIMER     = 0        PD'PORTSTATE    = 1
 PD'ALTCHARSET  = 1       PD'DTRTIMER     = 0        PD'POWERFAIL    = 0
 PD'BROKEN      = 0       PD'HARDWARE'TYP=%000000   PD'PPENTRYNUMB = 10
 PD'CF'CNT      = 0       PD'IODITP      =%012341   PD'PPROTOCOL    =%012520
 PD'CFTIMER     =%000000  PD'LINETYPE     = 0        PD'RPARITY      = 0
 PD'CHARSIZE    = 0       PD'MODEM'SIGNAL= 174      PD'RWPORTSTATE = 1
 PD'CLEARF      = 0       PD'MODEM'STATE = 6        PD'SPEED'SPECIF= 0
 PD'CONNECTTYPE = 1       PD'PARITYENAB  = 0        PD'TERMTYPE     = 10
 PD'CONTROLLER  = 2       PD'PENDING'STAR= 0        PD'WPARITY      = 0
 PD'DC'MODEM    = 0       PD'PORTSPEED   = 120      PD'XONTIMER     =%000000
 PD'DPORTSPEED  = 120


***** Variable Area *****

 PD'2631B'RESET = 0       PD'ESCPAIR      = 0        PD'READFLAGS    =%000003
 PD'ABSOFFSET   =%000002  PD'FILLING      = 0        PD'READLOC      = 1
 PD'ALTCHARS    =%00000C  PD'HEADOFFSET  =%000002   PD'READTIME     = 0
 PD'ALTEOR      = NUL     PD'HEADTBUF    =%054076   PD'READTYPE     = 1
 PD'ALTSSBREAK  = NUL     PD'IOQEOR       = NUL      PD'SBUF1        =%054076
 PD'BANKNUMB    =%054076  PD'LAST'STATUS = 0        PD'SBUF1'STAT  = 0
 PD'BANKOFFSET  =%054076  PD'LASTEOR      = NUL      PD'SBUF2        =%054076
 PD'BINARY'MODE = 0       PD'LBLOCKMODE  = 0        PD'SBUF2'STAT  = 0
 PD'BINARYREAD  = 0       PD'LDMDIT      =%063244   PD'SBUFREADCOMP= 0
 PD'BLOCKMODE   = 0       PD'LDMOPCODE   = 7        PD'SPDSENSE     = 1
 PD'BREAK       = 0       PD'LLDMC        = 2        PD'SSBREAK      = 0
 PD'BREAKENAB   = 1       PD'LOGONDEV    = 1        PD'SSBRKENAB    = 0
 PD'BREAKMODE   = 0       PD'LOPCOMPLETE = 0        PD'STAT'COUNT'2= 0
 PD'BRKTBUF     =%000000  PD'LOPSTATE     = 1        PD'STATUS       =%000000
 PD'BROKRCNT    = 0       PD'NETWK'SR'MOD= 0        PD'STATUS'RETRY= 0
 PD'BTANKED     = -1      PD'NETWRK'WAIT = 0        PD'STATUS'WCNT = 0
 PD'CHARSET     = 1       PD'NEWLINE      = 0        PD'SUSPLOPSTATE= 0
 PD'CNTRLX      = 1       PD'NEWTOP       = 0        PD'TAILOFFSET  =%000002
 PD'CONSENAB    = 0       PD'NO'READECHO = 0        PD'TAILTBUF     =%054076
 PD'CONSMODE    = 0       PD'NOLF         = 0        PD'TBUFS'IN'USE= 0
 PD'CRITICALW   = 0       PD'OLDXFERCNT  = 0        PD'TBUFWAIT     =%000000
 PD'DC2READ     = 0       PD'OWNREAD      = 0        PD'TIMINGREAD  = 0
 PD'DEVLINK     =%000000  PD'PCC'XON'XOFF= 1        PD'TRANSPARENT =%000000
 PD'DISCNCT'DEV = 0       PD'PENDLOPSTATE= 0        PD'VIEWREAD     = 0
 PD'DO'STATREQ  = 0       PD'PREADTIMING =%000000   PD'WAITFORTBUF = 0
 PD'EOF         = 3       PD'PRINTER      = 0        PD'WWC          = 0
 PD'EOFCNT      = 0       PD'RDTIMEOUTVAL= 0        PD'XFERCNT      = 0
 PD'EOFTBUF     =%000000  PD'RDTIMERINDEX=%000000   PD'XON'RETRYS  = 0
 PD'ERROR       = 0       PD'READCNT      = 279      PD'XONWAIT      = 0
 PD'DISCADDR    = (D)%00000000000  PD'RDSTARTIME  =%(D) 00000000000
```

These are some of the diplomatic terms favored by this suspect:

```
PD'PORTSTATE          1 - Reading
                      2 - Writing
                      3 - Idle
                      4   Input save
                     5-8  NOT USED
                      9 - Selftest
                     10 - Speed-sensing
                     11 - Setting up port protocol
                     12 - Set up special characters
                     13 - Control Modem
```

PD'CONTROLLER	1 - ATP 2 - ADCC
PD'LINETYPE	0 - asynchronous 1 - synchronous
PD'CONNECTTYPE	0 - Direct connect, subtype 14 1 - Modem, subtype 1 or 15 2 Modem, subtype 9 or 13
PD'BROKEN	0 - PORT OKAY 1 - PORT BROKEN
PD'POWERFAIL	0 - No powerfail 1 - Powerfail detected
PD'CF'CNT	nn - number of times data carrier has failed HP3000 will disconnect after 50 times.
PD'PENDING'STAR	0 - OKAY 1 - (MODEMS only) A carrier fail was detected and an operation (port controller command) is pending. The operation cannot be started until we have a carrier.
PD'MODEM'SIGNAL	0 - NOT USED 1 - Clear to Send 2 - Signal Quality 3 - Data Set Ready 4 - Call origin status 5 - Secondary carrier detect 6 - Ring indicator 7 - Carrier Detect
PD'PORTSPEED	The terminal (port) speed (Characters per second)
PD'PENTRYNUMB	The terminal type number; if a termtype file was specified, then this will be 31.
PD'TERMTYPE	The terminal type now in effect; 0 = termtype file was specified.

PD'LOPSTATE	Port Logical Operation State 0 - No operation in progress 1 - Reading 2 - Writing 3 - Status Request for a device 4 - Reading Status 5 - Responding to CTRL-X 6 - Waiting for ENTER or RETURN 7 - Write data and wait for read 8 - Setup pending read 9 - Write and then ask for status 10 - Speed-sense 11 - Set up port protocol 12 - Set up special characters 13 - Set up modem control lines 14 - Not Used 15 - Set up V/3000 read
PD'NO'READECHO	0 - Echo is enabled 1 - Echo is disabled
PD'BREAK	0 - BREAK has not been seen 1 - BREAK key has been detected
PD'SSBREAK	0 - CTRL-Y has not been seen 1 - CTRL-Y has been detected.
PD'VIEWREAD	0 - NOT A V/3000 READ 1 - V/3000 READ
PD'BINARYREAD	0 - Not a binary read 1 - Binary read
PD'ALTEOR	EOR character as specified in FCONTROL 41
PD'ALTSSBREAK	Alternate subsystem break; (it will not be stripped from read buffer)
PD'NEWTOP	0 - not at top of form 1 - at top of form
PD'BREAKMODE	0 - Not in break 1 - In break
PD'SSBRKENAB	0 - Ignore subsystem breaks (FCONTROL 16) 1 - Subsystem breaks are processed (FCONTROL 17)
PD'BREAKENAB	0 - BREAK is disabled (FCONTROL 14) 1 - BREAK is enabled (FCONTROL 15)

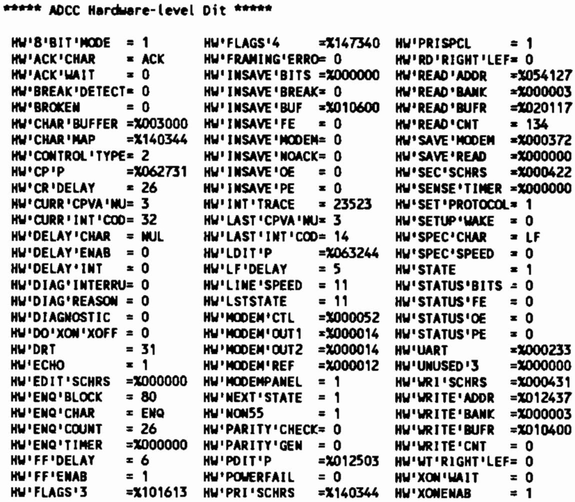
PD'WAITFORTBUF	0 - All okay, not waiting for TBUF 1 - Waiting for TBUF!
PD'PCC'XON'XOFF	0 - I/O driver will perform XON/XOFF handshake 1 - Hardware will perform XON/XOFF handshake
PD'READTIME	Elapsed read time (FCONTROL 22)
PD'TIMINGREAD	0 - Terminal input timer disabled (FCONTROL 20) 1 - Terminal input timer enabled (FCONTROL 21)
PD'RDTIMEOUTVAL	Number of .1 seconds for timed read (FCONTROL 4)
PD'ERROR	Driver sensed an error! 1 - Out of buffers 2 - Data Overrun 3 - Framing error 4 - Not Used 5 - Parity error 6 - Not used 7 - Modem error ?? 8 - Not used 9 - Not used
PD'DC2READ	0 - DC2 not read [ENTER] 1 - DC2 has been read, will initiate a blockmode or linemode read
PD'XONWAIT	0 - Not waiting for XON 1 - XOFF was seen, waiting for XON (only set up software is performing the XON/XOFF handshake)
PD'LBLOCKMODE^@	1 - Line Blockmode read was made
PD'BLOCKMODE	1 - Blockmode read
PD'BINARY'MODE	0 - Not in binary mode (FCONTROL 26) 1 - Binary transfers enabled (FCONTROL 27)

PD'READTYPE	(Type of read in progress) 0 - No read in progress 1 - Character mode read 2 - Spooled read 3 - Idle read 4 - transparent read (no editing) 5 - V/3000 read 6 - Binary read
PD'PRINTER	1 - device is a printer
PD'READLOC	0 - Not Used 1 - Read into TBUF 2 - Read into system buffer 3 - Read into frozen segment (NO-WAIT I/O)
PD'STATUS	status returned from printer (Only if port was configured as a printer)

The detective's review of the PDDIT showed that the port was setup as a 1200 baud modem (subtype 1 or 15). The port was not broken, and no powerfail had been encountered. The terminal type is 10, (it matches the other tables), and we are not in a V/3000 read (PD’VIEWREAD=0). The field PD’LOPSTATE says that we are reading or waiting for a read. The PD’MODEM’SIGNAL is %174 which decodes to "clear to send", "signal quality", "data set ready", "call origin status", and "secondary carrier detect", BUT NO CARRIER DETECT! Why? This is very unusual. This suspect will require closer surveillance, but we should not make a judgement yet because we still have several more suspects to question.

Hardware DIT (HWDIT)

The HWDIT will reflect the "opinion" of the actual computer hardware. Generally the contents of the HWDIT should be in agreement with the more sophisticated DITs already discussed. However keep in mind that in some cases such as XON/XOFF, the actual handshaking may be done by the software driver and not the hardware. (In that case the flags may actually disagree!) Remember that the HWDIT is a mechanic, so we are especially interested in the mechanical work that he has performed. A good inspector will look for data format (8 bit mode), which parity is in effect, any errors, what delays will be used, port speed, has [BREAK] been seen, echo on or off, and if a modem - what the modem line values are.



```
***** ADCC Hardware-level Dit *****

 HW'8'BIT'MODE  = 1       HW'FLAGS'4     =%147340 HW'PRISPCL     = 1
 HW'ACK'CHAR    = ACK     HW'FRAMING'ERRO= 0      HW'RD'RIGHT'LEF= 0
 HW'ACK'WAIT    = 0       HW'INSAVE'BITS =%000000 HW'READ'ADDR   =%054127
 HW'BREAK'DETECT= 0       HW'INSAVE'BREAK= 0      HW'READ'BANK   =%000003
 HW'BROKEN      = 0       HW'INSAVE'BUF  =%010600 HW'READ'BUFR   =%020117
 HW'CHAR'BUFFER =%003000  HW'INSAVE'FE   = 0      HW'READ'CNT    = 134
 HW'CHAR'MAP    =%140344  HW'INSAVE'MODEM= 0      HW'SAVE'MODEM  =%000372
 HW'CONTROL'TYPE= 2       HW'INSAVE'NOACK= 0      HW'SAVE'READ   =%000000
 HW'CP'P        =%062731  HW'INSAVE'OE   = 0      HW'SEC'SCHRS   =%000422
 HW'CR'DELAY    = 26      HW'INSAVE'PE   = 0      HW'SENSE'TIMER =%000000
 HW'CURR'CPVA'NU= 3       HW'INT'TRACE   = 23523  HW'SET'PROTOCOL= 1
 HW'CURR'INT'COD= 32      HW'LAST'CPVA'NU= 3      HW'SETUP'WAKE  = 0
 HW'DELAY'CHAR  = NUL     HW'LAST'INT'COD= 14     HW'SPEC'CHAR   = LF
 HW'DELAY'ENAB  = 0       HW'LDIT'P      =%063244 HW'SPEC'SPEED  = 0
 HW'DELAY'INT   = 0       HW'LF'DELAY    = 5      HW'STATE       = 1
 HW'DIAG'INTERRU= 0       HW'LINE'SPEED  = 11     HW'STATUS'BITS = 0
 HW'DIAG'REASON = 0       HW'LSTSTATE    = 11     HW'STATUS'FE   = 0
 HW'DIAGNOSTIC  = 0       HW'MODEM'CTL   =%000052 HW'STATUS'OE   = 0
 HW'DO'XON'XOFF = 0       HW'MODEM'OUT1  =%000014 HW'STATUS'PE   = 0
 HW'DRT         = 31      HW'MODEM'OUT2  =%000014 HW'UART        =%000233
 HW'ECHO        = 1       HW'MODEM'REF   =%000012 HW'UNUSED'3    =%000000
 HW'EDIT'SCHRS  =%000000  HW'MODEMPANEL  = 1      HW'WRI'SCHRS   =%000431
 HW'ENQ'BLOCK   = 80      HW'NEXT'STATE  = 1      HW'WRITE'ADDR  =%012437
 HW'ENQ'CHAR    = ENQ     HW'NON55       = 1      HW'WRITE'BANK  =%000003
 HW'ENQ'COUNT   = 26      HW'PARITY'CHECK= 0      HW'WRITE'BUFR  =%010400
 HW'ENQ'TIMER   =%000000  HW'PARITY'GEN  = 0      HW'WRITE'CNT   = 0
 HW'FF'DELAY    = 6       HW'PDIT'P      =%012503 HW'WT'RIGHT'LEF= 0
 HW'FF'ENAB     = 1       HW'POWERFAIL   = 0      HW'XON'WAIT    = 0
 HW'FLAGS'3     =%101613  HW'PRI'SCHRS   =%140344 HW'XONENAB     = 1
```

Here are some our social introvert's favorites topics of conversation:

```
HW'CONTROL'TYPE 1 = ATP
                2 = ADCC

HW'MODEMPANEL   0 = The port is configured as a direct connect,
                    subtype 14
                1 = The port is configured for a modem,
                    subtype 15

HW'WAIT'REASON  1 = Abort Pending
 (ATP ONLY)     2 = The port is being reset
                3 = Modem disconnect in progress
                4 = ?
                5 = Port just initialized, waiting for modem
                6 = Hung

HW'ECHO         0 = Echo is disabled
                1 = Echo is enabled
```

```
HW'STATE           1 = reading
                   2 = writing
                   3 = speed-sensing
                   4 - 9 = ?
                  10 = reading modem inputs
                  11 = idle read
                  12 = setting up modem
                  13 = monitoring modem signals

HW'POWERFAIL       0 = Normal, no powerfail
                   1 = Powerfail has occurred

HW'BROKEN          0 = The port is not broken
                   1 = The port is broken

HW'DELAY'ENAB      0 = Delays will follow CR,LF or FF
                   1 = No delays are added (NULs)

HW'FF'ENAB         0 = Replace FormFeed (FF) with a Linefeed
                   1 = Pass FF through without editing

HW'XONENAB         0 = Hardware will not perform XON/XOFF handshake
                   1 = XON/XOFF will be performed

HW'8'BIT'MODE      0 = data is 7 bit with 1 bit parity
                   1 = data is 8 bit data

HW'PARITY'GEN      (FOR 7 BIT DATA ONLY, SEE HW'8'BIT'MODE)
```

	ADCC	ATP & TIC
0	EVEN	BIT8 =0
1	ODD	BIT8 =1
2	EVEN	EVEN
3	ODD	ODD

```
HW'PARITY'CHECK   0 = do not check parity for received characters
                  1 = check received character parity
```

HW'LINE'SPEED

	ADCC	ATP & TIC
0	NOT USED	110 baud
1	NOT USED	300 baud
2	?	600 baud
3	?	1200 baud
4	?	2400 baud
5	?	4800 baud
6	600 baud	19200 baud
7	2400 baud	9600 baud
8	9600 baud	NOT USED
9	4800 baud	9600 baud
10	NOT USED	1200 baud
11	1200 baud	300 baud
12	2400 baud	?
13	300 baud	?
14	150	?
15	110	?

HW'READ'CNT byte count, reset to 0 after CR

HW'WRITE'CNT byte count, written thus far, reset at CR

HW'FRAMING'ERRO the number of framing errors seen thus far

HW'SPEC'CHAR contains the last special character detected.

HW'MODEM'OUTPUT These are the output lines from the HP3000
(ATP) to the modem
HW'MODEM'OUT1 & 2
(ADCC)

Bit	ADCC	ATP & TIC
0	NOT USED	speed select
1	NOT USED	?
2	?	?
3	?	?
4	Request-To-Send	Secondary Request-To-Send
5	Data-terminal-ready	call request
6	speed select	Request-To-Send
7	Secondary Request-To-Send	Data-Terminal-Ready

HW'MODEM'REF (reference MASK; "1" = check for change in state)

Bit	ADCC	ATP & TIC
0	NOT USED	NOT USED
1	NOT USED	Clear-To-Send
2	?	Signal Quality
3	Clear-To-Send	Data-Set-Ready
4	Data-Set-Ready	Call Origin Status
5	Ring indicator	Secondary Carrier Detect
6	Data Carrier Detect	Ring Indicator
7	Secondary Channel Detect	Carrier Detect

**** ADCCs ONLY ****

HW'SAVE'MODEM Same format as HW'MODEM'REF but indicates actual status of the signal lines from the modem

HW'XON'WAIT XOFF has been seen; we are waiting for an XON

HW'ACK'WAIT ENQ was sent; we are waiting for an ACK to be returned

HW'ENQ'BLOCK The number of characters which will be sent in between ENQ and ACK. "0" disables ENQ/ACK

HW'ENQ'COUNT The number of characters to be sent to the terminal before another ENQ will be sent. (Decreases as each character is sent).

HW'CR'DELAY nn * .1 second delay after each CR

HW'LF'DELAY nn * .1 second delay after each LF

HW'FF'DELAY nn * .1 second delay after each FF

After a long discussion with the mechanic and a close examination of his work, the detective has concluded that the agent (LDEV) has been instructed to work with 8 bit data, no parity detection or generation, no delays will be added to the data. The port speed is set to 1200 baud. The modem states were quite intriguing! The HP3000 had set the "clear-to-send" and "data-terminal-ready" control lines. What is so interesting is that the modem output states are what you would expect for a port that is either currently connected (it was not) or a port that has had the carrier drop was disconnected and is now waiting for another call! Something is very fishy! This suspect will also require closer surveillance.

ATP CNTRLPROG and ADCC CHANPROG

CNTRLPROG and CHANPROG both take their orders from the other DITs and I/O process. Since we know that these two suspects do only as they are asked and never have done anything on their own initiative, the inspector feels that they are innocent.

The ADCC channel & ATP control programs are beyond the scope of this paper.

IOQ

The IOQ is the boundary where we leave the realm of the file system (FOPEN, FCONTROL, FREAD, FWRITE, etc.) The higher order file system intrinsics act as guardians to the underworld, checking and ensuring that the user requests are reasonable. Once past the guardians the user's request is translated into parameters for an internal procedure called ATTACHIO. After entering the world of ATTACHIO, life is much more difficult, for a simple error in the calling sequence will result in a SUDDEN DEATH! ATTACHIO looks at the parameters it was passed, consults the appropriate tables and once again translates the user's request into another format. This last translation has a new twist. ATTACHIO will direct the translated command to one of many formats. If the user requested terminal I/O, the command becomes a Terminal IOQ entry. When the new command has been successfully added to the IOQ, ATTACHIO will send a message to the monitoring process for terminals to awake. This monitoring process will reference both the IOQ and the MONDIT for the requested device. It should be more apparent why our MONDIT was looking frayed and tired!

What should our detective look for? What command or function was being requested, general status, was a [BREAK] being processed, etc.

```
***** Terminal IOQ *****

 Q'ABORT        = 0        Q'FUNC         = 0        Q'RPLEVEL      = 0
 Q'ACCESS       = 0        Q'GEN'STAT     = 0        Q'SET'RESET    = 3
 Q'ADDR         =%000001   Q'ITEM         = 3        Q'SPEED1       = 3
 Q'BINARY       = 0        Q'LDEV         = 46       Q'SPEED2       = 0
 Q'BLOCKED      = 1        Q'LINK         =%000000   Q'STAT'WORD    =%000000
 Q'COMPLETED    = 0        Q'LOGON'TYPE   = 3        Q'STATE        = 0
 Q'CONTINUE     = 0        Q'MISC         =%000001   Q'STATUS       =%000000
 Q'COUNT        = -279     Q'OLD          = 1        Q'SUBSYS'BRK   = NUL
 Q'CRITICAL     = 0        Q'OWN'READ     = 0        Q'SUPRESS'LF   = 0
 Q'CTRL'RTRN    =%177351   Q'PARITY       = 3        Q'SYS'BUF      = 0
 Q'DATA'SEG     =%100544   Q'PARM1        =%000003   Q'SYSBF'PTR1   =%100544
 Q'DB'BASED     = 1        Q'PARM2        =%000000   Q'SYSBF'PTR2   =%000001
 Q'DSTN         = 356      Q'PCBN         = 49       Q'SYSBUF'PTR   =%000001
 Q'EOF'COND     = 3        Q'PRESPACE     = 0        Q'TERM'TYPE    = 3
 Q'EOR'CHAR     = ETX      Q'QUALIFIR     = 0        Q'TOKEN        =%000003
 Q'FLAGS        =%006000   Q'RD'TO'VALU   = 3        Q'V3000'READ   = 0
 Q'FLUSH        = 0        Q'READ'EOR     = NUL      Q'WAKE         = 1
 QD'SPOOL'ADR   = (D)%00000600000
```

The inspector general's staff has intercepted the cipher from the ATTACHIO organization. He has provided the following explanation to assist your investigation:

```
Q'ABORT   1 = Abort IOQ entry for device

Q'BINARY  1 = Binary read/writes
```

```
Q'BLOCKED 1 = Wait for I/O to complete

Q'COMPLETED      1 = IOQ request has completed

Q'FLUSH   1 = Subsystem break being processed

Q'FUNC    0 - read (FREAD)
          1 - Write (FWRITE)
          2 & 3 - File open/close (FOPEN/FCLOSE)
          4 - Device close
          5 - Set time-out interval (FCONTROL 4)
          6 - Set speed (FCONTROL 10)
          7 - Set speed (FCONTROL 11)
          8 - Enable echo (FCONTROL 12)
          9 - Disable echo (FCONTROL 13)
          10 - Disable system [BREAK] (FCONTROL 14)
          11 - Enable system [BREAK] (FCONTROL 15)
          12 - Disable subsystem [BREAK] (FCONTROL 16)
          13 - Enable subsystem [BREAK] (FCONTROL 17)
          14/15 - Enable/disable tape mode
          16 - Disable read timer (FCONTROL 20)
          17 - Enable read timer (FCONTROL 21)
          18 - Get timer value (FCONTROL 22)
          19 - Disable parity checking (FCONTROL 23)
          20 - Enable parity checking (FCONTROL 24)
          21 & 22 - ?
          23 - set termtype (FCONTROL 38)
          24 - Allocate terminal (FCONTROL 37)

Q'GEN'STAT
          0 - Not started or completed
          1 - Completed Okay
          2 - EOF detected
          3 - Unusual condition
          4 - irrecoverable error

Q'LOGON'TYPE
          0 - :DATA
          1 - :HELLO
          2 - :JOB

Q'PARITY  same as PP'FOPEN'PARITY

Q'PRESPACE       Perform Carriage Control before writes
                 (FCONTROL 1)

Q'QUALIFIER
            0 - none
           11 - Read terminated by special character
           13 - parity error
           23 - Read timed out
           24 - Block mode transfer time out
           33 - :ABORTIO / :ABORTJOB
           43 - Data overrun
           53 - Data set not ready
           63 - Power fail
          163 - Read timer overflowed
          173 - [BREAK]
          213 - Powered on, lost environment
          273 - VFC reset

Q'STAT'WORD      Q'GEN'STAT and Q'QUALIFIER combined

Q'TOKEN   1 = use first character in buffer
          %53 - CR, no LF, "+"
          %55 - triple space
          %60 - double space "0"
          %61 - go to top-of-form, "1"
          (for more information see CCTL in File system manual)
```

After minutes of tedious deciphering the detective discerned that the last entry made into the IOQ for this agent was a "read" request (Q'FUNC) which is still pending (Q'BLOCKED). The [BREAK] key has not been depressed and we are not in a V/3000 read. The IOQ entry does not appear to contain any incriminating evidence. For now we will assume that the IOQ is innocent.

Logical-to-Physical Device Table (LPDT)

The LPDT is used to map a request for an LDEV into the physical hardware. We personified the LPDT as a psychoanalyst. The LPDT is where we can check to see if the device is "real" or imaginary (virtual) - LPDT'VIRT. Real devices are terminals; virtual devices do not actually perform I/O but pretend to. Spooled device files and INPs are virtual devices. LPDT can also tell us the state of its mind: interactive - LPDT'INTR, job accepting - LPDT'JOBDATA, or duplicative.

```
***** Logical-to-Physical Device Table *****

 LPDT'BREAK     = 1       LPDT'DV'INFO   =%000000 LPDT'SSBREAK   = 0
 LPDT'DATA      = 1       LPDT'EOR       = 0      LPDT'ST'INFO   =%073041
 LPDT'DITP      =%063244  LPDT'INTR      = 1      LPDT'SUBTYPE   = 1
 LPDT'DRSTATE   = 1       LPDT'JOBDATA   = -1     LPDT'VIRT      = 0
 LPDT'DUP       = 1
```

```
LPDT'BREAK       1 = [BREAK] seen OR  C.I. ignore [BREAK]

LPDT'DATA 1 - Data accepting

LPDT'DRSTATE
          0 - Not owned by any process
          1 - Owned by a process
          2 - Service request (DEVREC), a [RETURN] has been
             pressed on a port that had been logged off.
          3 - Device reserved, (:STARTSPOOL in progress)
```

LPDT'DUP 1 - device is duplicative

LPDT'EOR End-of-file type
0 - No EOF
1 - Hardware EOF (tape mark), (:EOF: ??)
2 - :DATA
3 - :EOD
4 - :HELLO
5 - :BYE
6 - :JOB
7 - :EOJ

LPDT'INTR 1 - device is interactive

LPDT'JOBDATA 1 - device is Job accepting

LPDT'SSBREAK 1 - [CTRL-Y] has been detected

LPDT'SUBTYPE device subtype (from system configuration)

LPDT'VIRT 1 - virtual device, spooled device file, INP, etc.

After meeting with the LPDT, the detective left with a warm and comfortable feeling that the LPDT was okay. The detective verified that he was dealing with a real, interactive device and that the device had not encountered any unusual end-of-file condition. Verdict: probably innocent.

Logical Device Table (LDT)

Our accountant, the LDT, takes pride in his work. The LDT maintains a count of the number of FOPENs or allocations made for his device. He calls this number the FILE'USE'CNT; (it is also visible from :SHOWDEV.) A value of one is typical for spooled printers, two for devices with interactive sessions waiting with the Command Interpreter (C.I.) prompt (":"), and more than two generally means that there is a user program running/accessing the LDEV. Three other fields that interest the detective are AVAIL'TO'SYS, DOWN'REQUESTED and SPECIAL'FORMS.

```
***** Logical Device Table (LDT) *****

 AVAIL'TO'DIAG  = 0      DFT'OUTPUT'OEV = 46     SPECIAL'FORMS  = 0
 AVAIL'TO'SYS   = 1      DFT'TO'CLASSIDX= 0      SPOOL'STATE    = 0
 BAUD'RATE'CODE = 11     DOWN'REQUESTED = 0      SPOOLING'ENAB  = 0
 CHANNEL'ID     =!8011   FILE'USE'CNT   = 2      TERM'TYPE'DFT  = 10
 CS'DEVICE      = 0      HEADER'ON      = 0      TRAILER'ON     = 0
 CTL'Y'PIN      = 0      MAIN'PIN       = 49     VDD'INDEX      =%000033
 DEVICE'TYPE    = 16     RECD'WIDTH     = 40
```

The following acronyms are used by our accountant:

AVAIL'TO'DIAG 1 - The LDEV has been down so TERMDSM can run diagnostics

AVAIL'TO'SYS 1 - The device is not "owned" by a session

```
                     (:SHOWDEV displays "A")

BAUD'RATE'CODE    same as HW'LINE'SPEED in HWDIT

DEVICE'TYPE       device type defined in your system configuration

DOWN'REQUESTED    1 - The operator has issued a :DOWN nn on this
                      device.  As soon as it is available to the
                      system it will be marked as "down".

FILE'USE'CNT      number of active FOPEN calls for port

HEADER'ON         1 - Print preceding banner (HEADER)
                      For spooled printers only.  Turned
                      on by :HEADON and off by :HEADOFF

RECD'WIDTH        nn = number of words in each record.
                       (From the system configuration)

SPECIAL'FORMS     1 - Special forms have been requested or are
                      in use by this printer

SPOOL'STATE
                 0 - Not spooled
                 1 - Input spooled
                 2 - Output spooled

SPOOLING'ENAB     1 - Spool queues are open (I believe this
                      means that there is actually output on
                      the way.)

TRAILER'ON        1 - Print trailing banner. (See HEADER'ON.)
```

The detective's examinations of the accountant's books turned up a big zero. The detective verified that the agent (LDEV) was in use by a session (apparently only the C.I.). A down was not pending. Verdict? INNOCENT!

I have included a DISPLAY of an LDT for a spooled Laserjet II. Can you seen any difference?

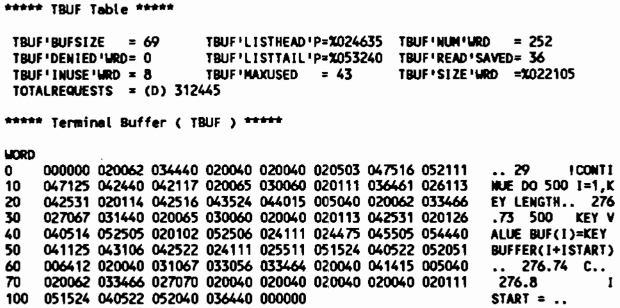
```
***** Logical Device Table (LDT) *****

 AVAIL'TO'DIAG  = 0      DFT'OUTPUT'DEV = 0      SPECIAL'FORMS  = 0
 AVAIL'TO'SYS   = 1      DFT'TO'CLASSIDX= 0      SPOOL'STATE    = 2
 BAUD'RATE'CODE = 8      DOWN'REQUESTED = 0      SPOOLING'ENAB  = 0
 CHANNEL'ID     =!8011   FILE'USE'CNT   = 1      TERM'TYPE'DFT  = 18
 CS'DEVICE      = 0      HEADER'ON      = 1      TRAILER'ON     = 1
 CTL'Y'PIN      = 0      MAIN'PIN       = 14     VDD'INDEX      =%000051
 DEVICE'TYPE    = 32     RECD'WIDTH     = 110
```

DRT, DLT, ILT, ILTX

The DLT, DRT, ILT, ILTX tables are beyond the scope of this paper. For more information consult the MPE V tables manual. (The real adventurous may wish to investigate the MPE Internals Classes offered by HP.)

Terminal Buffers (TBUF)

TERMDSM displays what are actually two separate pieces of information when you specify TBUF. The first is the TBUF table and the second is the actual TBUF. *The TBUF table contains one very important field, the TBUF'DENIED'WRD.* If the value of this field is not 0, then you need to consider changing your system configuration. Data overruns, poor system performance and HUNG ports may result (or have in the past) when insufficient TBUFs are available. TBUF photographic memory can sometimes be misleading, for often TBUF presents a picture not of what currently is happening to the session in question but of what's happened to somebody else. (This occurs because MPE does not blank the TBUF before assigning it to the next terminal read or write.) This photographic memory presents a possible security breach (even for the security monitor) since it will display the actual key strokes that were typed.



```
***** TBUF Table *****

 TBUF'BUFSIZE    = 69        TBUF'LISTHEAD'P=%024635  TBUF'NUM'WRD    = 252
 TBUF'DENIED'WRD= 0          TBUF'LISTTAIL'P=%053240  TBUF'READ'SAVED= 36
 TBUF'INUSE'WRD = 8          TBUF'MAXUSED    = 43     TBUF'SIZE'WRD  =%022105
 TOTALREQUESTS  = (D) 312445

***** Terminal Buffer ( TBUF ) *****

WORD
0     000000 020062 034440 020040 020040 020503 047516 052111    .. 29     !CONTI
10    047125 042440 042117 020065 030060 020111 036461 026113    NUE DO 500 I=1,K
20    042531 020114 042516 043524 044015 005040 020062 033466    EY LENGTH..  276
30    027067 031440 020065 030060 020040 020113 042531 020126    .73  500    KEY V
40    040514 052505 020102 052506 024111 024475 045505 054440    ALUE BUF(I)=KEY
50    041125 043106 042522 024111 025511 051524 040522 052051    BUFFER(I+ISTART)
60    006412 020040 031067 033056 033464 020040 041415 005040    ..  276.74  C..
70    020062 033466 027070 020040 020040 020040 020040 020111     276.8          I
100   051524 040522 052040 036440 000000                         START = ..
```

The detective's notes for the TBUF were:

TBUF'BUFSIZE	size of each buffer
TBUF'DENIED'WORD	number of times there were insufficient TBUFs
TBUF'INUSE'WRD	number of TBUFs currently in use
TBUF'REQUESTS	number of requests for TBUFs since last startup
TBUF'MAXUSED	maximum number of TBUFs used since last startup
TBUF'NUM'WRD	number of TBUFs in TBUF data segment
TBUF'READ'SAVED	number of TBUFs reserved for terminal reads

The detective liked what he saw. Plenty of TBUFs available and no TBUFs ever denied. The actual contents of the TBUF were from a terminal write to a port which we were not interested in. The actual number of valid bytes in the TBUF must be obtained from the other DITs. Verdict: innocent.

Process Control Block & Logical Process Control Block (PCB & LDTPCB)

The PCB and LDTPCB have the same format. Only the LDTPCB is available for spooled devices. Most of the fields are beyond the scope of this paper; however because of its importance, I have displayed a complete PCB and then highlighted the LDTPCBs for a terminal and a spooled device.

```
***** Process Control Block *****

 ALLOW'SOFT'INT = 0        PI'PENDING'CRIT= 0        STACK'DST'INDEX=%000544
 AWAKE'SOFT'OK  = 0        PPC            = 0        STACK'OVF'ABORT= 0
 BLKIDX         = 0        PREEMPT'CAPABLE= 0        STACK'OVF'ALLOC= 0
 BMS            = 0        PREV'IMPED'PIN = 0        STOV           = 0
 BOUNDS'FLAG    = 0        PREV'Q'PTR     =%000000   SYS'CODE'EXEC  = 1
 BPTLINK        = 0        PRIORITY       = 152      WAIT'BLOCKED'IO= 1
 BROTHER'PIN    = 735      PROCESS'ALIVE  = 1        WAIT'FATHER    = 0
 CORE'RESIDENT  = 0        PROCESS'DEAD   = 0        WAIT'FOR'TIMER = 0
 CRITICAL       = 1        PROCESS'TYPE   = 2        WAIT'FOR'TIMOUT= 0
 CST'MAP'DST    = 0        PSEUDO'BREAK   = 0        WAIT'GLOBAL'RIN= 0
 CSTX'MAP'INDEX =%000000   PSEUDO'HARDKILL= 0        WAIT'IMPEDED   = 0
 DB'IS'ABSOLUTE = 1        PSEUDO'HIBERNAT= 0        WAIT'IO        = 0
 DELAYED'SOFT   = 0        PSEUDO'INT'MODE= 7        WAIT'JUNK      = 0
 EVENTFLAGS     =%000000   PSEUDO'SOFTKILL= 0        WAIT'LOCAL'RIN = 0
 FAC            = 0        PSEUDO'SSBREAK = 0        WAIT'LONG      = 1
 FATHER'PIN     = 147      PSEUDO'STOP    = 0        WAIT'MAIL      = 0
 HAS'SIR        = 0        QUANTUM'USED   = 0        WAIT'MEMORY    = 0
 HOLD'IMPED'PRI = 0        QUEUE'CQ       = 1        WAIT'MESSAGE   = 0
 HOLD'SIR'PRI   = 0        QUEUE'DISPQ    = 0        WAIT'MOURNING  = 0
 INCORE'PROT'EXP= 0        QUEUE'DQ       = 0        WAIT'SHORT     = 0
 INTERACTIVE    = 1        QUEUE'EQ       = 0        WAIT'SIR       = 0
 LASTSWAPSEG'CST=%000000   QUEUE'LQ       = 0        WAIT'SON       = 0
 LASTSWAPSEG'NUM=%000000   RITBK          = 0        WAIT'TERM'READ = 1
 LASTSWAPSEG'TYP=%000000   SCHED'ATTN'REQD= 0        WAIT'UCOP      = 0
 NEXT'IMPED'PIN = 0        SI             = 0        WS             = 0
 NEXT'Q'PTR     =%000000   SLL'PTR        =%006636   XDS'DST'INDEX  =%000000
 OA             = 2        SON'PIN        = 0
```

Terminal LDTPCB

```
***** LDT Process Control Block (LDTPCB) ****
      ...                     ...                        ...
      ...                     ...                    WAIT'BLOCKED'IO= 1
      ...                  PROCESS'ALIVE  = 1            ...
      ...                  PROCESS'DEAD   = 0            ...
      ...                     ...                        ...
      ...                     ...                    WAIT'IMPEDED   = 0
      ...                     ...                    WAIT'IO        = 0
      ...                     ...                        ...
      ...                     ...                    WAIT'LONG      = 1
      ...                     ...                        ...
      ...                     ...                    WAIT'MOURNING  = 0
      ...                     ...                        ...
 INTERACTIVE    = 1           ...                    WAIT'SIR       = 0
      ...                     ...                    WAIT'SON       = 0
      ...                     ...                    WAIT'TERM'READ = 1
      ...                     ...
```

LDTPCB for spooled Laserjet II

```
***** LDT Process Control Block (LDTPCB) *****

    ...                 ...               ...
    ...           PROCESS'ALIVE  = 1  WAIT'FATHER    = 1
    ...           PROCESS'DEAD   = 0      ...
    ...           PROCESS'TYPE   = 4      ...
    ...                 ...           WAIT'IO        = 0
    ...                 ...           WAIT'LONG      = 1
    ...                 ...               ...
    ...                 ...           WAIT'SON       = 1
```

Notes on the LDTPCB and PCB:

```
INTERACTIVE       1 - The process is interactive

PROCESS'ALIVE     1 - The process is still considered
                      active by MPE

PROCESS'DEAD      1 - The process is being eliminated by MPE

PROCESS'TYPE
                  0 - user process (CREATEPROCESS)
                  1 - user (:RUN x)
                  2 - user main (C.I.)
                  3 - user main task???
                  4 - system (spoolers, DEVREC, etc)
                  5 - ?
                  6 - system, UCOP (user controller, it processes
                      the :HELLO command)
                  7 - ?

WAIT'BLOCKED'IO   1 - waiting for terminal Read or Write to
                      complete

WAIT'FATHER       1 - waiting to be awoken by father process
                      (spooler waiting for data to print)

WAIT'IMPEDED      1 - waiting for a resource to become available

WAIT'IO           1 - waiting for disc I/O

WAIT'LONG         1 - long term wait, remove from dispatch queue

WAIT'MOURNING     1 - process is mourning the loss of a son

WAIT'SIR          1 - waiting for a system internal resource

WAIT'SON          1 - waiting for son process to wake me up

WAIT'TERM'READ    1 - waiting for a terminal read to complete
```

The detective was left in awe after meeting with the PCB and his cousin the LDTPCB. He has not been able to classify these individuals. With the help of the inspector general, he did ascertain that the "process" was alive, waiting only for a terminal read. He also recognized that if additional help with the suspects was required, INTERPOL (HP SE & CE) would need to be involved.

CONCLUSION

After a detailed investigation our detective was left with two suspects, the PDDIT and the HWDIT. Both argued that they were innocent victims. Their attorneys pointed out that the prosecution only had circumstantial evidence and suggested that both were being framed! The inspector general could not deny this and therefore authorized a "wire tap" (data communication analyzer). The wire tap confirmed the innocence of the suspects and identified the real criminal as an incorrectly configured Support Link modem. (The modem was never dropping DSR, as it should when the carrier is lost.)

TERMDSM is a valuable tool for the system manager. We have seen that through its use we are very quickly able to look at an LDEV from the operating system's viewpoint. This becomes necessary when foreign (non-HP) devices are installed as part of your data communications network. TERMDSM allows the user to quickly verify the termtype (in actual use) that has been customized with the Workstation Configurator. The system manager may even use TERMDSM to view system tables such as the PCB while looking for the cause of system performance problems. You will find that with practice TERMDSM will become a valuable tool in your shop. Your success with TERMDSM can be further enhanced by keeping a log (or diary) of problems you have seen, the effects, causes and cures. By taking DUMPs of properly working terminal and printer ports you can build a reference library for later comparison when problems arise.

Good luck and happy sleuthing...

REFERENCES

[1] TERMDSM Manual, HP part number 32033-90010, Dec 1987, pp 2-2

[2] TERMDSM Manual, pp 2-3

BIBLIOGRAPHY

[1] Fundamental Data Communications Handbook, HP part number 5957-4634

[2] Workstation Configurator Reference Manual, HP part number 30239-90001

[3] Point-to-Point Workstation I/O, HP part number 30000-90250

[4] TERMDSM Manual, HP part number 30144-90013, December 1987

[5] MPE V Tables Manual for MPE V/E, HP part number 32033-90010

[6] ATP/TIC/ADCC Tables Reference Manual, HP part number 84061-1500

[7] Scroggs, Ross, *Everything You Wanted to Know About Interfacing to the HP3000 PART I, Paper presented at the 1982 HP3000 International Users Group Conference, San Antonio, Texas*

[8] Mears, David B., User Control of Terminal Type Characteristics, Paper Presented at the 1983 HP3000 International Users Group Conference, Montreal, Quebec, Canada

[9] Scroggs, Ross, *Everything You Wanted to Know About Interfacing to the HP3000 - The Inside Story,* Paper presented at the 1983 HP3000 International Users Group Conference, Montreal, Quebec, Canada

[10] Buiteweg, Anton J.W., *From Terminal To Computer Port In 232 Easy Steps.,* Paper presented at the 1985 HP3000 International Users Group Conference, Amsterdam, The Netherlands

[11] Beetem, Jim and Smith, Catherine, *New Trends in Workstation I/O,* Paper presented at the 1984 HP3000 International Users Group Conference, Anaheim, California

BIOGRAPHY

Dennis Heidner received his BSEE degree from Montana State University, Bozeman, Montana. Mr. Heidner has written and presented numerous papers at the HP International Users Group Conferences. Mr. Heidner is a co-author of *The IMAGE/3000 Handbook* and the *TurboIMAGE Supplement,* 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 (ACM) and the Institute for Electrical and Electronic Engineers (IEEE).

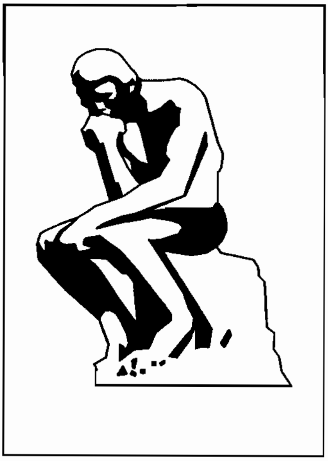
In Search of a Better Mouse Trap

Dennis Heidner
Boeing Aerospace

ABSTRACT

This paper covers basic concepts of "expert systems" and their use in business data processing. The author discusses several examples of in-house applications which have been implemented on an HP3000. The cases discussed disprove the beliefs that artificial intelligence systems must be programmed in PROLOG or LISP, and that performance is marginal on a small stack machine.

INTRODUCTION



Why Artificial Intelligence? The current demographics have been radically changing. The post World War II employees are now nearing the age of retirement. When they retire, years of expertise will leave with them. This loss of knowledge and expertise will have staggering effects on companies that are unprepared. Artificial Intelligence (AI) is an attempt to mimic the human brain with highly processed sand (computer chips) and retain valuable expertise! In their book "Artificial Intelligence: Underlying Assumptions and Basic Objectives" Nick Cercone and Gordon McCalla identify the roots for AI as being *"psychology, philosophy, linguistics, electrical engineering, and computer science"*.[1] (After dabbling in AI for several years I believe that they left off the most important root - sheer madness!) The term AI often causes a considerable debate over what really constitutes a "smart" system. Currently the field which has captured the interest of many researchers is neural nets. Research into neural networks has been as diverse as attempting to simulate neurons in software or developing special hardware which actually uses individual brain cells from slugs and snails. The more traditional areas of AI are expert systems, search and problem solving, theorem proving and logic programming, knowledge representation, learning, and miscellaneous game playing.

One exciting area of AI research is the study of new man-machine interfaces. A considerable amount of attention is now being spent on creating programs which communicate with users on the users' own terms and do not require specialized training. It is possible to buy pocket chess or backgammon games which demonstrate a high level of expertise and yet a very simple and clean man-machine interface. This should be a goal for any expert system. With many newer programs the manual is built-in as part of a very complete help subsystem. The more sophisticated programs even provide context sensitive help; the more errors and more trouble you are in, the more assistance they automatically provide for a specific area.

Expert systems are computer programs whose behavior duplicates, in some sense, the abilities of a human expert in his/her area of expertise.[2] Expert programs are considered to be a knowledge base. This knowledge base generally includes any rules of thumb (heuristics) and production rules, facts and relations, and special logic used to present the assertions and questions (the inference engine).

Production rules are similar in appearance to the conditional statements used in existing third generation languages like FORTRAN, BASIC, and COBOL. For example:

```
RULE 1: IF ANIMAL HAS FEATHERS THEN ANIMAL IS BIRD
RULE 2: IF ANIMAL FLIES AND ANIMAL LAYS EGGS THEN ANIMAL IS
        BIRD
```

Facts and relations may be implemented in production rules (as above) or contained within "frames". Frames allow the knowledge engineer to add an hierarchical structure to the knowledge, allow inheritance of traits, and provide slots for data, attributes and rules for interpreting the knowledge. The frame for a bird might look something like:

```
Frame name: Bird
Inherited from: Animal
Slot:  Skin_covering
       Type: Feathers
       Do-procedure: none
Slot:  Reproduction
       Type: Eggs
       Do-procedure: none
Slot:  Extinct_species
       Type: Dodo
       Do-procedure: Take_picture
```

The knowledge acquired by the expert is of little value unless the computer can effectively question and apply the knowledge. This is accomplished with an inference engine. The engine is the "control tower" for the expert system. It is responsible for taking the given facts, applying the assertions and deducing the conclusion. With traditional software systems, the compiler acts as an expert system interpreting the rules (source code), optimizing the object based on expert knowledge, and generating output suitable for use by the CPU. There are two very basic types of engines. These are the *forward chaining* and *backward chaining* engines.

The forward chaining engine applies the first rule to the known conditions (antecedents); if it is successful then a conclusion (consequent) is accepted. The engine then takes the consequent and looks forward for another rule which may be applied. This chaining of antecedent to consequent and consequent to antecedent continues until there are no more rules to apply or we have reached a goal (the answer). Sometimes it is possible that all rules have been applied (asserted) but a goal has not yet been reached; this is the result of either missing information or inaccurate facts.

A backward chaining engine starts with a goal and, using the known facts, it verifies that the conclusions are supported by the facts. This is accomplished by checking the antecedents in the goal for facts that match. If the facts are unknown then the current antecedent becomes a "sub-goal" (assumption) which the engine tries to prove or disprove. If there are no facts to verify the antecedent then the user is prompted for additional facts.

So far our discussion on the expert system assumes that we live in a perfect world where all facts are known or readily available. However, in the real world, we must often make a decision without all the facts. Experts are able to apply heuristics and make educated guesses for their answers. A good expert system also accommodates missing knowledge or uncertainty.

Missing knowledge or uncertainty in expert systems may be addressed in several ways. The first is to ensure that there is a redundancy of facts. This helps assure that there are fewer holes and provides alternate paths to a solution. A second method is through the use of fuzzy logic. With fuzzy logic there is no longer a simple true or false answer but instead "most likely false", "most likely true," and a large grey area between. When the expert collects the necessary facts to implement the expert system, he/she also tries to determine a default answer and the probability that it will be correct if it must be used in place of an actual fact.

We become experts through a cognitive process in which new facts and techniques are added to our existing knowledge and later reinforced as they are applied to new tasks. Very few expert systems exist which can learn through the same process. Instead, as the knowledge expert becomes aware of deficiencies in the knowledge base, he/she must adapt the knowledge base. The knowledge base for a good expert system is very dynamic. For this reason it is important to involve the end user early in the project, and prototype the system whenever possible. (This is a strong argument for PROLOG and LISP: knowledge prototyping in these languages is immensely easier than in traditional business languages.)

When developing inference engines using COBOL, BASIC, C, SPL, PASCAL or FORTRAN, it is imperative that the engine be simple and modular. Concentrate instead on developing a good user interface, and look for ways to provide redundancy of facts in your knowledge base. Doing so allows your engine to cope with missing data in an easier manner.

In this paper we will cover three different expert systems. The first is a forward chaining engine, with a discussion about how it was implemented. The second example demonstrates how several independent expert systems can be connected to provide an even smarter system. The third case is a backward chaining inference engine.

THE SEARCH

Within Boeing Aerospace we have an organization responsible for operating a resource library which contains general purpose test and measurement equipment. The customers (pool users) are the other Boeing organizations which need equipment for engineering design, manufacturing, facilities maintenance, or calibration of other equipment. When a pool user has a need for some test equipment, he/she typically either asks for a specific manufacturer and model number or requests an alternate item which can be used for the test. This search for an alternate model led to the title of this paper - "In Search of a Better Mouse Trap".

Types of test and measurement equipment range from the simple balance scale to highly sophisticated logic analyzers and computer systems. The expert system needs to be able to recommend an alternate for a logic analyzer as quickly as the solution for a scale. The design of the expert system begins by trying to replicate the thought process that an expert instrumentation engineer uses when selecting or looking for an alternate. This process begins with a specific need - "I must weigh a box 4 inches by 5 inches, with a weight between 1 and 5 pounds. The measurement must be accurate to within an ounce." The second step in the process is to identify the manufacturers of scales. The third step is to review the manufacturer literature looking for a scale which is suitable in dimensions, weight range, and accuracy. If step three fails, then a fourth step is to look for alternate weight measuring devices which might be able to perform the measurement with a slightly degraded accuracy.

An early review of our needs identified several basic requirements. First, the expert system for alternates must be very fast (we already had literature libraries and were quite adept at manually locating alternates.) Second, because our computer users varied significantly in education level and typing skills, the user interface must be friendly and easy to use. Third, we want to be able to print a catalog (wish list) containing all the models ir our inventory, with specifications, which could be distributed to our pool users. 'The last requirement is that the engine be flexible enough to support knowledge for many diverse types of test equipment.

The Mouse Trap

The most difficult task in implementing an expert system is acquiring knowledge from experts. Many experts must first visualize a problem before they can begin to solve it. How visualizing helps in resolving problems has been studied in great detail by psychologists studying information processing. Their conclusion is that the knowledge engineer must "*ask the expert to introspect about the internal processes, to report on inner experience.*"[3] Thus the knowledge engineer must elicit knowledge by asking the expert to describe what thoughts and feelings were used to reach the conclusion. The knowledge engineer must be constantly alert for buzz words which result in missing or uncertainty in the knowledge base. *"References for comparative words like "better," "easier," and "cheaper" sometimes are not qualified. If an expert states: "This is the better system," clarification is required. Appropriate responses are: "How do you know that?" "Better compared to what?" "What, specifically, is better about it?" "Better for what purpose?"*"[4]

The experts that we used to collect our knowledge database were our own internal equipment engineering technical staff. The technical staff comprises four people (including myself) with more than 100 years combined experience in the field of test and measurement equipment. Our knowledge engineers were college students on a summer break or new engineering graduates. The knowledge engineers were given a list of models for which we wanted specifications and the vendors' literature, then encouraged "pick our brains." They were told to restrict the amount of information they amassed to seven or eight of the most critical specifications (COLUMNS) for each class (NOMENCLATURE) of instruments. An early review of the selection process showed that most instruments are selected on the basis of only four or five specifications. We recognized that by restricting the number of fields to only seven or eight there would be instances in which the inference engine could not definitely recommend one model over another. In real life that same uncertainty is often present, even for an experienced instrumentation engineer. Often much more detailed research must be done in order to finally choose an alternate. Therefore, if the expert system is able to reduce the field of items to five or six from one hundred or more, it still can be considered a success!

Each family of instruments is placed into a larger super-class (GEN NOM or CHAPTER). For example BALANCE SCALES, BATHROOM SCALES, and POSTAGE SCALES are all collected into a super-class called SCALES. The knowledge engineers were also requested to identify other classes of instruments which were related or should be checked if a suitable alternate was not found in the original class. (We collected this information in the SEE NOM dataset.)

The collection of super-class (GEN-NOM), class (NOMENCLATURE), and slots (COLUMNS) form a frame, the basis of our expert system. The information for the frame is contained within four separate datasets. The first dataset, called CUSTOMIZE, is an array which contains global information and strategies to be used by the inference engine. An example of a global rule is whether or not the expert engine will allow "vendor loyalty" to be considered when looking for an alternate. The second dataset, called NOMCL-DETL, contains the class name, fields to identify the super-class and any alternate classes, and eight fields which contain pointers to the definitions for up to eight slots. The definitions for the slots, their units and associated rules are contained in a dataset called NOMH-DETL. The facts which are unique to each vendor's products are contained within a dataset called SPEC-DETL. Our frame of knowledge looks like:

```
              facts                      assertions
------------------------------------------------------------------------------------------------------------
SPEC-DETL: MODEL-CODE, X14
                 NOMEN-CODE,  I <=> NOMCL-DETL: NOMEN-CODE,   I
                 LINE-NUMBER, I                 NOMENCLATURE, X16 -- class name
                 COLUMN1,    R2                 GEN-NOM,      I   -- super-class
                 COLUMN2,    R2                 SEE-NOM,      I   -- alternate class
                 COLUMN3,    R2                 HEADING1,     I
                 COLUMN4,    R2                 HEADING2,     I
                 COLUMN5,    R2                 HEADING3,     I ==> NOMH-DETL: TAG#,      I -- rule#
                 COLUMN6,    R2                 HEADING4,     I                HEAD_NAME, X8
                 COLUMN7,    R2                 HEADING5,     I                UNITS,     X8
                 COLUMN8,    X8                 HEADING6,     I                BETTER-IF, I  ==> global
                                          HEADING7,     I                                 scoring rules
                                                                                           from CUSTOMIZE
                                                                                           dataset
```

A requirement for our expert system is that it be easy to use. A brief glance at the knowledge frame above shows that we have several complex relationships which confuse most non-data-processing personnel. It was for this reason and to ease the acquisition and implementation of the expert system that the frame was broken into the four datasets previously shown. Separate data entry routines were provided for each dataset. These routines provided the means to add, delete, or revise facts and assertions. The assertions for the frame must be entered first. The rules are assigned a unique number and may be used by many different frames. A sample dialogue for adding a new rule is:

```
Add/Delete/Revise ? ADD
ENTER TAG NO. 9999
ENTER HEADING WEIGHT
ENTER UNITS POUNDS
BETTER IF ( =<>X ) ? =
** TRANSACTION COMPLETED **
```

A dialogue revising the assertions for a BALANCE SCALE looks like:

```
Add/Delete/Revise  REV
ENTER NOMENCLATURE BALANCE SCALE
Working with nomenclature: SCALE,BALANCE
NOMENCLATURE  GEN-NOM  SEE NOM  TOTAL  COL1 COL2 COL3 COL4
SCALE,BALANCE    -120    1515       0     3    2    4    6
ENTER FIELD NAME? HEADINGS
ENTER COL 1 9999
Heading: WEIGHT     Units: POUNDS
ENTER COL 2 ____
ENTER COL 3 ____
ENTER COL 4 ____
ENTER FIELD NAME __________
** TRANSACTION COMPLETED **
```

After the rules and class have been established, it is now possible to add the actual facts into the knowledge base. The inference engine bases its decision on the lowest and highest ranges specified by the engineer. A sample dialogue for the acquisition of facts is:

```
Add/Delete/Revise ? ADD
ENTER MFG NONIN______
ENTER MODEL INTEREX______
ENTER NOMENCLATURE SCALE,BALANCE_______
ENTER DESCRIPTION It's really neat___
ENTER COST 10______
Specification line#: 1
ENTER WEIGHT IN POUNDS 1_____
ENTER Height IN inches  ______
ENTER Width IN inches  ______
ENTER Length IN inches  ______
Specification line#: 2
ENTER WEIGHT IN POUNDS  5_____
ENTER Height IN inches  ______
ENTER Width IN inches  ______
ENTER Length IN inches  ______
Specification line#: 3
ENTER WEIGHT IN POUNDS  ______
QUIT (Y/N)? Y
** TRANSACTION COMPLETED **
```

Once the knowledge has been entered it may be viewed from any terminal connected to our HP3000. This is an example of what the user sees when viewing the facts just entered.

```
ENTER MFG NONIN_____
ENTER MODEL interex___

MANUFACTURER  MODEL       DESCRIPT        NOMENCLATURE   NEWCOST
NONIN        INTEREX    It's really neat SCALE,BALANCE       $10.

#  WEIGHT   Height   Width    Length
   POUNDS   inches   inches   inches
1   1
2   5
```

It's Fuzzy

With the knowledge base assimilated, it is now possible to request the expert system identify alternate make/models. When the inference engine is used, it prompts the inquirer for a manufacturer name and specific model number. The engine retrieves the facts for the specific make/model - then by using the nomenclature code (part of the facts for the model), it connects into the NOMCL-DETL and the NOMH-DETL to load the appropriate assertions. The engine then reads the global rules stored in the CUSTOMIZE dataset so that it knows how to treat missing facts and what strategy to use. After the assertions have been loaded onto the stack, the engine begins to examine other make/models in the same class (nomenclature) as the one specified by the inquirer. Our engine uses the forward chaining technique for applying the assertions that it has loaded. The potential capability of each model is "scored" using a point system. Only the scoring results for the top one hundred make/models are saved. While examining each model the reason why each has received its high marks is saved with the score. Later when we display the results we can also display the reason why.

In the score card that we use, a perfect match is worth 100 points and is indicated as "GOOD." A near match (within 10% of the desired range) is worth 80 points and remembered as a "FAIR" match. An instrument which comes within 20% of our original instrument is considered a "POOR" match and receives only 60 points. An instrument which falls short is given no points for the specific slot unless the slot is empty because of missing information. In this case 10 points is assigned. Each slot has a multiplier associated with it. The first column, considered to be the most important, is worth five times more points then the last column. This results in the following point system:

Column	Multiplier
1	6
2	5
3	4
4	3
5	2
6	1.5
7	1

The thresholds for poor, fair, good as well as the multiplier values are stored in the global rule set (CUSTOMIZE) and can be adjusted independently of the facts in the SPEC-DETL or the assertions in the NOMCL-DETL or NOMH-DETL.

Eeek! A mouse!

The visual format for product comparisons is generally a table which lists the specifications and features for each selection. Comparisons of this type are seen in almost every magazine and many newspaper advertisements. Because this is an accepted method for comparisons with which most people are already familiar, it was the method that we chose for the inference engine's presentation of equipment alternates. The only information that the engine needs from the inquirer is a manufacturer name and the model number. For example:

```
ENTER MFG Heidner
ENTER MODEL B723c
There are 2  Heidner B723C
MANUFACTURE   MODEL        DESCRIPT          NOMENCLATURE  NEWCOST  NEWDATE
 Heidner     B723C     100Hz-22GHz      ANLYZ,SPECT     $55,555. 01/01/89
NATIONAL STOCK NUMBER:      -  -  -

#  Freq      Resoltn  Shape Ft MaxAC In In Pwr   In Accu  Swp Time Unit
   Hz        Hz       Type      dbm       dBm    +/- dB   Sec      Requires
1   100       10       15       30        -134    .6       1.0u    Stand
2   22G      3000K     11                 30      3        1500    Alone
ME, LP, NITE, QUIT? ME
```

The engine first displays the specifications of the item for which you are seeking an alternate. The inquirer is allowed to choose whether the results of the search are listed to the CRT (ME), sent to the line printer (LP),or submitted as a job to be run at night (NITE); to quit altogether, he/she enters QUIT. Choosing ME causes the following to be displayed.

```
ALT FOR         COL1      COL2      COL3      COL4      COL5      COL6      COL7
 Heidner        Freq      Resoltn   Shape Ft MaxAC In In Pwr    In Accu   Swp Time
 B723C          Hz        Hz        Type      dbm       dBm       +/- dB    Sec

Z#   MFG        MODEL        COL1  COL2  COL3  COL4  COL5  COL6  COL7
--------------------------------------------------------------------------
 1   ELKTRONIX  8M12               GOOD
 2   ELKTRONIX  8M13                     GOOD  GOOD  GOOD
 3   ELKTRONIX  8M18                     GOOD  GOOD  GOOD
 4   ELKTRONIX  892A                     GOOD  GOOD
 5   ELKTRONIX  8M5                      GOOD  GOOD
 6   ELKTRONIX  892P                     GOOD  GOOD
 7   ELKTRONIX  892                      GOOD  GOOD
 8   ELKTRONIX  894P                     GOOD  GOOD
 9   ELKTRONIX  8M14                     GOOD
Press [RETURN] to go on, ZOOM number or 'QUIT' to stop: Zoom 1
```

The engine then proceeds to display the more detailed specifications for the item chosen by the inquirer.

```
MANUFACTURER  MODEL        DESCRIPT          NOMENCLATURE  NEWCOST  NEWDATE
 ELKTRONIX  8M12       2GHz ELK8000 PI  ANLYZ,SPECT      $5,000. 01/01/73
NATIONAL STOCK NUMBER:      -  -  -

#  Freq      Resoltn  Shape Ft MaxAC In In Pwr   In Accu  Swp Time Unit
   Hz        Hz       Type      dbm       dBm    +/- dB   Sec      Requires
1   100K      300                                                  8000
2   1800M    3000K                                                 Mframe
USE WITH:8813     8803     8854
```

The software required to collect or edit the facts and assertions requires three code segments, each about 6K words. The inference engine requires two segments each approximately 8K words long. The stack size used is approximately 5K words. This is quite small compared to what would be expected for a PROLOG or LISP implementation. However, in exchange for the smaller size and faster execution, we were required to forego the extra flexibility offered by PROLOG or LISP.

THE SCROUNGER

The test equipment pool has almost one thousand customers who use tens of thousands of items spread throughout western Washington state. The equipment pool is very dynamic with equipment moving approximately every 50 days. As a resource organization we are committed to serve our customers with fastest possible response time at the lowest possible cost. This requires the assistance of master "scroungers" (like Radar O'Riley from the television program "M.A.S.H.") Radar's uncanny ability to "perceive" events about to happen provided him with an edge he needed to cope with the war, perform his routine activities, and maintain the stock of supplies for the unit. The ideal expert system "scrounger" should have many of these same attributes. The scrounger must be able to find "who's got it", ask "can I get it?", and then try to "grab it!"

Who's got it?

The location of all the test equipment in the equipment pool is contained within a TurboIMAGE database on our HP3000. With access to this database the scrounger is able to locate all items. That is the easy part! If we have 500 or 600 items of one make/model, we do not want to make 500 or 600 telephone calls while trying to negotiate a loan or swap of an item. (This is even more important if there is one sitting in a stockroom someplace!) We have solved this problem by implementing an expert system which locates the items and assigns a probability that we can borrow or loan the item out. The procedure RATEQUIP is a forward chaining inference engine that also loads rules and strategy from the CUSTOMIZE dataset. The engine examines all items of the specific make/model, checking to see if the item is currently assigned to a user, if it has been reserved for a future test, if the current schedule is about to expire, if the current user of the item has many like items, and whether or not the current user has a history of not fully using the equipment. RATEQUIP returns the asset identification numbers and the score to the procedure FINDMFGMODEL. If an inquirer wishes to see what is available, FINDMFGMODEL displays more detailed information, with the item which should be picked first at the top of the list.

Can I get it?

After a list of potential items has been created, it can then be passed on to the next expert system. This next system uses an engine called CHECKSCHEDULE. CHECKSCHEDULE takes each item and looks to see if we can "squeeze" another use in for this item. If we cannot, CHECKSCHEDULE returns an error, along with the date when the item will be available. If the item is available, then a flag is returned to indicate we can proceed to schedule this item.

Grab it!

The function which grabs the items has not yet been completed. The procedures have been tried manually and are straightforward: take an item which can be scheduled, and lock it in.

Trade?

Remember the real world? More often than not, the specific item is not available to be "grabbed". In that case, the inference engine from our specification expert system is invoked to identify other possible models which could be used. The engine SCOREMODEL returns a list of up to 100 possible alternates. Each model can, in turn, be sent back to the RATEQUIP engine which provides a list of items with the most likely to be loaned at the top of the list. CHECKSCHEDULE is invoked for each item, those which fail are dropped off the list. Finally the several remaining items are passed on to the "Grab it" process with the request that soft schedules be placed on them. ("Soft" means that a firm request for that specific model will take precedence.) The "Grab it" process is instructed to send an electronic mail message to the inquirer who requested the make/model explaining what has been done.

Will it work? The modules RATEQUIP, SCOREMODEL, CHECKSCHEDULE, FINDMFGMODEL, and SENDMAIL have been in production for several years. They are fast and trouble free. Of all the expert systems discussed so far, the "grab it" module appears to be almost trivial. The concept of coupling the engines together is not new - we "experts" currently perform related tasks by hand every day.

THE DOCTOR

Controlling the cost of software development and maintenance is a major concern of data processing managers. Software maintenance is labor-intensive work. Anything which can be done to improve the productivity of the maintainer will, in turn, reduce support costs. One technique that we have employed is extensive built-in diagnostics and debugging tools. The programs have been written so when an error is detected, information is collected which will assist the maintainer in quickly locating and correcting the problem.[5] A special program called ADPAN reads the information "snapshot" and attempts to identify where the error occurred.[6] ADPAN acts as a specialized "doctor" for software. In the process of diagnosing the error ADPAN must locate the last valid stack marker, play the part of the MPE loader and locate all subroutines which created stack markers, look for transitions between user-code and system-code, check trap Plabels, and finally check the status of all files open at the time of the snapshot. ADPAN contains several inference engines which cooperate while making the diagnosis. We will concentrate on the simplest, the process used to find the stack markers.

When we first began implementing snapshots into our programs, we used the MPE intrinsic STACKDUMP to collect and format the process stack. This originally provided us with a solution which did not require privilege mode and was supported by H-P. When a program encountered an error, it entered a procedure which opened a snapshot file, then directed STACKDUMP to copy the entire stack into it and format the program's stack markers. Later an analyst could simply use FCOPY to print the snapshot. Reading the formatted markers required some effort, but at least there was documentation on how to interpret them[7]. That was life with MPE III and early versions of MPE IV.

Then sometime around MPE IV D-MIT, several bugs were introduced with new versions of the operating system. The first was somewhat humorous (but a significant security risk). STACKDUMP allowed the calling procedure to specify a range to dump. If the range specified was beyond what the user really had - STACKDUMP should dump only up to the limits of the user's stack. The new undocumented "feature" instead dumped the entire 64K word memory page, complete with other users' data and passwords! After reporting the problem H-P provided a patch. The patch corrected the security problem; however, the stack markers were no longer formatted!

Without formatted markers the post mortem dumps appeared to be worthless, until we realized that we could manually look for the pattern which appeared to be a marker and verify it by hand-tracing back to the initial marker. This required some effort and was difficult to teach. Unfortunately the new version of MPE which would contain the correction to STACKDUMP (Q-MIT) was months away. After a little deliberation we decided to write a program which would replicate what we were doing manually. This program later became the basis for our backward chaining inference engine.

The stack for the "classic" HP3000 contains a dynamic region between "DL" and "DB"; the area above DB is called the global area and ranges from DB up to the first stack marker Qi. Every call to a subroutine (or COBOL PERFORM) causes a new stack marker to be created and added to the stack. The stack on the HP3000 appears to "grow" downward until a maximum limit of Z has been reached. The "Top-of-Stack" (TOS, shown at the bottom of the figure) is called S. The procedure's dynamic local variables are located between the last stack marker and S.

Stack markers have four special values in them. The first, located at Q, is called displacement (or delta). The next stack marker can be located by taking the current address location of Q and subtracting the displacement from it. The second word (Q-1) contains the status of the CPU at the time of the subroutine call. The third word (Q-2) is called the "P_Relative" value. P_Relative is the word location in the program to which the program will return when the current subroutine completes. The last word (Q-3) is an index register. This word can contain any value. For the experienced gurus out there, this may be quite boring; however from the neophytes, I can already hear the demands to stop. It should be now apparent why we want an expert system to assist us in locating the markers.

```
DL    +--------------+
      |              |
DB    |--------------|  = beginning of global
      |              |    data
      |              |
Qi    |  ----------  |   = initial Q "marker"
      |              |
Qi+1  |  -----------  |  = second marker
Qi+2  |  -----------  |  = third marker
Qi+n  |  -----------  |
S     |--------------|   = "Top-of-Stack"
      |              |
Z     +--------------+   = Maximum data limit
```

The method we use to find all the stack markers is:

1. Start by checking the value at S. If it is not the marker then decrement the address count by 1 and check this new value.

2. We can prove or disprove that word is Q in a stack marker by applying the following rules.

   A. The Delta Q must always be a positive number greater than 4 but less than the address location of Q.

   B. Check P_Relative. It must range between 0 and 16384.

   C. The value of STATUS (Q-1) may not be 0.

   D. The value pointed to by the address of Q minus the displacement must be a stack marker.

   E. If Q=4 and P_Relative=0, we are at Qi and have located all the stack markers.

We can more clearly see why this inference engine which locates the markers is considered to be backward chaining by carefully looking at rule 2D. This requires we not pass judgement on the starting value we are examining until we have chained backward and verified each "sub-goal". The FORTRAN source code which is used to implement the inference engine is listed next.

```
      SUBROUTINE FIND MARKERS(Q,IDUMPFILE,START,END,LASTREC,DB OFFSET,S,INBUF,IERR,MPE V)
C
C     The purpose of this routine is to try to find the stack markers in the dump file, when the marker
C       display has been damaged.
C
C     Error codes returned are:
C     IERR=0  - OK
C     IERR=-1 - Problems encountered trying to read dumpfile
C     IERR=1  - Unable to locate a good marker!
C
      INTEGER*4 LASTREC,START,END
      INTEGER Q,S,IDUMPFILE,IERR,Q TEST,DBOFFSET
      LOGICAL INBUF(128),MPE V
C
C     The algorithm used to find a valid marker is as follows.
C
C     1.  Set the absolute maximum number are words we will try to 2048.
C     2.  Set  Q TEST = S
      Q TEST = S
C
C     3.  VERIFY MARKER using the value of Q TEST. If ok then IERR = 0, and Q = Q TEST, then RETURN.
C         Else....
C
      DO 100 I=1,2048
      CALL VERIFY MARKER(Q TEST,IDUMPFILE,START,END,LASTREC,DB OFFSET,S,INBUF,IERR,MPE V)
      IF(IERR.EQ.0) GOTO 200
C
C     4.  Decrement Q TEST.
C
      Q TEST = Q TEST - 1
C
C     5.  Proceed to step 3.
C
100   CONTINUE
      IERR=-9
      RETURN
C
C     WE FOUND A GOOD MARKER
C
200   Q = Q TEST
      RETURN
      END
      SUBROUTINE VERIFY MARKER(Q,IDUMPFILE,START,END,LASTREC,DB OFFSET,S,INBUF,IERR,MPE V)
C
C     The purpose of this routine is to take a given value
C      for Q and prove or disprove that it is a stack marker.
C
C     Error codes returned are:
C     IERR = 0    Okay. Hypothesis proven
C     IERR = 1    Bad marker, Hypothesis is false
C     IERR =-1    Problems encountered in GETWORD, Hypothesis false
C
      INTEGER*4 START,END,LASTREC
      INTEGER DB OFFSET,Q,IDUMPFILE,IERR,S,Q TEST,P RELATIVE,DELTA Q
      INTEGER STATUS
      LOGICAL INBUF(128),MPE V
 C
       Q TEST = Q
 C     The rules for checking for a valid stack marker are:
 C
 C     1.  Take the value of Q it must fall between DB OFFSET and S.
 C         If not then IERR=1 and return.  If ok proceed...
 100   IERR=1
       IF((Q TEST.LT.DB OFFSET).OR.(Q TEST.GT.S)) RETURN
 C
 C     2.  GETWORD at address of Q. If error encountered in GETWORD then IERR=-1, and return
 C         If ok  proceed....
 C
       IERR=-1
       CALL GETWORD(Q TEST,DELTA Q,IDUMPFILE,START,END,LASTREC,DB OFFSET,INBUF,NERR)
       IF(NERR.NE.0) RETURN
 C
 C     3.  Check value at address of Q, it must be in range of 4 to address of Q.
 C         If not then IERR=1 and return. If okay proceed...
 C
       IERR=1
       IF((DELTA Q .LT.4).OR.(DELTA Q.GT.Q TEST)) RETURN
 C
 C     4.  GETWORD at Q-2.  This is P'RELATIVE.   The value for P'RELATIVE must be
 C    between 0 and 16384.  If not then IERR=1, and return.  Else proceed....
 C
       IERR=-1
       CALL GETWORD(Q TEST-2,P RELATIVE,IDUMPFILE,START,END,LASTREC,DB OFFSET,INBUF,NERR)
       IF(NERR.NE.0) RETURN
       IERR=1
       MPE V = .FALSE.
       IF ( P RELATIVE .EQ. %40000 ) MPE V = .TRUE.
       P RELATIVE [0:2] = 0
 C
 C    4B. GETWORD at Q-1.  This is STATUS. The value for STATUS must not be zero!  If it is then IERR=1,
 C        and return.  Else proceed....
 C
       IERR=-1
       CALL GETWORD(Q TEST-1,STATUS,IDUMPFILE,START,END,LASTREC,DB OFFSET,INBUF,NERR)
       IF(NERR.NE.0) RETURN
       IF ( STATUS[8:8] .EQ. 0) RETURN
 C
 C     5.  Save value at Q   Derive  deltaQ. Take Q and subtract value  at Q from Q.
 C
       Q TEST = Q TEST - DELTA Q
 C
 C     6.  If value of Q = 4  and  P'RELATIVE = 0  then  IERR=0 and RETURN!   (WE HAVE FOUND Qintial).
 C
       IERR=0
       IF((DELTA Q .EQ. 4).AND.(P RELATIVE.EQ.0)) RETURN
 C
 C     7.  Use the value of the Q - delta Q and a new address for Q.  Then proceed to step 1.
 C
       GOTO 100
       END
```

CONCLUSION

Dr. David Hu characterizes the expert system as:

> "* *An expert system mimics experts or specialists in a specific field - for example, medicine or computer configuration.*
>
> * *The power of an expert system lies in knowledge and how it is represented, not in programming technique.*
>
> * *The principal components of current systems are knowledge base, inference engine, and man-machine interface.*
>
> * *The knowledge base contains facts and rules that embody an expert's expertise.*
>
> * *The three commonly used methods for encoding facts and relationships that constitute knowledge are rules, frames, and logical expressions.*
>
> * *Inference engines are relatively simple. The most commonly used methods are backward chaining and forward chaining.*
>
> * *User interface is a weak but critical element of expert systems. Many current expert systems are equipped with "menus" and explanation modules to allow users to query expert systems and examine their output statements"[8]*

We have seen three examples of applications which are expert systems in their narrow field. These systems are been written in the traditional third generation languages of FORTRAN and SPL. They are fast, use little stack and code space, yet provide functions which otherwise would consume an expert's time. In the first two applications the database management system is TurboIMAGE and the third is accomplished using ordinary MPE file I/O.

REFERENCES

[1] Cercone, Nick and McCalla, Gordon, *Artificial Intelligence: Underlying Assumptions and Basic Objectives, from Journal of the American Society for Information Science,* Volume 35, Number 5 September 1984 pp 280-290

[2] ibid.

[3] Evanson, Steven E., *How to TALK to an EXPERT, AI EXPERT,* February 1988 pages 36-41

[4] ibid.

[5] Heidner, Dennis L., *The Bug Stops Here,* Paper presented at the 1987 HP3000 International Users Group Conference, Las Vegas, Nevada

[6] ADPAN is in the public domain contributed software library available from INTEREX, 680 Almanor Ave. P.O. Box 3439, Sunnyvale, California, 94088-3489

[7] Hewlett-Packard, *MPE Debug/Stack Dump reference Manual*, part number 30000-90012

[8] Hu, David, *Programmer's ReferenceGuide to ExpertSystems*, (Indianapolis, IN: Howard W. Sams & Co., 1987) page 10

BIOGRAPHY

Dennis Heidner received his BSEE degree from Montana State University, Bozeman, Montana. Mr. Heidner has written and presented numerous papers at the HP International User Group Conferences. Mr. Heidner is a co-author of *The IMAGE/3000 Handbook* and the *TurboIMAGE Supplement* 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 (ACM) and the Institute for Electrical and Electronic Engineers (IEEE).

Data Structures: "The KEY to Performance"

- a tutorial -

by David G. Robinson

PowerSpec
International

Data Structures: "The KEY to Performance"

Abstract

PowerHouse
Data Structures: "The Key to Performance"

There are many factors which play an integral part in the overall performance of an application. The objective of this tutorial is to identify why data structures are the "KEY" to efficient and well tuned applications. In order to do this design criteria will be introduced and discussed for optimizing your structures.

Some of the criteria to be presented are:

- Logical Data Base Design
  - . Essential Systems Analysis
  - . Data Modeling
  - . Normalization

- Physical Data Base Design
  - . Image DBMS vs Indexed Files
  - . Blocksize & Capacites
  - . Paths/Keys
  - . Application Considerations

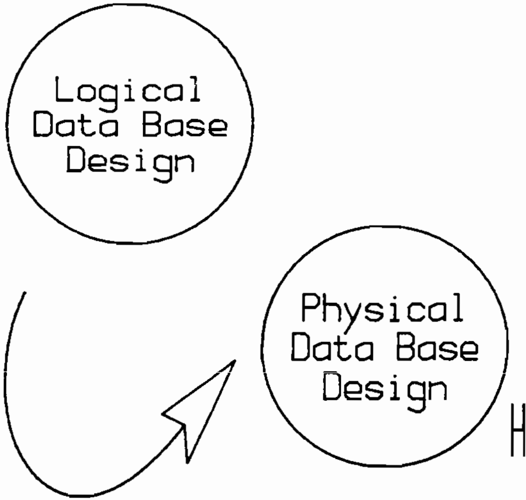
- Prototyping Application using 4GL PowerHouse

This design criteria introduced in this tutorial can be used for applications developed in other 3GL or 4GLs.

Data Structures: "The KEY to Performance"

First, ...

Define ...
What is the
Data ?



Logical
Data Base
Design

Physical
Data Base
Design

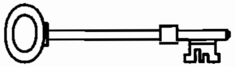
.... Next,

How to store the
Data ?

Data Structures: "The KEY to Performance"

Logical Data Base Design

Analysis Phase

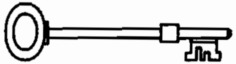


- Formal methodology
  - Essential Systems Analysis (ESA) [Top-Down Approach]
    - Data Flow Diagram (DFD)
    - Mini specifications
    - Data Dictionary
  - Blitz

- Data Modeling
  - Entity relationships
    - Entity Relationship Diagrams (ERD)
      - 1 to 1
      - 1 to n

* 4GLs are perfect fit for Data Modeling

Data Structures: "The KEY to Performance"



Logical Data Base Design

Analysis Phase

- Informal methodology
  - Identifying data entities and their attributes
    - Normalizing data structures

      " Does each data element depend on its key, the whole key and nothing but the key ?"

    - Boyce-Codd Normal Form (BCNF)

      " Every determinant (attribute) must be a candidate key of the relation (entity) "

Data Structures: "The KEY to Performance"

CASE Tools - EXCELERATOR

- . data analysis
  - . automates
  - . validates

LDBD

PDBD

PHLEX by Cognos

. PHDict Entities

Link to

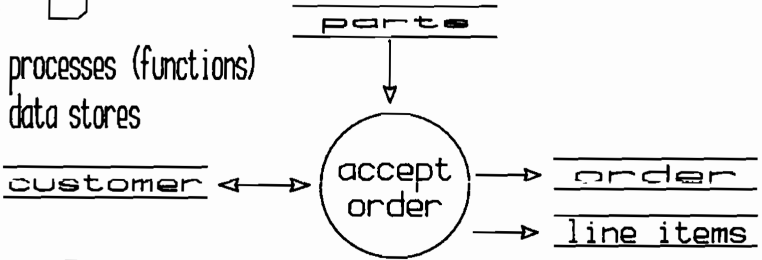
EXCELERATOR by Index Technology Corp.

Data Structures: "The KEY to Performance"

DFD

. Identify processes (functions)

. Identify data stores



Mini Specs

accept order

. For each line-item

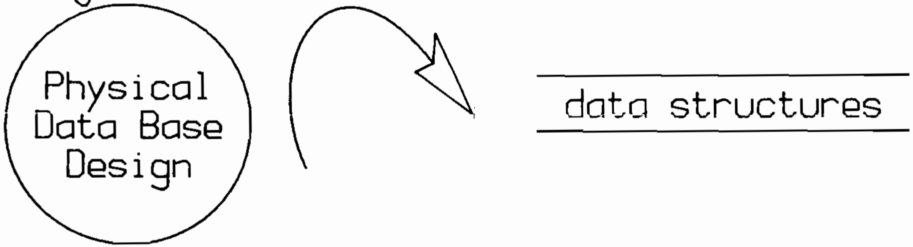
. multiple qty * price = extension

Data Dictionary

customer = [cust id, name, address, current balance]

Data Structures: "The KEY to Performance"

Design Phase



Physical Data Base Design

data structures

What data structures are supported ?

HP - MPE/Indexed/Image DBMS

DEC - Indexed/RMS/R db

DG - Infos

Data Structures: "The KEY to Performance"

"Define data structures with PowerHouse in mind ..."

Design Criteria

- Data Base Structure
  - Image DBMS or Indexed (KSAM)
- Number of Paths/Keys
- Physical Data Structures
  - Blocking Factors
  - Capacities
- Application Considerations
- Other Software Alternatives

Data Structures: "The KEY to Performance"

" A database models the dynamic behavior of its entities and their attributes by means of entries"

Alfredo Rego

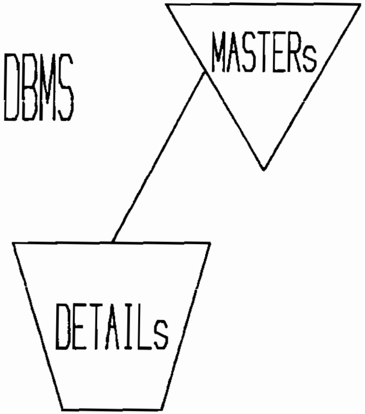
. HP Image DBMS "Network Structure" with two levels of hierarchy
    . Masters
    . Details

What are the advantages of using Image ?
disadvantages ?

Data Structures: "The KEY to Performance"

How can I access records in Image ?

Image DBMS

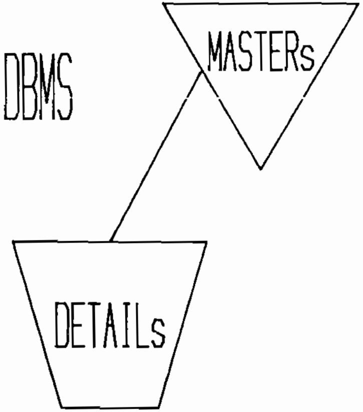


- Unique KEY Masters
  - "Hashing" - contents-oriented access method
- Relationships for these masters stored in Details
  - "Chaining" - contents-oriented access method
- Image DBMS
  - Also supports address-oriented access methods
    - Serial
    - Directed

Data Structures: "The KEY to Performance"

Advantages of using Image DBMS as data structures !

Image DBMS

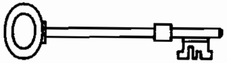


- Referential Data Integrity
- Backup & Recovery Features
  - Transaction logging
    - STARTLOG & STOPLOG for PowerHouse
- Image DBMS Utilities
- Third party Utilities
  - Adager/DBGeneral

Image DBMS

Data Structures: "The KEY to Performance"

MASTERs



Design Criteria for Masters !

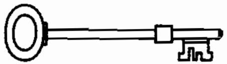
- . Optimal Block Size (BLOCKMAX)
  - . REBLOCK feature of ADAGER or DBGENERAL
- . Set capacities to odd number
  - . Improve "hashing algorithims"

- . Keys
  - . Types of X,U, and Z reduce migrating secondaries
  - . For numeric fields < 5 digits declare as J1
    - . PowerHouse type INTEGER size 2
  - * . For numeric fields < 10 digits declare as J2
    - . PowerHouse type INTEGER size 4
  - . For numeric fields > 10 declare as PACKED size n

Image DBMS

Data Structures: "The KEY to Performance"

MASTERs



* J2 Master Keys

" ... If value does not exceed the capacity ..."

" ... Then record# of entry will equal to its KEY value ... "

Results

. Master set will contain no synonyms
. Disc space (sectors) will not be wasted
. Entries can be batch loaded faster than hash entries
. Physically order the entries in any desired sequence

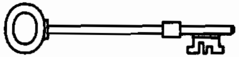
Mark Trasko,
"The Future of Database Technology"
Supergroup Association Magazine, July 1986

Image DBMS

DETAILs

Data Structures: "The KEY to Performance"

" Poor Master set performance ... affect Details"



Design Criteria for Details !

. Optimal Block Size (BLOCKMAX)

. Avoid more than two paths

. Avoid sorted paths if at all possible

* . Determine optimal Primary Path

Data Structures: "The KEY to Performance"

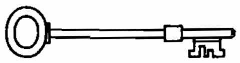
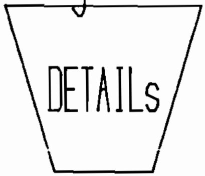
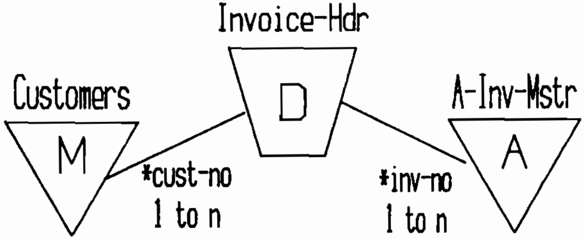


Image DBMS



" Primary Path .... increase/decrease retrieval access ... "

" Select most frequent PATH with more than one entry point "

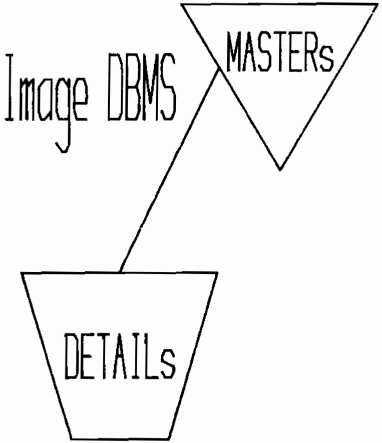


... Primary Path should be *cust-no in this example ...

Data Structures: "The KEY to Performance"

Disadvantages of using Image DBMS data structures !

" The Three Bears of Image"
Fred White



. Papa Bear	- Integer Keys
. Mama Bear	- Sorted Paths
. Baby Bear	- No. of Paths

" ... use the above with considerable thought ... "

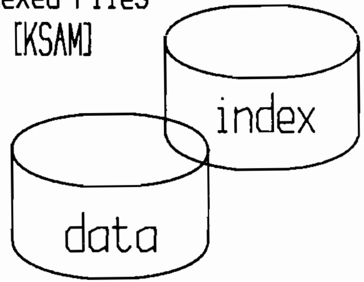
. Image DBMS allocates all necessary disc space "upfront"

. Without third party utilities can be time consuming for
  . file reorgs, etc ...

Data Structures: "The KEY to Performance"

How can I access records for Indexed Files !

Indexed Files
[KSAM]

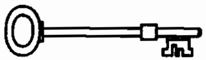


. KEY access

. Partial KEY (Generic KEY Retrieval) access

. Serial

. Directed

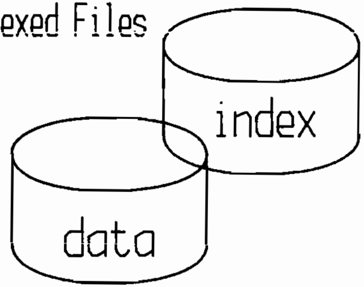


. PowerHouse products allow generic retrievals for QUIZ & QTP & QUICK

Data Structures: "The KEY to Performance"

The good and bad of using Indexed data structures !

Indexed Files



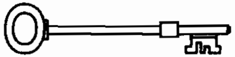
" The good "

- . more flexibilty
  - . data structure easier to change
  - . generic retrieval for character keys
- . less disc resources
  - . data file allocated in extents

" The bad "

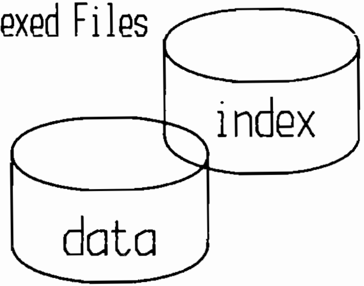
- . increase memory requirements in multi-user environment
- . no built in referential data integrity like Image Masters to Details

Data Structures: "The KEY to Performance"



Design Criteria for Indexed Data Structures !

Indexed Files

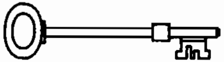


" ... referential data integrity ... "

- Data integrity can be enhanced using PowerHouse
  - Master to Detail type relationships
    - Lookups
    - Linkage of screens
      - Passing/Receiving file
    - Append mode processing
      - Detail file

Data Structures: "The KEY to Performance"

Application Considerations



- KEYS (Paths)
  - Number of keys
    - More keys increase inquiry capabilities
    - More keys also increase I/O in Updates
  - Concatenated keys (Indexed Files)
    - Retrieval of records in ascending order (key = [claim no + diary date])
- Security
  - May require additional data structures
    - QUICK Screen functional Menus
    - Password Security by ID
- Audits
  - Additional files to track (audit) changes, etc

Design Criteria for QUICK Screens !

- Mirror data base design
  - For each Master a screen
  - For each Detail a screen
    - Same for multiple indexed files
      - Build in referential data integrity

- Exception is usage of Append Mode Processing
  - Detail Files
    - 1 to n relationship for two files on single screen

- Incorporate Locking Strategies
  - For multi-user environment
    - Screen LOCK Base/File or LOCK/UNLOCK Verbs

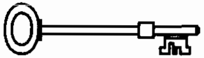
- Closing Files Explicitly

Data Structures: "The KEY to Performance"

- Develop/prototype application using Indexed structures
  - Easier to change structures

- Specify optimal blocking factor for data structures
  - MPEX   -   ALTFILE file-set,BLKFACT=BEST -

- Once data base design is accepted can change to Image

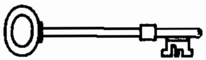


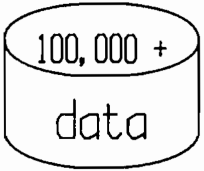
  - Large Blocking Factors for batch processing
    - QUIZ and QTP serial reads

  - Small Blocking Factors for screen processing
    - QUICK record retrieval

Other Software Alternatives

OMNIDEX / OMNIQUIZ by D.I.S.C



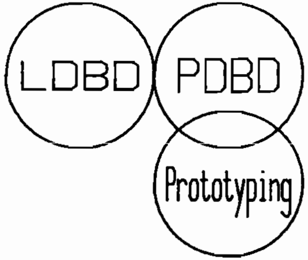


QUICK & QUIZ & QTP

- Advantage to end-user
  - Allows search on non-key fields, etc
- Advantage to data base administrator
  - Change search criterie without reorgs
    - Change only OMNIDEX structures

Data Structures: "The KEY to Performance"

" Prototyping Application"



Architect --> Cognos, Inc.

LL'Spirit --> Singapore Computer Systems

" Is feasible in a 4GL environment "

Data Structures: "The KEY to Performance"

Advantages of Prototying in 4GL !

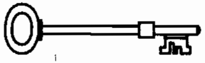
Prototyping

- Confirms users specifications (LDBD)

- Encourages users participation
  - validating inputs and outputs of the system
  - determine data base design is efficient or lack of
  - easier to evaluate then written specifications

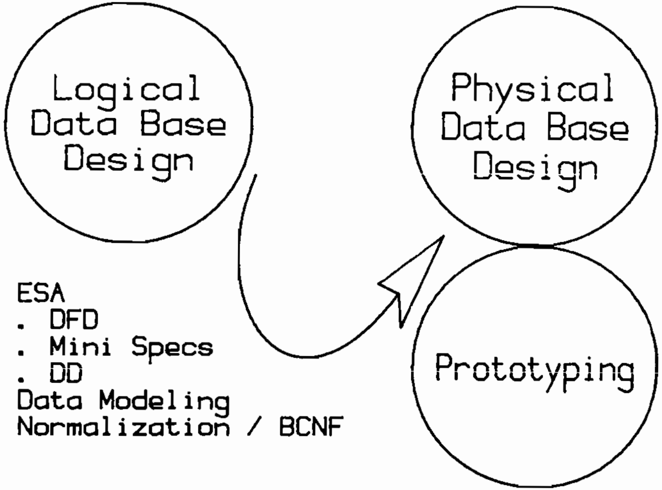
- Shortens development cycle
  - Staff could be developing other modules at same time

- Prototype can be done on Micro computers using PowerHouse PC
  - After acceptance; code transportable to mini (HP, VAX, DG)



" committment from DP staff and end-users "

Data Structures: "The KEY to Performance"

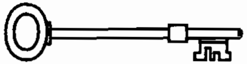


- ESA
  - DFD
  - Mini Specs
  - DD
- Data Modeling
- Normalization / BCNF

Design Criteria !

- Image DBMS vs Indexed Files
- Optimizing Data Structures
  - Blocking Factors
  - No. of Paths/Keys
  - File Capacities
  - Primary Path
  - Sorted Paths
- Application Considerations in QUICK
  - Locking Strategies

Prototyping is a viable solution !



" Data structures are KEYS to PowerHouse Performance "

Data Structures: "The KEY to Performance"

Author:

David G. Robinson is considered one of the leading authorities on PowerHouse software. He is founder and general partner of PowerSpec International the world's premier PowerHouse training center. David is an active member in the user community having presented many techical papers on PowerHouse in Europe and the United States.

He is also co-editor of TNT, a quartlerly publication of Tips, News, and Techniques on PowerHouse, which is distributed world wide.

EFFECTIVE BACKUP STRATEGIES FOR THE HP3000

Bud Beamguard
Syntex Corporation
3401 Hillview Avenue
Palo Alto, CA 94304

Let us start out with a thought:

"Risk is a theoretical concept until you have been burned."

A backup is the transferal of data from a more volatile medium to a less volatile medium, in order to provide a temporary second copy of the data offline. On the HP3000, this usually means storing off the contents of discs to magnetic tape, using SYSDUMP, STORE, DESTORE, etc. With SYSDUMP a complete copy of the system can be made.

We do backups of various kinds in order to preserve the integrity of the system:

1. The System Directory
2. System I/O configuration, system tables and parameters
3. System software: FOS and subsystems from HP
4. Third-party packages
5. In-house software
6. User data

Several additional reasons why backups are done:

1. Because HP tells you to
2. Legal liability
3. Government regulations
4. Loss of financial records
5. Non-reconstructable data, e.g., data from real-time instruments
6. Corporate/governmental auditing requirements

There are plenty of things that can happen to a system, both at the hardware and software levels:

1. Disc head crashes and other disc problems
2. Corrupt system directory
3. Volume table destroyed, must perform RELOAD
4. Inappropriate use of PM capability resulting in carnage
5. Not enough free space on LDEV 1 during load

6. Sabotage and security breaches
7. Fire, water, acid, etc. in computer room
8. Earthquakes, tornadoes, floods, meteor strikes, lightning, war, the End of Civilization As We Know It, etc.

Any of these can develop into a recovery situation, i.e., situations where data (or even the system itself) must be recovered from previously prepared backup tapes.

By their nature, emergencies are unforeseen. For purposes of developing a backup strategy, it is not important what causes emergencies: software, hardware, or human error. Rather we attempt to be prepared as best we can WHEN and IF they happen. It cannot be emphasized enough that no universal solution exists, particularly where extensive Turbo-IMAGE databases are in use.

Our goal is to be able to:

1. Bring MPE back up with as little delay as possible (Obvious? How are you going to get anything done with a dead system? How long will it take to recreate MPE from you last MPE upgrade tapes?)
2. Bring back the accounting structure/system directory/UDC structure/passwords
   (How long will it take to reconstruct all of this by hand? How complete is the documentation?)
3. Recover the system I/O configuration, system tables, system operating environment
4. Recovery software (MPE, subsystems, user-developed)
5. Recover user data
   User data gets the most attention in discussions of backup strategies, but it can be useless without a coherent system to run it on. Applications are often highly dependent upon a particular system environment.

Although backup tapes are frequently useful for recovering files which have been accidentally purged or otherwise ruined during day-to-day operations, this function is secondary. Backups should not be approached as archival operations, either.

Ideally, we should be able to restore the system to the exact state it was at the time of the outage. Even more ideally, we should be able to do this on a completely different hardware setup, should the old machine be physically destroyed. Usually we have to settle for a system as it was during the most recent backup.

A backup strategy goes hand-in-hand with a disaster recovery plan. A disaster recovery plan will clarify your needs and goals based on YOUR specific environment, and make you aware of the often cold realities of recovery operations. With a good set of backup tapes and a reasonable level of technical competence, you can recover from an emergency and come out looking like a genius. The exact opposite is also possible if you are not prepared! Develop a backup strategy that will provide you with the means to approach an emergency with confidence. Draw up a comprehensive recovery plan. A routine reload of the system, done to repack disc files, is an excellent opportunity to test such a recovery plan and to eliminate the glitches which occur during reloads. If you feel up to it, get out your DUS tape and practice loading and using SADUTIL. SADUTIL can be a life-saver in that it is able, in some cases, to rescue files which would otherwise be lost; sometimes there are vital files for which a timely backup copy does not exist.

A few general considerations about backups:

1. Backups are a form of insurance.
   Unless you intend to go naked, you are going to have to get some of this insurance.

2. Backups are an attempt to keep risk within acceptable limits. They DO NOT eliminate risk. Even tapes have their problems and limitations.

3. Like insurance, backups cost money. The cost is directly proportional to the coverage obtained.

4. Remember that system downtime is a major cost factor.

5. Only YOU know the acceptable level of risk in your situation.

6. The acceptable level of risk is going to be different for every set of files or databases on the system.

7. You are going to have to balance cost versus coverage when deciding on a backup strategy.

8. Backups must be a design requirement in any application system.

9. Be very careful about promising more than you can deliver. Never underestimate the potential complexities of a recovery, and never overestimate your technical abilities.

Backups require downtime, or at the minimum intervals during which applications and data are not available for use. It is this non-availability which is the major problem for most system managers.

Several suggestions to deal with the downtime dilemma:

1. Make sure that upper management understands what backups are and why they are necessary. It is your job to help them appreciate this necessity.

2. Make certain that backups occur on an iron-rigid schedule. Users can get used to anything as long as you are punctual and consistent. Discipline on your part helps them to utilize downtime for other purposes, to best advantage.

3. See to it that backups are included during the requirements phase of system/application planning. Beware of situations where backups are unfeasible due to uptime demands, because this is a no-win situation for you.

4. Try to present backups in a positive manner, not as a waste of time and money.

5. What goes up (MPE) must come down: nobody ever promised either a fail-safe operating system or a perpetual-motion hardware setup. Make sure that everybody realizes this. Sometimes people expect reliability levels out of a computer which they would never ask from a car or airplane. Hasten to correct these delusions.

SYSDUMP

Every system manager is familiar with SYSDUMP. An understanding of its operation can provide ideas for an effective backup strategy.

SYSDUMP performs the following steps (in order):

1. Asks ANY CHANGES?

2. Asks for dump date.
   Will backup all files created or modified ON or SINCE this date, on the basis of their file labels.

3. Asks for dump file set.
   Identical with a STORE command parameter string.

4. Asks whether to do a dump list.

5. Dumps the System Directory, the system SL, the I/O configuration, the FOS files from PUB.SYS, etc. onto the tape.

6. Activates STORE as a son process, which carries out the remaining three steps:

7. Searches the System Directory and disc file labels for files matching the date and file-set parameters specified.
   Prepares a list of these in a temp file called GOOD.

8. Stores the files listed in GOOD to tape.

9. If specified, prepares a printed list of the files, using GOOD.

Here is an example of a job stream to perform a full backup:

```
!JOB SYSDUMP,MANAGER.SYS,OPERATOR;HIPRI;outclass=lp,4,1
!COMMENT This jobstream does a full backup of the system.
!FILE T;DEV=TAPE
!FILE DUMPLIST;DEV=LP,4,1
!SYSDUMP *T,*DUMPLIST
NO                         << any changes >>
0                          << dump date >>
@.PUB.SYS,&                << dump file set >>
@.@.SYS-@.PUB.SYS,&        << dump file set >>
@.@.@-@.@.SYS;&            << dump file set >>
FILES=24000;&              << dump file set >>
PROGRESS=1                 << progress message every 1 minutes >>
YES                        << list files dumped >>
!EOJ
```

The dump file set specification in this job stream will cause the contents of PUB.SYS to be dumped first, then the SYS account minus PUB.SYS, then everything else minus the SYS account. All files are dumped only once. The tape set thus created allows you to restore "first things first" during a recovery or a reload. For example, COMMAND.PUB.SYS is extremely important because it contains the UDC structure of the system, and can be restored quickly since it is toward the beginning of the first tape reel. Similarly, the PUB.SYS group can be restored and a measure of order restored to the system environment before the user accounts are recovered. It may be possible to release certain applications even while the recovery is still in progress.

Several frills to add to your backup jobstreams:

1. STARTCACHE commands

2. Do a run of BULDACCT.PBU.TELESUP. This program will create three jobstreams, JOBACCT, JOBACCTB, and JOBCUDC. These streams can be used to recreate the System Directory. They contain ALL your passwords!

3. LIMIT and JOBFENCE commands to keep stray users off the system.

4. Tape validations using VALIDATE or FCOPY. (These take time, but may be worth it.)

An enhanced version of a full backup:

```
!JOB SYSDUMP,MANAGER.SYS,OPERATOR;HIPRI;outclass=lp,4,1
!COMMENT*****************************
!COMMENT This jobstream does a full backup of the system.
!COMMENT*****************************
!LIMIT 0,0
!JOBFENCE 14
!COMMENT*****************************
!COMMENT Make sure disc caching is turned on.
!CONTINUE
!STARTCACHE 1
!CONTINUE
!STARTCACHE 2
!CONTINUE
!STARTCACHE 3
!CONTINUE
!STARTCACHE 4
!COMMENT*****************************
!COMMENT Execute BULDACCT
!PURGE JOBACCT.OPERATOR.SYS
!PURGE JOBACCTB.OPERATOR.SYS
!PURGE JOBCUDC.OPERATOR.SYS
!RUN BULDACCT.PUB.TELESUP
!COMMENT*****************************
!SHOWTIME
!COMMENT*****************************
!FILE T;DEV=TAPE
!FILE DUMPLIST;DEV=LP,4,1
!SYSDUMP *T,*DUMPLIST
NO                       << any changes >>
0                        << dump date >>
@.PUB.SYS,&              << dump file set >>
@.@.SYS-@.PUB.SYS,&      << dump file set >>
@.@.@-@.@.SYS;&          << dump file set >>
FILES=24000;&            << dump file set >>
PROGRESS=1               << progress message every 1 minutes >>
YES                      << list files dumped >>
!COMMENT*****************************
```

```
!PURGE JOBACCT.OPERATOR.SYS
!PURGE JOBACCTB.OPERATOR.SYS
!PURGE JOBCUDC.OPERATOR.SYS
!LIMIT 5,45
!JOBFENCE 2
!COMMENT*******************************
!SHOWTIME
!COMMENT*******************************
!COMMENT validates the tape(s) using VALIDATE.PUB.TELESUP
!RUN VALIDATE.PUB.TELESUP
N    << PRINT THE TAPE DIRECTORY ? >>
N    << PRINT THE FILE LABEL INFO ? >>
Y    << VALIDATE THE ENTIRE TAPE ? >>
N    << PRINT LIST ON LINE PRINTER (LP) ? >>
!COMMENT*******************************
!SHOWTIME
!COMMENT*******************************
!EOJ
```

A widely used backup strategy is to use SYSDUMP to perform a full system backup once a week (often on weekends), with a partial backup on each working day thereafter, using the date of the full backup as the dump date of the partial; all files modified on or after that date are then dumped. This strategy is so popular that the commands FULLBACKUP and PARTBACKUP were recently added to MPE to facilitate operations. This is an excellent approach to backups: the tape sets thus created make recoveries quick and straight-forward. If downtime and operator time are not a problem, this is certainly a recommended backup strategy.

Nevertheless, this strategy contains plenty of redundancy (i.e., overkill):

1. MPE, the System Directory, I/O configuration, etc. are repeatedly backed up, perhaps needlessly should the system be static.

2. The "@.@.@" fileset specification will cause the machine to examine EVERY FILE LABEL ON THE SYSTEM, which takes plenty of time; again perhaps needlessly.

3. A great many possibly unimportant files will get backed up, usually several times; this takes more downtime.

4. A large number of backup tapes are generated, with resulting storage/security problems.

5. FULLBACKUP and PARTBACKUP use an unqualified "@.@.@" as the fileset, creating tapes which may be clumsy to use in a recovery. It is usually better to use SYSDUMP and fix up your own sequence.

If your site is under pressure to keep downtime to a minimum, there are alternatives which may or may not be appropriate for your situation.

1. Perform partials based on the date of the previous partial (or full, whichever is more recent). This will tend to reduce redundancy and thus shorten backup time. The downside of this approach is that during a recovery situation a RESTORE must be run on each of the partials, in correct order by date; an error-prone and time-consuming process.

2. Extend the time between full backups, e.g., to the first of each month. This may be a good plan in situations where the same set of user files gets modified and hence backed up day after day, while the system environment is stable.

3. In situations where the MPE environment is stable, consider the following strategy:

   a. Prepare a SYSDUMP tape containing @.PUB.SYS, based on a dumpdate of 0. This tape can be used to recover MPE and its environment.

   b. Use STORE to perform full and partial backups.

   The redundant backup of the MPE environment is eliminated.

4. Establish an Account structure in which active or critical files are kept in special groups by themselves. Examples would be major database, or a source code library. During the backup use a dump file subset which contains only these selected groups. The amount of time consumed by the directory search will be greatly reduced. This is at some risk to files NOT in these groups (which may be tolerable).

5. Sometimes a TurboIMAGE database contains datasets which are extremely volatile, and other datasets which are static. Consider breaking off the volatile datasets into a second separate database. In this manner the static sets will not be backed up merely because of activity in the volatile datasets.

6. If there are several large, unrelated applications all on the same system, consider dividing the applications among two or more networked (new) machines, thus reducing the overall time that the applications are

unavailable. This approach is especially good if one of them "just has to be up", since the others will not be exposed by its uptime demands.

7. Remember that an old, slow, antique tape drive costs lots of money in downtime and in operator time. Use this fact to cost-justify a new drive. Your HP Sales Rep will be only too glad to help.

8. Use one of the third-party data-compression packages that are available. This can greatly shorten your backups. But before you buy, be sure to test your ability to recover your system with the resulting tapes, preferably by doing an actual full RELOAD. Do not buy unless you are happy, since it is not worth the worry. Find out what HP has to say about the package.

A few considerations about backup tapes:

1. Store your tapes offsite or in another building. Obviously do not keep them in the computer room, where they could go up in smoke along with the machine.

2. The best place to keep tapes is in a fire-proof vault with a lock, even though vaults are very costly and there never seems to be enough room in them.

3. Remember that SYSDUMP tapes contain the entire System Directory, and hence ALL your passwords. If you use BULDACCT, the passwords are even easier to find since they are contained in the JOBACCT, JOBACCTB, and JOBCUDC jobs. Keep the tapes in a secure place (like a vault).

4. Tapes do wear out. Replace them every three years or so.

5. Data stored on tapes can fade away magnetically after a period of years. Use TDTCOPY or TAPECOPY (in TELESUP) to create new copies.

6. The current "live" set of backup tapes (full and partials) should be kept in a definite, easily found place where they can be retrieved quickly. Do not mix them with other tapes. You may have to tell people where to find them -- over the telephone.

7. Use good-quality tapes for your backups.

8. Have a retention schedule for backup tapes.

Discs which contain lots of dead wood are obviously going to take plenty of backup time, especially over the long haul:

1. Do not unwittingly use your discs as an archive. Insist that inactive data be removed to TAPES. Get rid of seldom-referenced accounts: TELESUP and the CSL accounts are often in this class. If certain programs are used from these accounts, copy them to special groups in SYS.

2. Make sure that the application people are not creating huge TurboIMAGE databases which remain mostly empty. Sometimes dataset sizes are specified on the basis of what might happen years down the road. During the meantime YOU are going to have to backup all that empty space.

3. Do not assume that your users have an understanding of what disc space means. Watch out for people who create monster files without realizing what they are doing. Consider putting limits on sector usage.

4. REPORT will tell you who is eating up your disc space. If you do not know why, find out.

A few suggestions to keep you prepared for a recovery:

1. Retain the System Coldload Tape and the Diagnostic Utility System tape created during the most recent MPE upgrade, as instructed by Hewlett-Packard.

2. Using SYSDUMP, create a separate coldload tape with a dumpdate of 0 and a fileset of @.PUB.SYS.
   If it turns out that your backup tapes cannot revive MPE, this tape can be used for a coldload, and a great deal of time saved (not to mention nerves).
   It can also be used to restart the system after a system failure.
   Update this tape every time a configuration change is made, and keep the tape in the computer room.

3. Keep up-to-date copies of the system I/O configuration in your files and taped to the computer room wall. SYSINFO will provide a comprehensive listing of all system parameters, both for you and for your CE.

4. Keep a list of the Disc Volume Table in your files and taped to the computer console. Ditto for the LOAD, START and DUMP parameter settings for the system, even if default settings are used.

5. Occasionally practice loading your DUS tape, and keep up to speed on the use of SADUTIL. SADUTIL can be of considerable help during a recovery -- but that is not the time to figure it out from the manual.

6. Prepare a written recovery procedure based on RELOAD. Validate this procedure with a live test! Work the bugs out and know what you are going to do BEFORE the moment of truth arrives. Involve your operators, and remember that plenty of recoveries have had to be done over the phone.

7. Maintain control over your backup tape library, and know exactly where everything is.

8. Do not put too much stock in the war stories that people tell about recoveries. You only hear about the successes (just like the stock market). Do not be duped into believing that recoveries are easy.

It is my heartfelt wish that you will never have to recover from a flames-and-ashes cataclysm (or even from a "routine" system failure). Hopefully, these suggestions will help you to be ready for a quick and confident recovery, should it come. In the meantime, you will sleep better knowing that you have addressed some of the more dreadful possibilities in the life of the system manager.

wp/3410d

I Haven't Got a Lot of Time - I Haven't Got a Lot of Money
(Food For Thought For Penniless System Managers)

George T. Blessing
City of Pasadena
100 N. Garfield Ave, B-29
Pasadena, CA 91109-7215

There is a seemingly steady flow of new products designed to help the System Manager take care of his system. Some of us, however, don't have the funds to pay the big-ticket prices associated with some of these items. For those of us in this situation, this paper suggests a few simple (and relatively cheap) methods to help us do our jobs more effectively without spending a lot of money or time.

More specifically, I will deal briefly with three major concerns: Backup Strategies, Disc Space Management, and File Capacity Management. We've all been told repeatedly that attention to details is the most important concern a System Manager can have. But paying attention to details can be time-consuming. What I have done at our site is to try to develop strategies for system management which require the least amount of time and give the largest benefits. In this paper, I will outline a few of these strategies and provide the listings for processes I've written where applicable.

BACKUP STRATEGIES

One of the problems facing us was a shrinking window for our overnight processing. At our site, we do a Full Backup on Monday morning, and a Partial Backup on Tuesday-Friday mornings. The backup took 3 to 3.5 hours for the Full, and 1.5 - 2 hours for the Partial. We also had only one operator who was working a split shift: coming in as early as 4:00 in the morning and then coming in later to do other work at night. Since additional staffing was out of the question and we needed more open time at night and better use of the operator we had, I decided something would have to be done.

I decided that the first thing to do was shorten the backup time, by cutting it into two pieces. We had two 6250bpi tape drives, but TurboSTORE was not yet released. I split the backup in half by defining two files with explicit account names in them, and doing a backup to each tape drive of different accounts. The format of the file containing the account names looks like this:

```
@.@.ALDON
@.@.BUDGET
@.@.CENTRAL
@.@.CLERK
    :
```

```
    :
@.@.INFOSYS
@.@.LICENSE
```

I stored the lists in two files called ACCT7 and ACCT17 (7 and 17 are the LDEV numbers of our tape drives), and we began splitting the backup by running one on each of two terminals. The backup commands for the backup to the LDEV 17 tape drive are:

```
:FILE SYSLIST;DEV=LP
:FILE TP17;DEV=17
:SYSDUMP *TP,*SYSLIST
Enter dump date: 10/1/87
Enter Subset of files:
!ACCT17
List files dumped to printer: Y
```

This change produced terrific results. The full backup was split into two halves which took similar amounts of time. The length of the full backup dropped to 1:50. The partial the next day dropped to about 0:55. Then I realized that the moment's paradise was slipping away fast. I now had two files which had to be maintained manually, and if they got out of whack, part of the system would not be backed up.

So I thought to myself "I need a way for the machine to maintain these, or I'll get burned for certain." I decided it wouldn't take me long to build an account and forget to put it in. So I picked an HP command which provides a list of all the accounts on the system which can be dumped into a file, and edited. I wrote JCL to create this file, text it into the editor, change it to the format I needed, and split it into two pieces, saving them as ACCT7 and ACCT17. I ran the job, and after a few tries, the output was correct. I went home with the satisfaction of a job I thought was well done.

I returned to the office to find trouble. The partial backup had taken too long and users were about to start complaining. The problem proved the theory that most things that look simple are actually not. The split which worked so well for the Full Backup, had soon proven to be unbalanced for the Partial. Though it was not so obvious on Tuesday, by Wednesday, the difference between the two sets was severe. Some accounts had more activity than others, and they happened to be on the same tape drive. So, I went back to the drawing board.

I modified the JCL for the job which sets up the files (ACCTJOB), changing it so that it did two splits, the first one for the expected balance for a full into FACCT7 and FACCT17, and the second one for partial backups into PACCT7 and PACCT17. I told the operator of the change. He was not thrilled that he would have to change his already typed documentation, and prepared for the worst. The changes worked well anyway, and after a couple tweaks to the partial side, it got even better. Here is the final listing of the JCL for ACCTJOB, with comments on what different commands are doing:

```
!JOB ACCTJOB,MANAGER.SYS
!OCSSTART
!COMMENT   THIS JOB EXAMINES THE ACCOUNTING STRUCTURE
!COMMENT   OF THE SYSTEM AND PREPARES EXPLICIT BACKUP
!COMMENT   LISTS, INSURING THAT NO ACCOUNT GETS LEFT
!COMMENT   OUT OF A BACKUP.  IT SHOULD BE RUN EACH DAY
!COMMENT   IN THE EVENING TO CAPTURE ANY ACCOUNTS BUILT
!COMMENT   DURING THE DAY.
!COMMENT
!COMMENT   THE FOLLOWING LINES PURGE THE CURRENT SPLITS.
!CONTINUE
!PURGE ACCTLIST.SYSOP.SYS
!CONTINUE
!PURGE PACCT7.SYSOP.SYS
!CONTINUE
!PURGE PACCT17.SYSOP.SYS
!CONTINUE
!PURGE FACCT7.SYSOP.SYS
!CONTINUE
!PURGE FACCT17.SYSOP.SYS
!COMMENT
!COMMENT   NOW, A REPORT IS DONE TO A TEMPORARY FILE.
!COMMENT
!FILE RLIST=RLIST;REC=-80,,F,ASCII;NOCCTL
!REPORT BUGGER.@,*RLIST
!COMMENT
!COMMENT   THE FILE IS TEXTED INTO THE EDITOR AND MODIFIED
!COMMENT   TO PRODUCE A FULL ACCOUNT LIST (ACCTLIST)
!COMMENT
!ED
 T RLIST
 SET TIME=9000
 CQ 9 TO "-" IN ALL
 FQ 1
 WHILE
   FQ "-"
   DQ *(*)/*(LAST)
 CQ 1 TO "@.@." IN ALL
 CQ " " TO "" IN ALL
 DQ 1/2
:COMMENT   AT THIS POINT THE FULL LIST IS FINISHED.
:COMMENT
:COMMENT   MODIFY FAMISX REFERENCE TO EXCLUDE THE
:COMMENT   GROUP @.DATA.FAMISX.
 CQ "@.@.FAMISX" TO "@.@.FAMISX-@.DATA.FAMISX" IN ALL
:COMMENT   SAVE THE FULL LIST
 K ACCTLIST.SYSOP.SYS,UNN
:COMMENT   THE FIRST SPLIT IS FOR PARTIAL BACKUPS
:COMMENT   (PACCT7,PACCT17).
:COMMENT   THE ACCOUNT MENTIONED BELOW - FQ "acctname"
:COMMENT   WILL BE THE LAST ACCOUNT IN THE FIRST SET.
 FQ FIRST
 FQ "FAMTEST"
 K PACCT7.SYSOP.SYS(FIRST/*),UNN
 FQ FIRST
 FQ "FAMTEST"
 K PACCT17.SYSOP.SYS(*+1/LAST),UNN
:COMMENT   THE SECOND SPLIT IS FOR FULL BACKUPS
:COMMENT   (FACCT7,FACCT17).
 FQ FIRST
 FQ "LICENSE"
 K FACCT7.SYSOP.SYS(FIRST/*),UNN
 FQ FIRST
 FQ "LICENSE"
 K FACCT17.SYSOP.SYS(*+1/LAST),UNN
:COMMENT   CLEAR OUT THE BUFFER
 DQ ALL
 YES
:COMMENT   TEXT IN SET 2 OF THE PARTIAL (PACCT17)
 T PACCT17.SYSOP.SYS
:COMMENT   ADD ONE LINE TO THE BEGINNING TO GET
:COMMENT   @.PUB.SYS AND @.DBLOG.@ ONTO THE FIRST
:COMMENT   TAPE.
 A .5
@.PUB.SYS,@.DBLOG.SYS
//
:COMMENT   KEEP SET 2 OF THE PARTIAL (PACCT17)
 K
 YES
```

```
:COMMENT
:COMMENT   TEXT IN SET 2 OF THE FULL (FACCT17)
 T FACCT17.SYSOP.SYS
:COMMENT   ADD ONE LINE TO THE BEGINNING TO GET
:COMMENT   @.PUB.SYS AND @.DBLOG.@ ONTO THE FIRST
:COMMENT   TAPE.
 A .5
@.PUB.SYS,@.DBLOG.SYS
//
:COMMENT   KEEP SET 2 OF THE FULL (FACCT17)
 K
 YES
:COMMENT
:COMMENT   CLEAR OUT THE BUFFER
 DQ ALL
 YES
 EXIT
!COMMENT
!EOJ
```

For those of you who have a contributed library, I recommend the use of BULDACCT (sometimes in pub.telesup) at the end of this jobstream to build current files with the accounting/udc structures in them to be backed up in the morning. This eliminates another one of those things you always want to do but never get around to.

This change was a solution, but not a complete one. I still wanted to get more mileage from my operator on Partial days. I examined the listings from the Partial backup, and after a few minutes, several items drew my immediate attention. Big datasets. Big datasets that got backed up every partial. I mean, datasets over 200,000 sectors a piece. So I picked up my expensive analytical tools (calculator and highlighter) and went to work. It shouldn't have been a surprise, but it was. Most of the backup for the first day or so was made up of datasets from the major applications.

Well, if you've been thinking about Image/TurboImage logging, and data recovery hasn't been enough of an issue to get you to do it, here are two more reasons to do it. First, if you're going to log a database and create that extra overhead, you may as well turn on Autodefer, since the log is written to disk whether Autodefer is on or not. Also, most people who didn't pay attention to the manuals or classes haven't noticed that performance with logging and Autodefer enabled is actually <u>better</u> than without it.

Second (and I must admit that this was my primary motivation), if you backup the log files on the partial, you don't need to backup the database itself. Now if your databases aren't much of your system, this will be meaningless, but if they are, think of the difference it would make. I did, twice, because I couldn't believe it the first time. Think of it. A 400,000 sector dataset receives 10 DBPUT's to it on Monday afternoon, and you're backing up the whole thing on Tuesday's partial. On a 6250bpi tape drive that's about 2/3 of one tape! Well, maybe I'm not a genius, but it didn't take me long to figure out that this would be the answer I was looking for.

So we tried it on two databases, then four, then nine. It was a hit. Response time on some database operations got faster, the backup got even shorter, and the night processing window grew to a fat 13 hours. We developed templates for different tasks to make the logging easier to manage and prevent us from being unprepared in case of corruption.

First, we developed a job stream to do a roll-forward recovery for one database. For each one of the databases which is logged, there is a copy of this job stream with the information for that database filled in. This enables us to recover by restoring the the database from the last full backup tapes and then applying the best available copy of the logfile against it by streaming the recovery job. Here is the template of the recovery file:

```
!JOB dbname,dbcreator.dbaccount,dbgroup
!COMMENT   THIS JOB RECOVERS THE DATABASE FROM
!COMMENT   THE LAST FULL BACKUP BY APPLYING TRANSACTIONS
!COMMENT   CONTAINED IN THE CURRENT LOG FILE.
!COMMENT
!COMMENT   FIRST SET FLAGS IN THE DATABASE
!RUN DBUTIL.PUB.SYS
 DISABLE dbname FOR ACCESS
 ENABLE dbname FOR RECOVERY
 EXIT
!COMMENT
!COMMENT   NOW, PERFORM THE RECOVERY
!RUN DBRECOV.PUB.SYS
 CONTROL NOSTAMP,NOSTORE
 RECOVER dbname
 RUN
```

```
!COMMENT
!COMMENT   THEN SET FLAGS BACK IN THE DATABASE
!RUN DBUTIL.PUB.SYS
 ENABLE dbname FOR ACCESS
 DISABLE dbname FOR RECOVERY
 EXIT
!COMMENT   DATABASE IS READY TO BE LOGGED AGAIN
!EOJ
```

We prepared jobstreams to handle the starting, restarting, and stopping of the logging processes, in order to reduce the error possibility. You will notice references in all three to the "existence" of a particular file. To allow the jobs to start logging processes, they need to be granted the capability of using the LOG command. To do this we have the jobs start, send a message to the console, and then wait for the existence of a particular file. When the operator sees the message, he enters the command SYSALLOW, which allows the job the LOG command and then builds the file. Here is are the listings of the three jobs, LOGSTRT, LOGRSTRT, and LOGSTOP:

```
!JOB LOGSTRT,OPRSM.SYS
!PURGE SYSALLOW.DBLOG
!COMMENT   THIS JOB STARTS ALL LOGGING PROCESSES
!COMMENT   IT MUST BE RUN ONLY AFTER THE FULL BACKUP!
!COMMENT
!COMMENT
!COMMENT   THERE MUST BE TWO LINES PER LOGGING PROCESS
!COMMENT   INSERTED BELOW IN THE FOLLOWING FORMAT:
!COMMENT    !FILE P=dbname.DBLOG.dbaccount;SAVE
!COMMENT    !PURGE *P
!COMMENT
!COMMENT ***** Account: CLERK *****
!FILE P=recdc.DBLOG.clerk;SAVE
!PURGE *P
!COMMENT
!COMMENT ***** Account: CP *****
!FILE P=cpdl.DBLOG.cp;SAVE
!PURGE *P
!COMMENT
!COMMENT ***** Account: ENG *****
!FILE P=pwrdf.DBLOG.eng;SAVE
!PURGE *P
!COMMENT
!COMMENT ***** Account: LICENSE *****
```

```
!FILE P=falldb.DBLOG.license;SAVE
!PURGE *P
!FILE P=fblldb.DBLOG.license;SAVE
!PURGE *P
!FILE P=fbl2db.DBLOG.license;SAVE
!PURGE *P
!COMMENT
!COMMENT ***** Account: PASADENA *****
!FILE P=citsdb.DBLOG.pasadena;SAVE
!PURGE *P
!COMMENT
!COMMENT ***** Account: PMSPAS *****
!FILE P=bsprdb.DBLOG.pmspas;SAVE
!PURGE *P
!FILE P=bssidb.DBLOG.pmspas;SAVE
!PURGE *P
!COMMENT ***** End of File Maintenance *****
!COMMENT
!TELLOP   ***************************************§
!TELLOP   *   YOU HAVE 3 MINUTES TO ISSUE THE   *
!TELLOP   *   FOLLOWING COMMAND OR THE LOGGING  *
!TELLOP   *   WILL NOT START.......             *
!TELLOP   *            :SYSALLOW                *
!TELLOP   ***************************************
!COMMENT   THE FOLLOWING PROGRAM CHECKS FOR EXISTENCE
!COMMENT   OF THE FILE SYSALLOW.DBLOG.SYS EVERY 15 SECS
!COMMENT   FOR 3 MIN.   IF FOUND THE PROCESS CONTINUES.
!RUN FILEINQP.COMP.OCS;PARM=15;INFO="12,SYSALLOW.DBLOG"
!PURGE SYSALLOW.DBLOG
!COMMENT   THERE MUST BE ONE LINE PER LOGGING PROCESS
!COMMENT   INSERTED BELOW IN THE FOLLOWING FORMAT:
!COMMENT         !LOG dbname,START
!COMMENT
!LOG recdc,START
!LOG falldb,START
!LOG fblldb,START
!LOG fbl2db,START
!LOG citsdb,START
!LOG bsprdb,START
!LOG bssidb,START
!LOG cpdl,START
!LOG pwrdf,START
!EOJ

!JOB LOGRSTRT,OPR.SYS
!PURGE SYSALLOW.DBLOG
!COMMENT   THIS JOB RESTARTS ALL LOGGING PROCESSES
!COMMENT   IT MUST BE RUN AFTER THE PARTIAL BACKUP!
```

```
!COMMENT
!TELLOP   ***************************************§
!TELLOP   *   YOU HAVE 3 MINUTES TO ISSUE THE    *
!TELLOP   *   FOLLOWING COMMAND OR THE LOGGING   *
!TELLOP   *   WILL NOT RESTART.....              *
!TELLOP   *           :SYSALLOW                  *
!TELLOP   ***************************************
!COMMENT   THE FOLLOWING PROGRAM CHECKS FOR EXISTENCE
!COMMENT   OF THE FILE SYSALLOW.DBLOG.SYS EVERY 15 SECS
!COMMENT   FOR 3 MIN.   IF FOUND THE PROCESS CONTINUES.
!RUN FILEINQP.COMP.OCS;PARM=15;INFO="12,SYSALLOW.DBLOG"
!PURGE SYSALLOW.DBLOG
!COMMENT
!COMMENT   THERE MUST BE ONE LINE PER LOGGING PROCESS
!COMMENT   INSERTED BELOW IN THE FOLLOWING FORMAT:
!COMMENT         !LOG dbname,RESTART
!COMMENT
!LOG recdc,RESTART
!LOG falldb,RESTART
!LOG fblldb,RESTART
!LOG fbl2db,RESTART
!LOG citsdb,RESTART
!LOG bsprdb,RESTART
!LOG bssidb,RESTART
!LOG pwrdf,RESTART
!LOG cpdl,RESTART
!EOJ

!JOB LOGSTOP,OPR.SYS
!PURGE SYSALLOW.DBLOG
!COMMENT   THIS JOB STOPS ALL LOGGING PROCESSES
!COMMENT   IT MUST BE RUN BEFORE FULL OR PARTIAL BACKUP!
!COMMENT
!TELLOP   ***************************************§
!TELLOP   *   YOU HAVE 3 MINUTES TO ISSUE THE    *
!TELLOP   *   FOLLOWING COMMAND OR THE LOGGING   *
!TELLOP   *   WILL NOT SHUT DOWN...              *
!TELLOP   *           :SYSALLOW                  *
!TELLOP   ***************************************
!COMMENT   THE FOLLOWING PROGRAM CHECKS FOR EXISTENCE
!COMMENT   OF THE FILE SYSALLOW.DBLOG.SYS EVERY 15 SECS
!COMMENT   FOR 3 MIN.   IF FOUND THE PROCESS CONTINUES.
!RUN FILEINQP.COMP.OCS;PARM=15;INFO="12,SYSALLOW.DBLOG"
!PURGE SYSALLOW.DBLOG
!COMMENT
!COMMENT   THERE MUST BE ONE LINE PER LOGGING PROCESS
!COMMENT   INSERTED BELOW IN THE FOLLOWING FORMAT:
!COMMENT         !LOG dbname,STOP
```

```
!COMMENT
!LOG recdc,STOP
!LOG falldb,STOP
!LOG fblldb,STOP
!LOG fbl2db,STOP
!LOG citsdb,STOP
!LOG bsprdb,STOP
!LOG bssidb,STOP
!LOG pwrdf,STOP
!LOG cpdl,STOP
!EOJ
```

To keep the logged databases themselves from being backed up during a partial backup, I created a jobstream which prevents them from being picked up by sysdump. Basically, the jobstream requests a store of the chosen files. The operator streams this job before doing the real partial, but never replies to the request for LDEV number for the tape. This keeps the files from being backed up by the other processes as they are in use by STORE. When the partial is completed, the operator replies to the request with an LDEV number of 0, and the jobstream terminates. The list of files to be kept from backup is kept in a separate file which can be edited. For those who have HPDESKMANAGER on their system, if you do a MAILMAINT each night, you may choose not to backup the mail databases in the morning (the MAILMAINT job does a store of the databases before the mailmaint takes place). Here is the listing of the JCL for the lockdown job, DBLLOCK, and its configuration file, DBLLIST:

```
!JOB DBLLOCK,OPR.SYS
!TELLOP   ***********************************************
!TELLOP   *  THIS HAD BETTER BE A PARTIAL!§ BACKUP...    *
!TELLOP   *  IT IS OK TO BEGIN THE REAL PARTIAL BACKUP  *
!TELLOP   *  ONCE THE REQUEST FOR "LOCKTAPE" HAS COME   *
!TELLOP   *  FROM THIS JOB...                           *
!TELLOP   ***********************************************
!TELLOP   *  IF THIS ISN'T A PARTIAL, REPLY '0' TO THE  *
!TELLOP   *  REQUEST FOR "LOCKTAPE" NOW.                *
!TELLOP   ***********************************************
!TELLOP   *  AFTER THE PARTIAL BACKUP HAS COMPLETED,    *
!TELLOP   *  REPLY '0' TO THE REQUEST FOR "LOCKTAPE".   *
!TELLOP   ***********************************************
!FILE LOCKTAPE;DEV=TAPE
!STORE !DBLLIST;*LOCKTAPE;SHOW
```

```
!EOJ


RECDC@.PUB.CLERK
FAL1DB@.DATA.LICENSE
FBL1DB@.DATA.LICENSE
FBL2DB@.DATA.LICENSE
CITSDB@.PUB.PASADENA
BSSIDB@.DATA.PMSPAS
BSPRDB@.DATA.PMSPAS
@.@.HPOFFICE
```

So, with the implementation of the split backup, image logging, and no backups of logged databases on partial days, the full backup now takes about 1:45 to complete, and the partials vary from 20 - 30 minutes. We do a full backup now on Monday, since it has the largest processing window in front of it, and partial backups on Tuesday through Friday.

DISC SPACE MANAGEMENT

The next problem I tackled is a difficult one to deal with: The use/abuse of disc space. This is particularly tough in a development environment where people are leaving their sometimes not-so-little test files around for posterity. It's not possible for you to watch everything that goes on, so let the machine do everything possible for you. I don't want to plug anyone's software, but I confess that a great deal of this function is being handled by MPEX from Vesoft.

I use two jobstreams to monitor the appearance of files on the system: one that alerts me to new files being built, and a second one that lists files not accessed. The first job logs on at 2AM on Saturday morning, and does a special LISTF of all files created during the past week. You should try it just to see what shows up. Huge files with no records often appear with misspelled names that seem to elude their creator's dustpan.

```
!JOB NEWFILES,MGR.OCS
!COMMENT  THIS JOB LISTS ALL FILES CREATED IN THE LAST
!COMMENT  WORK WEEK WHICH ARE LARGER THAN 500 SECTORS.
!FILE PP;DEV=PP;ENV=LP4.HPENV.SYS
!CONTINUE
!RUN MPEX.PUB.VESOFT
```

```
SET DEFAULT,LISTF,!
LISTF @.@.@(CREDATE>=*-7 & DISCSPACE>500),2;*PP
EXIT
!EOJ
```

Then at around 3AM the second job logs on. It also uses MPEX and does a listf of all files that have not been accessed in the last 90 days. This listing often reveals only items which need to be archived, but occasionally a choice find appears. It's a good practice to get into, but again, one which takes time to run and time to review. So let the machine at least take care of running it while you're not around.

```
!JOB OLDFILES,MGR.OCS
!COMMENT  THIS JOB LISTS ALL FILES NOT ACCESSED IN THE
!COMMENT  PAST 90 DAYS.
!FILE PP;DEV=PP;ENV=LP2.HPENV.SYS
!CONTINUE
!RUN MPEX.PUB.VESOFT
SET DEFAULT,LISTF,!
LISTF @.@.@(ACCDATE<*-90),2;*PP
EXIT
!EOJ
```

I also believe that it's important to watch for sudden changes in freespace on your system. The Hewlett-Packard utility FREE5.PUB.SYS (as we all know) is supplied with the fundamental operating system and is quite capable of creating its own instant I/O bottleneck when running. The output of this program can be useful in determining how fragmented the freespace on your discs is or the amount of total freespace on the system. I have found that having a record of the total freespace can help determine what has been happening on the system. Obviously, if the amount of free sectors drops by 1.2 million, someone is building some hefty files. To facilitate having the operator record the freespace on a daily basis, I wrote a jobstream to supply him with only the information I was interested in.

```
!JOB FREEJOB,MGR.OCS
!CONTINUE
!PURGE FREE5OUT
!BUILD FREE5OUT;REC=-80,,F,ASCII;NOCCTL
```

```
!RUN FREE5.PUB.SYS;STDLIST=FREE5OUT
!IF JCW < FATAL THEN
!  CONTINUE
!  ED
T FREE5OUT
F FIRST
WHILE
  FQ "LARGEST"
  DQ */*+7
DQ 1
CO LAST-1 TO 1
A .5
          SYSTEM FREE SPACE
//
CQ 1 TO ":TELLOP " IN ALL
K TEMPUSE,UNN
USE TEMPUSE
:PURGE TEMPUSE
DQ ALL
YES
EXIT
!ENDIF
```

This jobstream runs FREE5.PUB.SYS and directs the output to a file. It then runs EDITOR.PUB.SYS and massages the discspace information until it becomes a series of :TELLOP commands. The altered file is saved in the file TEMPUSE, then USEd by the editor to send messages to the console showing only the total freespace for each disc drive and the system as a whole. Although this output is rather rudimentary and was intended for a written record, if you have access to a PC with a spreadsheet application, time comparisons can be made. I am hoping to write an application later in this year to go a little further and evaluate the condition of the discs based on the fragmentation information as well as the freespace.

FILE CAPACITY

The last thing I'd like to discuss is database capacities. If you are involved in a small shop, it's very likely that as the System Manager you work many different jobs. Probably you are also acting as the Database Administrator. There is very little (with the possible exception of a system failure) that annoys me as much as having a dataset fill up unexpectedly. Our system has a great many small applications running on it, producing an endless supply of databases. Keeping track of how full each of them is could be a full-time job in itself. In fact, on our system just keeping track of where they are could be a full-time job. So while I was wearing the DBA hat one day I decided it would be wonderful if the HP3000 could just tell me when sets were filling up.

Evidently, other people were thinking the same thing because I've noticed at least one commercial product that seems to do this to some degree. But my no-cost procedure at least gives me a list of the sets which have passed a particular threshold. Hewlett-Packard provided the means to accomplish this quite a while back, perhaps without even knowing it. When they introduced TurboIMAGE, they also introduced a special group in the SYS account called CONVALL. For those of you who haven't checked it out, I suggest that you do. It contains several goodies like TMPCONVP, a program which lets you change your session log-on without losing your capabilities or UDC's. Aside from that, it also contained a utility which searches the system, looking for databases, and creates a JCL from a supplied template file for each one it finds. Its original use was to convert each DB to TurboIMAGE. The CONVALL programs were well-documented and is able to insert variables such as the creator, passwords, etc. into the template you supply. I imagine that this utility is still available with some coaxing from your S.E. or may be on one of your old tapes.

I decided to use this utility in combination with a contributed program called DBANALYS. DBANALYS can provide a list similar to FORM SETS in QUERY, but also shows the percentage of each dataset in use. I know there are other contributed programs floating around which provide similar output. I've been using this one for some time now and have stuck with it. I wrote a sequence of jobstreams which essentially performs three tasks: 1) find all of the databases on the system and produce JCL to run DBANALYS against each database, appending the output from DBANALYS to a single file 2) launch the new JCL 3) massage the output from DBANALYS and produce a report. Below are the listings of DBCAPJ1, DBCAPJ3, and a sample report from the process.

```
!JOB DBCAPJ1,MANAGER.SYS,SYSOP
!OCSSTART
!COMMENT  ***************************************************
!COMMENT  *  THIS JOB IS THE FIRST OF THREE JOBS USED       *
```

```
!COMMENT  *  TO CHECK FOR DATASETS OVER 85% FULL ON    *
!COMMENT  *  THE SYSTEM...                             *
!COMMENT  **********************************************
!COMMENT
!COMMENT  **********************************************
!COMMENT  *  FIRST, PURGE EXPENDABLE FILES...          *
!COMMENT  **********************************************
!PURGE DBCAPX.SYSOP
!PURGE DBCAPJ2.SYSOP
!PURGE JCONVERT.SYSOP
!PURGE TEMPLATE.SYSOP
!COMMENT  **********************************************
!COMMENT  *  THEN, BUILD NEW FILES...                  *
!COMMENT  **********************************************
!BUILD DBCAPX;REC=-80,128,F,ASCII;DISC=100000,32,1
!RELEASE DBCAPX.SYSOP
!COMMENT  **********************************************
!COMMENT  *  BUILD THE TEMPLATE FILE...                *
!COMMENT  **********************************************
!ED
A
|RUN TMPCONVP.CONVALL.SYS
\FILE,\LOGON
|FILE DBCAPX=DBCAPX.SYSOP.SYS;ACC=APPEND
|CONTINUE
|RUN DBANALYS.UTIL.SYS;STDLIST=*DBCAPX
BASE
\FNAME
;
5
SETS
EXIT
//
CHANGE "|","!" IN ALL
:COMMENT  *  NOW KEEP THE TEMPLATE FILE...              *
K TEMPLATE,UNN
EXIT
!COMMENT  **********************************************
!COMMENT  *  FIND ALL OF THE DB'S...                   *
!COMMENT  **********************************************
!BUILD JCONVERT;REC=-80,128,F,ASCII;DISC=100000,32,1
!FILE TEMPLATE.CONVALL.SYS=TEMPLATE.SYSOP.SYS
!RUN CONVALL.CONVALL.SYS
@.@
!FILE NEWCON=JCONVERT,OLD
!FILE OLDCON=JCONVERT,OLDTEMP
!FCOPY FROM=*OLDCON;TO=*NEWCON
!PURGE JCONVERT,TEMP
```

```
!COMMENT  ***********************************************
!COMMENT  *  CREATE THE JOBSTREAM DBCAPJ2...            *
!COMMENT  ***********************************************
!ED
T JCONVERT
A.5
|JOB DBCAPJ2,MANAGER.SYS,SYSOP
|OCSSTART
//
A
|RUN TMPCONVP.CONVALL.SYS
DBCAPJ2,MANAGER.SYS,SYSOP
|IF JCW < FATAL THEN
|SCHEDULE "DBCAPJ3"
Y
|OCSNORMAL
|SET STDLIST=DELETE
|ELSE
|OCSABNORM
|ENDIF
|EOJ
//
CHANGE "|" TO "!" IN ALL
K DBCAPJ2,UNN
EXIT
!COMMENT  ***********************************************
!COMMENT  *  READY FOR DBCAPJ2 TO EXECUTE...            *
!COMMENT  ***********************************************
!IF JCW < FATAL THEN
!SCHEDULE "DBCAPJ2"
Y
!OCSNORMAL
!SET STDLIST=DELETE
!ELSE
!OCSABNORM
!ENDIF
!EOJ

!JOB DBCAPJ3,MANAGER.SYS,SYSOP
!OCSSTART
!ED
 T DBCAPX
 SET TIME=9000
 CQ 67 TO "|" IN ALL
:COMMENT ***************************************************
:COMMENT ** THE FOUR LINES FOLLOWING ENSURE REPORTING OF **
:COMMENT ** THE CAPACITY OF THE ITEM-HEADER SET IN HPDESK *
:COMMENT ** WITHOUT RESTRICTION BY PERCENTAGE            **
```

```
FQ FIRST
F "LOCAL"
F "ITEM-HEADER"
CQ "|","" IN *
:COMMENT ***************************************************
 CQ "|8 ","|0" IN ALL
 CQ "|9 ","|0" IN ALL
 CQ "|80","|0" IN ALL
 CQ "|81","|0" IN ALL
 CQ "|82","|0" IN ALL
 CQ "|83","|0" IN ALL
 CQ "|84","|0" IN ALL
 CQ "|8","8" IN ALL
 CQ "|9","9" IN ALL
 CQ "|  "," " IN ALL
 FQ FIRST
 WHILE
   FQ "|"
   DQ *
 FQ FIRST
 FQ "EXIT"
 COPY *-1/* TO 1
 COPY 1 TO .5
 CHANGE "->" TO "REPORT OF DATASETS WITH LOADING >85% OF CAPACITY"
 FQ FIRST
 WHILE
   FQ "->"
   DQ */*+13
:COMMENT NEW STUFF ADDED 12/16/87
 C 26 TO "|" IN ALL
 FQ FIRST
 WHILE
   FQ " |"
   DQ *
 C "|" TO "" IN ALL
 C 5 TO "|" IN ALL
 FQ FIRST
 WHILE
   FQ " |"
   BEGIN
     CQ "SET|S" TO "|SET|S" IN *-1
     FQ *+2
     END
 FQ FIRST
 WHILE
   FQ " SET|S"
   DQ *
 C "|" TO "" IN ALL
 L ALL,UNN,OFFLINE
 EXIT
!IF JCW < FATAL THEN
!OCSNORMAL
!SET STDLIST=DELETE
!ELSE
!OCSABNORM
!ENDIF
!EOJ
```

```
REPORT OF DATASETS WITH LOADING >85% OF CAPACITY
 SETS  OF DATABASE : OBV.FGLSRC.BUDGET          MAY 26, 1988
  11  PERSONNEL        DETAIL       3000       2920      97
 SETS  OF DATABASE : PHONDB.DATA.CENTRAL
   6  ODX'M-IRN        MANUAL       809        726       90
   8  ODX'M-IRN-TREE   DETAIL       62         58        94
 SETS  OF DATABASE : SURP.DATA.CENTRAL
   5  RDETAIL          DETAIL       18984      16101     85
                               .
                               .
                               .
```

This process now runs at night once a week and enables us to review quickly what datasets are reaching the threshold and need attention without having to wade through a stack of printouts.

I have also felt somewhat remiss in the amount of attention which I can pay to the loading of the disc drives. As files tend to move if they are built on class DISC, it can be difficult to follow them around. I was particularly interested in datasets and the problems that develop when two related or heavily traveled sets reside on the same LDEV. When I started to evaluate them after the last reload, I stopped almost immediately. It is difficult to remember that in database AIDB the critical sets are 10, 14, and 15, and not 12, 16, and 17. After creating a paper nightmare and still not effectively solving the problem, I called on my old friend EDIT/3000 one more time. Since datasets are built on one LDEV (unless you mess with them), I reasoned that the output from LISTDIR5 with the MAP option would provide me with the LDEV number for any file I chose to list.

So, I began to work on this problem. I built a small editor file in which I placed the names of several datasets (e.g. AIDB04.PUB.SO) and a brief description of each set (to remind me of their contents). I wrote a jobstream which texts in that file, and changes it into a series of commands for LISTDIR5. It then runs LISTDIR5 using the new file of commands as its $STDIN. The output from LISTDIR5 is captured in a new file, which is brought into the editor again. The editor massages the data, passes it through a sort by LDEV number and filename, and then on to a printed report. The report lists the files and their respective LDEV numbers and shows at a glance which files are on which disc drives. Below is a listing of the files DLCONFIG, DLJOB, and a sample report.

```
CITSDB01.PUB.PASADENA         M,CITATION
CITSDB02.PUB.PASADENA         A,LICENSE
CITSDB04.PUB.PASADENA         D,VEHICLE-ID-NO
CITSDB07.PUB.PASADENA         D,BAIL-DISPOSITION

FMMASTER.DATA.FAMIS           KSAM, FAMIS MASTER
FMMASKEY.DATA.FAMIS           KSAMK, FAMIS MASTER KEYFILE

CPDL05.PUB.CP                 A,M-CITE-KEY
CPDL07.PUB.CP                 A,M-COURT-KEY
CPDL13.PUB.CP                 A,M-IRN
CPDL20.PUB.CP                 D,CDISPO
CPDL21.PUB.CP                 D,CHARGES
CPDL23.PUB.CP                 D,CHISTORY
CPDL26.PUB.CP                 D,COURT
CPDL28.PUB.CP                 D,MASTER
```

```
!JOB DLJOB,MANAGER.SYS,SYSOP
!PURGE DLIN
!PURGE DLOUT
!PURGE DLUSE
!ED
SET TIME = 9000
T DLCONFIG.SYSOP.SYS
CQ 2 TO "|" IN ALL
FQ FIRST
WHILE
  FQ " |"
  DQ *
CQ "|","" IN ALL
CQ 31/72 TO "" IN ALL
CQ 1 TO "LISTF " IN ALL
A
EXIT
```

```
//
K DLIN,UNN
EXIT
!COMMENT
!BUILD DLOUT;REC=-72,7,F,ASCII;DISC=1000
!RUN LISTDIR5.PUB.SYS;STDIN=DLIN;STDLIST=DLOUT
!COMMENT
!ED
SET TIME=9000
T DLOUT
CQ 2 TO "|" IN ALL
FQ FIRST
WHILE
  FQ " |"
  DQ *
CQ "|" TO "" IN ALL
FQ FIRST
WHILE
  FQ "FCODE"
  DQ * / "DISC DEV"
FQ FIRST
WHILE
  FQ "DISC TYPE"
  DQ * / ">"
FQ FIRST
WHILE
  FQ "LISTF"
  DQ * / *+1
CQ "*"," " IN ALL
CQ 3 TO "|" IN ALL
FQ FIRST
WHILE
  FQ " |"
  DQ *
CQ "|","" IN ALL
CQ " #"," " IN ALL
FQ FIRST
WHILE
  FQ "ACCESSED:"
  DQ *(*) / *(72)
CQ " :","                              " IN ALL
CQ "FILE: ","+" IN ALL
G ALL
DQ 1/2
DQ LAST
K
EXIT
!ED
```

```
SET TIME=9000
T DLOUT
FQ FIRST
WHILE
  FQ "+"
  DQ *
CQ 32 TO \C 31 TO "\ IN ALL
CQ 43 TO \" IN *+1\ IN ALL
K DLUSE,UNN
T DLOUT
CQ 2 TO "|" IN ALL
FQ FIRST
WHILE
  FQ " |"
  DQ *
CQ "+|","" IN ALL
A .5

//
FQ FIRST
USE DLUSE
K
EXIT
!FILE DLLIST;DEV=LP
!RUN SORT.PUB.SYS
I DLOUT
O *DLLIST
K 32,1
K 31,1
K 1,8
V
END
!TELL MANAGER.SYS;DLJOB COMPLETED!
!EOJ
```

```
FMMASTER.DATA.FAMIS                  1
CITSDB04.PUB.PASADENA                3
FMMASKEY.DATA.FAMIS                  3
               .
               .
               .
CITSDB01.PUB.PASADENA                9
CITSDB07.PUB.PASADENA                12
CITSDB08.PUB.PASADENA                12
CITSDB02.PUB.PASADENA                13
```

I realize that my methods are no panacea, and could be easily expanded upon. I'm certain that those who pick up where I have stopped will continue to evolve some of these ideas into even more efficient and practical tools, and I expect mainly to spark the curiosity and imagination of others.

Remote PC Information Network

John Wilson
Dan MacFayden

Coles Book Stores Limited
90 Ronson Drive
Rexdale, Ontario
M9W 1C1

I would like to introduce myself, my name is John Wilson and I am the Systems Development Manager for Coles Book Stores. I am speaking to you this morning along with Dan MacFayden, the Project Leader responsible for our network systems.

This paper is a bi-product of the development of a PC based store information network for a retail book chain with 192 loctions across Canada and a head office in Toronto utilizing two HP3000s (a 68 and a 70)

With 100 locations now up and live, we hope to share with you some of our findings on how to integrate PCs into a distributed DP environment.

THE ENVIRONMENT

I would like you to consider the following corporate environment:

- * The existance of multiple business locations spread across a large geographic area.
- * A centralized corporate office supporting the business functions common to all locations.

This type of environment is common to many industries because it makes good business sense to exploit the advantages associated with an economy of scale.

Examples - Customer Service Organizations with multiple depots.
- Branch Sales and Distribution Operations located close to the customers.
- Retail specialty chains commonly found in malls across the country.

The centralized business functions are supported by a computing facility running customized business application software for:

- Financials
- Order Processing
- Invoicing
- Shipping

and most other facets of the business to improve both the company's profitability and level of support to its customer base and remote sites.

Traditionally, much of the activity in the remote sites is used to feed the systems at H.O. with only limited information being sent back to the remote sites to support "their" own business activities.

Herein lies both the problem and the opportunity. Access to the corporate information data base is essential to the optimum operation of the remote sites. At the same time, the value of the corporate data is only as good as the accuracy and timeliness under which it is acquired from those same remote sites. There are two basic opportunities available in moving the access to corporate data and business applications out to the remote sites:

1. An online network to provide access to the corporate information and applications.

2. A PC network with remote application software and a subset of the corporate data updated using batch data communications to keep the remote and corporate data bases syncronized.

Both of these alternatives are feasible but up until recently, the online network has usually been the choice of most corporations willing to bear the costs of developing and running a network. However, the availability of low cost micro computers, comprehensive software development tools and solid data communications software has meant that our second alternative is now both feasible and cost justifiable.

When making a decision on which alternative to select, given the environment under consideration, the main consideration is based on a need for information updated minute by minute at the remote sites or will remote information updated on a daily basis be sufficient. The answer to this question will indicate whether a batch integrated distributed environment is applicable or not for your business.

BENEFITS

There are many benefits to be obtained from implementing the integrated PC solution in the remote sites:

- Savings: There is the opportunity to save manhours through the automation of tasks at the remote sites.

- Improved Operations and Profitability: Operations at the remote site can be improved in its efficiency and profitability through access to accurate and timely information at the remote site. (eg. reduced inventory levels, faster reorder cycles, better customer service, improved margin with better costs and price controls, improved cash flow)

- the corporate facilities do not have to be enhanced to support an online network with all of its added complexity and back up requirements.

- There is no single point of failure with the concept of distributed processing.

- Much of the processing requirements are at the remote site with little impact on the corporate resources.

- The data communication costs are substantially less in volume, can be done during non-business hours and do not even require dedicated phone lines at each remote site.

- Tailored application systems can be developed for the remote PCs without the high cost of multiple site licenses from software vendors.

- Onsite service is available across the country for all hardware

Lets first take a look at the overall framework of a distributed data processing environment using PCs. We will then delve further into what the structure of the remote PCs data base should be and the actual components that will make the distributed process work.

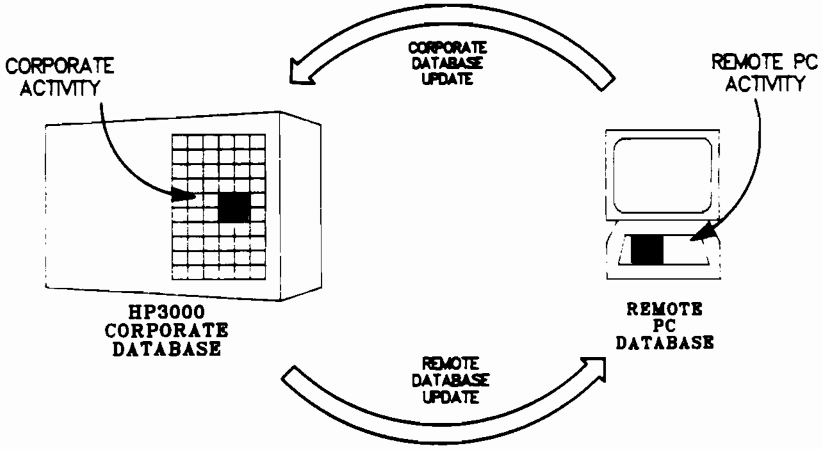
THE FRAMEWORK

The concept of developing applications to run on a remote PC is based on taking a subset of the corporate data base as it is applicable to a particular remote site and to put it on a PC. Applications are then written for the PC to support the business activities at the remote site using the information on the PC data base.

Any transaction affecting the remote data base must be accumulated for transmission to H.O. to be used as input to the production systems and update the corporate data base. Conversely, any online or batch production activity on the corporate systems that affects the remote PC data base must be accumulated for transmission to the remote site.

What is important is the ability to develop a distributed environment where the remote and corporate databases cannot go out of sync. Controls must be built into the database update processes to ensure that the databases cannot be corrupted by duplicate, lost or bad data. This means that transactions unsuccessful in updating a database must be cycled back to the originating location for the necessary action. The measure of how solid the systems are at both ends, including the data communications software, will dictate the number of error situations to be dealt with. A word of advice to those of you contemplating development of a distributed processing environment: if your present corporate systems are not solid don't extrapolate your problems by spreading your errors around onto additional machines.

FRAMEWORK



STRUCTURE

Let us consider one more area when looking at the framework of a distributed environment and that is the logical structure of the databases on the corporate and remote system. Corporate systems are usually made up of "static master file" data such as product, vendor, department information and secondly, "transaction based data" such as purchase orders, invoices, etc. Integrated corporate systems structure the data to support both company level processing and for production based systems focused at the remote business activity level (eg. creating purchase orders or invoices).

The processing requirements for the remote PC only require those pieces of static master file data and transaction based data as is necessary for that individual remote site to function.

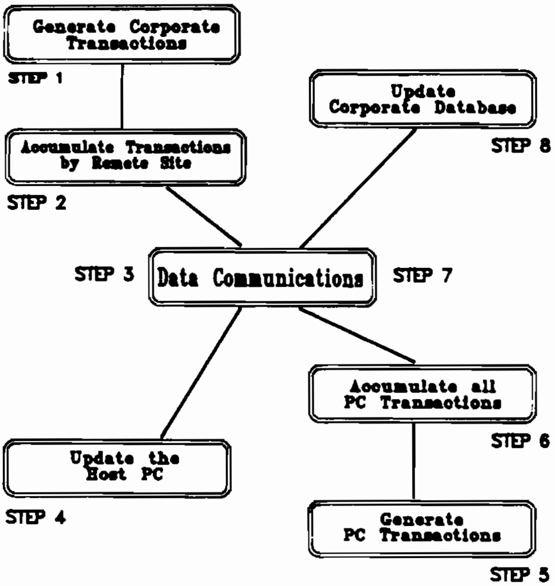
ie. Products carried, related vendors, store orders, associated sales history

What we can recommend is that the structure of the remote data base must reflect the data base structure on the corporate systems. This gives you a situation which is straight forward to understand and possible to keep synchronized. Without the similar structures, it would be extremely difficult if not impossible to even detect when the systems are out of sync let alone know how to go about correcting the situation.

COMPONENTS

The process of keeping the corporate database and the remote locations' data bases synchronized is comprised of a number of component activities. These components can be viewed separately in order to more easily understand and maintain the integrated processing that is involved:

COMPONENTS



Generate Corporate Transactions
STEP 1
Accumulate Transactions by Remote Site
STEP 2
Data Communications
STEP 3
STEP 7
Update the Host PC
STEP 4
Generate PC Transactions
STEP 5
Accumulate all PC Transactions
STEP 6
Update Corporate Database
STEP 8

Step 1 <u>Generate Corporate Transactions</u>

Every corporate system, (online or batch), that alters masterfile or transaction data on the corporate database that is also carried on the remote PC database must generate a transaction that can be used to update the remote databases.

There are two alternatives for the transaction record format: The first is to simply send all information about the record to be updated or secondly, to develop complex techniques that identify the pieces of data that have been changed and only send this data. This will obviously have an impact when considering your data communication costs.

Step 2 **Accumulate All Transactions by Remote Location**

The transactions that have been generated by the corporate systems must be accumulated by remote location and organized by date and time so the same processing sequence will occur at the remote sites. It will also be necessary to append transactions to any remote site's files that have not transmitted since the last time this accumulation was run.

On the HP3000 it is advisable to set up separate groups for each remote site to hold the files that are specifically for that location.

Step 3 **Data Communications from Head Office to the Remote PC**

There are a number of products available today that are able to provide error free data transmissions between an HP3000 and remote PCs. There are, however, a number of other features of the communications component that are not only desirable, but essential, if communications with a large number of remote PCs is to be successful. These include:

- the ability to initiate communications without requiring manual intervention at the remote locations or at head office for unattended after hours transmission

- an automatic restart capability beginning from the last successfully transmitted transaction, whenever communications is interrupted.

- file integrity routines such as traditional batch header control techniques to ensure the correct data is being processed by the proper location.

- the ability to maintain and monitor the transmission status for each transmit file, coupled with the ability to retransmit unsuccessfully transmitted files in the next run.

- the availability of network management information for performance and trend reporting as well as daily problem and exception reporting.

Step 4 **Update the Remote PC**

Software on the remote PC will process the Head Office transactions and update the data base. Controls should be built in to insure the integrity of the remote database is not threatened by corrupt incoming transactions or by failure during the transaction processing. The update component can also provide the ability to generate reports to be printed at the start of the next day. New versions of programs can be implemented at this time (if they have been down loaded from the corporate machine). A transaction activity summary should be generated to inform the remote site of the activity that occurred over night and any transactions that could not be processed successfully should be added to an error file to be transmitted back to the corporate site for reivew.

Step 5 **Generate PC Transactions**

All systems on the PC that update master file data or generate dynamic data that impacts corporate systems must generate a logical transaction that can be sent back to the corporate site.

Step 6 **Accumulate All Transactions on the PC**

Before setting up for data communications this component has to create the files containing:

- the new transactions to be processed at head office
- any previously rejected head office transactions
- relevant PC database and disk utilization statistics to be used at head office to manage and support the remote PCs

Step 7 **Data Communications from the Remote PC to Head Office**

This process is the same as the head office to store data communications with the same controls and restart capabilities.

Step 8 **Update the Corproate Database**

The incoming transaction files from the remote loctions must be split up by transaction type and the transactions integrated into the associated corporate production systems.

Any transactions that can not be successfully processed should be reported at head office for investigative and corrective action, and the remote loation should be informed via the next transmission.

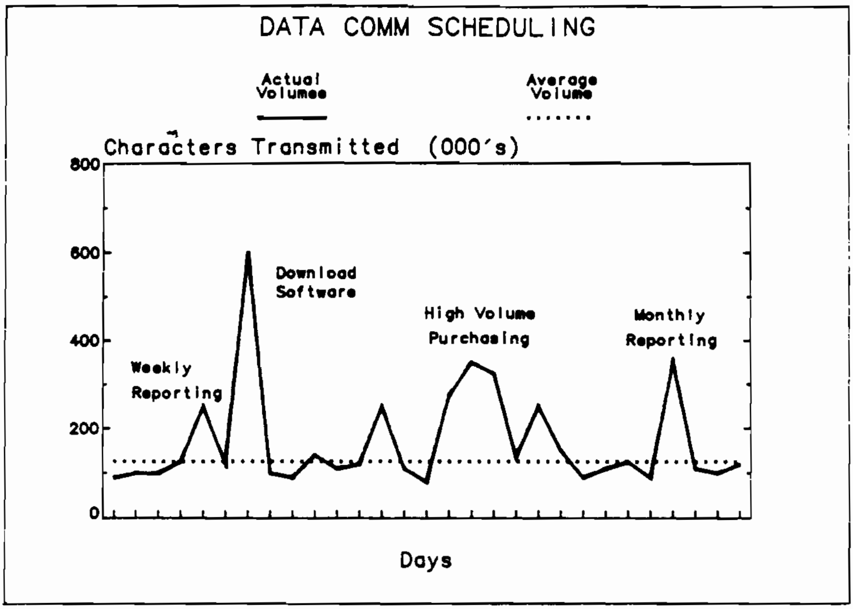
DATA COMM SCHEDULING

There are two basic alternatives for driving the communications process--either the central HP3000 or the remote PCs can initiate or drive the communications process. Software products exist to support either of these alternatives.

In our opinion, PC initiated communications is viable in a multiple remote-location scenario if the volume of transactions per location is small or predictable enough for you to schedule the incoming calls sufficiently far apart to avoid contention. It is also viable if you can afford to add enough incoming communication lines to avoid contention. But recognize that when the remote sites are scheduling the upcoming calls there are certain factors that you cannot control which can cause contention:

- the corporate HP3000 may be unavailable
- the initial data communication attempt may fail, forcing an unscheduled re-dial
- increased transaction volumes can require longer connect times than planned

So, if your corporate processing resources are limited, or your transaction volumes are volatile, we recommend using the corporate HP3000 to poll the remote PCs.



This allows you to handle planned and unplanned transmission variances such as the planned release of software revisions or an unplanned increase in transaction volumes. From head office you can also reschedule any redial attempts later in your transmission window whenever a communication link has been unsuccessful without causing contention with other sites.

The priority of communications with specific remote PCs can also be parameterized and controlled. Indeed, it is much easier to alter the entire polling pattern (based on volume changes, business activity changes, special circumstances, etc.) when you have control of the data comm at head office.

Once you have purchased your hardware and your network has been developed and is now operational, the greatest cost you will incur will be your data communication cost. A centralized polling facility with consolidated network performance reporting gives you the best opportunity to control your data comm costs.

MAINTAINING A DISTRIBUTED PC NETWORK

Now that you have your network up and functional with nightly data communication processes to maintain the remote and corporate systems, you must be prepared to support the remote sites.

For most companies, technical support is only available at the central site due to its high cost. Not only is technical expertise unavailable at the remote sites, but in a PC environment the level of computer expertise will usually be limited to the use of business application software only. This makes it unwise to rely on personnel at the remote sites for any computer maintenance related activity that might usually be found in a corporate mini or mainframe environment. Remote personnel cannot be relied upon to perform computer-problem diagnostics as part of their job responsibilities.

We have addressed this limitation of a lack of expertise at the remote sites by employing a strategy that provides for centralized network support, that can resolve any PC or network problem from the central location. In order to achieve this objective we have developed the following tools and procedures:

1. A PC software product that allows access to the remote PC by corporate support personnel. This provides access to MS-DOS level commands and diagnostics. Software problems at the remote location can also be investigated by running versions of the same application programs on a corporate PC while utilizing the remote PC's data.

2. Tools similar to the HP3000 Image database utilities are used to remotely rebuild a PC database or dataset as well as perform other necessary database management functions. It is worth emphasizing that these remote utilities must have a sufficient level of data communications integrity to insure that corruption of the remote data is not caused by the investigation process.

3. A facility is used at the corporate site to download program revisions and special corrective programs to a single site or to all sites as part of the daily transmit process. Instructions and manual updates can also be sent with any program revisions.

4. In the event of a disaster where the hard disk is lost, software is available to recover and rebuild the systems for a remote site. This can be done from a local backup, or from the corporate database which contains the subset of data that constitutes the remote site's database. This is one of the biggest benefits of keeping the remote and corporate databases in sync.

5. Escalation routines are in place to insure problems are not left unresolved. Included in these routines is an adequate hardware service contract for each remote site.

6. Network support reporting is used to:

   - monitor database and disk capacities for the remote sites
   - provide historical trend analysis by transaction type
   - provide daily operational and management reporting of network data communications activity and costs

   We have found that reporting on an exception basis for the network support staff is most useful in focusing attention on possible problems.

7. A messaging facility is used to keep the remote sites informed whenever potential problems are discovered or solutions are introduced as well as to introduce new processses, complete with the manual updates.

INSIGHT

Based on our experience at Coles, we would like to relate the following "bleeding edge" lessons to anyone who is considering using PCs in a distributed DP environment.

1. Ask yourself how up-to-date the information on your remote PCs has to be. If you really need minute-by-minute updates then the process we have described may not be appropriate.

2. If your corporate systems are not in place and stable, or if you anticipate dynamic changes in your corporate systems, then you should probably defer the development of the PC network until the work is completed.

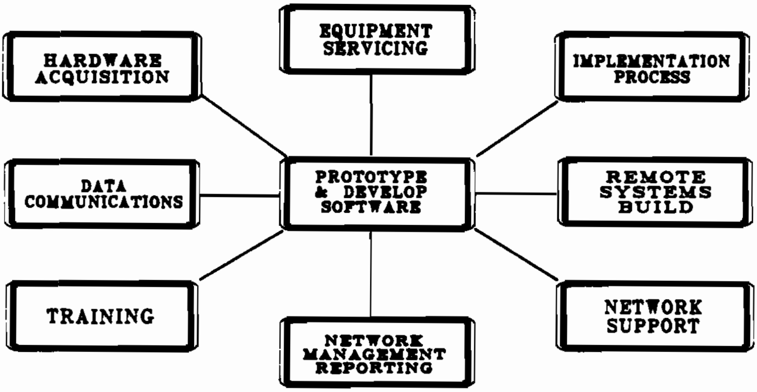
Moving from a centralized DP environment to a distributed approach with many locations cannot be done overnight. Your corporate systems must be able to support both remote locations on the existing process and al$o the integration of the distributed PCs as they are implemented. Also recognize that business does not stand still and changes will be required as new business opportunities dictate. There will be a need to maintain and evolve both the new and the old systems during your conversion process.

3. Prototype your remote PC application before developing the interfaces with the corporate systems. This allows the flexibility of fine tuning the PC design before developing the interfaces to the corporate systems.

4. If the number of remote sites is low, then you may not require the level of automation we have described for data communications, remote support, network management, etc. Twenty local sites is about the maximum that can be implemented in a "manual" mode. More sites or a greater geographical spread will require automation of many of your processes.

5. Set up separate project teams and the necessary project controls in the areas shown in the following diagram. Don't short change on the requires planning effort.

PROJECT ACTIVITIES



6. Buy or develop the best data comm software you can find. It's an important cornerstone to your network.

7. Be prepared to develop your own in-house technical expertise on both the HP3000 and the PCs to create and maintain your network.

8. Stagger the remote site implementations to give yourself opportunities to learn, revise, and improve before your support level becomes to high. For instance, bring up 1 site, then 5, then 20, then 100, with time in between each group to evaluate and make appropriate changes.

SUMMARY

The advent of micro computers has given us the ability to put processing power into the hands of remote PC users. At the same time, MIS has retained the responsibility for the corporate information resource. The issue that continually faces those of us in MIS is how do we provide the access and update capability of the corporate data base to the remote PC user without threatening the integrity of the corporate data base. The answer of course is that only by using the same development methodology and processing controls and standards as we use in developing corporate systems can the full functionality of corporate data access be passed down to the remote PC user. I stress the use of controls and standards because the hardware, software development tools and data communiations software already exist to do the job. All that is needed is the ability to apply these technologies within a constant framework. This, I think, we have done with the development of the Coles Store Information Network.

IMPLEMENTATION OF AN AUTOMATED CODE ENFORCEMENT SYSTEM VIA THE INTEGRATION OF THIRD PARTY AND IN-HOUSE DEVELOPED SOFTWARE IN A MIXED 3rd and 4th GL ENVIRONMENT

Kathleen P. Metz Edwards
City of Plano
Plano, Texas

INTRODUCTION

Before I launch into a discussion on the merging or co-developing of third party and in-house developed software, I will describe the process by which my organization, the City of Plano, Texas, decided to embark on this means of systems development. Following the discussion on why, I will discuss the implementation phase, as well as the benefits to the City and to the contracting third party vendor. Lastly, I will conclude the paper with discussion of the question, "Is it for everyone"?

DEFINITION OF PROJECT ALTERNATIVES

While many users may look at this means of project development as a substitute for good design work, they are mistaken. As with all systems development projects, the design phase is absolutely critical to the ultimate success of the project.

The process of automating the City's permits and inspections began with a definition of the current, albeit manual system. In 1985 the City's Code Enforcement Department manually issued nearly 12,000 building permits and tracked 98,609 supporting inspections to accommodate a 300 percent increase in growth. Permits were issued via eight separate documents and inspections were tracked using 13 different posting cards. The resulting paperwork was time consuming and inefficient. Documents were lost and inquiries were impossible.

An internal review of procedures was jointly performed by the Code Enforcement and Data Processing Departments. The review resulted in the replacement of the permit and posting forms by single forms designed for each function. These forms were implemented nine months prior to the installation of the software. This nine month leeway allowed the users to become accustomed to the new forms, without the added burden of automation. It also defrayed some of the fear of change that was to be expected in a project of this magnitude. In addition to training Code personnel, meetings were also held for developers and contractors as the new forms would drastically alter what they had traditionally become accustomed to. Lastly, the City published documented working procedures in support of the streamlined processing. Construction guides for both residential and commercial projects were also published and released to the public in conjunction with the new forms and procedures.

Data Processing and Code Enforcement were then ready to begin the definition of requirements for an Automated Code Enforcement System (ACES). The requirements were compiled and a Request for Proposal was sent out to prospective vendors. At the same time, Data Processing, with the support of the users, prepared a Detail Design strategy, defining the systems' basic program functionality (attachment 1).

The systems' requirements were divided into five broad functional areas: major file maintenance and report programs, other file maintenance and inquiry programs, other file maintenance report programs, operational report programs and management report programs. Each broad functional area was then further subdivided into programs. Admittedly, we designated these subdivisions as "programs" for purposes of evaluation. In practice, if the City had decided on an in-house solution, a single "program" might have become several actual programs.

Each of the "programs" was assigned a "relative degree of difficulty", i.e., "H" (hard) - 10 days, "M" (medium) - 5 days, and "S" - 2 days (simple). The programs were then factored with the assigned values. Please note that the document was not a Detail Design. It merely identified the functions requiring design work and some "guesstimates" that would be used for comparative purposes. Using this methodology, we estimated we would need 245 days to complete a detail design document.

As a public institution, the City also had the option of reviewing and implementing public domain software. This option was considered in the evaluation process, recognizing that the software would require changes to meet our specific requirements.

The time estimates for the implementation of an in-house developed system and the three modified public domain systems were evaluated via the programming estimates form (attachment 2). A separate programming estimates form was completed for the in-house development option, as well as for the three public domain options. Analysis revealed the City would need an additional 640 days to code and test in-house developed programs. Time to code and test public domain software systems included 447 days, 540 days and 580 days.

Vendor responses to the RFP were scrutinized in a similar manner to determine if the City would incur additional costs. For example, did the vendor include all travel and training expenses in his proposal? Were all reports coded or was the City expected to code them using a report generator or QUERY? Would the vendor provide source code? Was the system written in COBOL, IMAGE and V/PLUS? Would the vendor offer flexibility in maintenance contracts especially during the implementation phase? Lastly, if a vendor omitted any costs, they were added by the City as part of the evaluation process.

The City also applied an additional $34,545 to each alternative, i.e., in-house, public domain or vendor for parallel testing and system documentation.

DECIDING ON A COURSE OF ACTION

A cost analysis was performed on the options described above. The recap figures for the alternatives are attached (attachment 3). The cost of development ranged from $293,205 to $61,545. The City selected Interactive Computer Applications Development (ICAD) of Sarasota, Florida not solely on the basis of cost. ICAD was willing to work with the Data Processing staff to customize the software to meet the City's specifications and to provide flexibility in meeting the City's long term goals.

In addition to the cost analysis, I would like to briefly mention the goals of the organization in evaluating the criteria. Plano is a rapidly growing City. The City has many automation needs that have yet to be defined. While we knew in time the City would implement a parcel/geo-file system as a basis for permits, we simply did not have the time to implement a system of that magnitude - we needed automated permits and inspections yesterday. Therefore we wanted to make sure that the system we implemented would accommodate an interface to a geo-based system. Since we were not clairvoyant, we obviously needed the source code to ensure our ability to interface. Additionally, the City fully intended to assume the long term maintenance of this software.

IMPLEMENTATION

The system we selected met approximately 75 percent of our requirements as demonstrated. Namely, the system issued permits and tracked historical inspection data. The system did not calculate permit fees or provide for on-line inspection requests, two critical requirements. Additionally, the software was built upon a land use parcel system. Since the City was not ready to implement parcel management at the time, it did not wish to pay for software that might never be used. Through discussions, verified by contract, the vendor agreed to modify the software to calculate fees and to process on-line inspection requests. The City agreed to develop two sub-systems, specifically, the Street Index for street verification and the contractor sub-system for registering and licensing contractors/sub-contractors. The Street Index system would be used in lieu of the parcel system. While the vendor's software supported contractor registration, it did not meet Texas legislative nor, Plano's City Council requirements. Lastly, the City agreed to interface the software to its existing systems.

A WORD ABOUT COMMUNICATIONS

I would like to digress here and speak briefly about communications. Before the project was completely implemented, it would touch on personnel in Florida - the selected vendor, Texas - the City of Plano and California - the 4GL vendor. Our contract with the vendor called for only two on-site visits; the costs of additional visits, if needed, would be borne by the City. During the vendor's first visit, we discussed the necessary modifications to meet our requirements. The vendor returned to install his portion of the software on the second, and last visit. All other communications were via the telephone or letter.

The need for good communications cannot be overstated. Coordinating the implementation of the software required many hours in preparation of written correspondence. I firmly believed that projects fail because they lack clear, concise, written communications. Programming has proven to be the smaller part of any project but frequently we have been under pressure to begin coding before questions have been answered and problems resolved.

All successful automated projects have required consistency in the definition of the user's requirements. As I have previously stated, this method of systems development was not a panacea. In any new project, the users needs must be outlined and agreed upon prior to beginning software development. Admittedly, this consistency was easier said then done.

The successful implementation of integrated in-house and vendor developed software was also dependent on the delineation of duties. Each entity understood fully what it was to accomplish to make the project a success. This delineation was also contractually stated, however, if it was not understood, all the legalese in the world would not have gotten the job done.

Lastly, the successful integration of in-house and vendor developed software was facilitated in part by recognizing the need for a single contact point on either side. While several programmer/analysts worked on the project, all questions to the vendor were directed through a single individual. Along the same line, we directed all our questions to a single individual in the vendor's office. This arrangement worked well throughout the implementation process.

The system when it was implemented on November 1, 1986, consisted of 20,245 lines of on-line COBOL, 100 V/PLUS screens, and 39,267 lines of batch COBOL written by the vendor, and 52 Speedware modules written by the City to meet its portion of the agreement. The total elapsed time from the definition of manual permit processing to the implementation of on-line permit processing was 13 months.

OTHER IMPLEMENTATION REQUIREMENTS

Naturally, to be successful in this environment you must have access to an experienced HP programming staff. During the 13 month implementation period, there was at least one, and at times four members of the Data Processing staff working on the project. The staff members worked in the definition phase, as well as in the final implementation phase. Without the expertise of the various staff members, the project could not have been completed within the 13 month time frame.

Simply stated, if you do not have this type of expertise available, you would not be a good candidate for co-development.

INTERFACING VENDOR & IN-HOUSE SOFTWARE

The vendor sent the source code tape to the City several weeks prior to his arrival for installation and training. This lead time allowed the City the time to compile the programs in its own environment. We experienced only one compatibility problem due to a disparity in operating system releases, specifically, the vendor was on a later release of COBOL. The vendor made some minor changes and we were ready to continue.

CONTRACTOR/SUBCONTRACTOR SUB-SYSTEM

At the same time we were defining ACES, the City made the decision to replace its C/PM micro computers with IBM PC clones. Code Enforcement had a simple contractor/sub-contractor system in place on its C/PM micro that needed immediate replacement. Because of the short time frame, we developed a Contractor/Sub-contractor System using Speedware. We originally intended this sub-system to be a temporary one, however, as our evaluation progressed, we realized that the system we developed was preferential to others on the market.

Over time, the contractor sub-system has evolved into a full Contractor/Sub-contractor management information system. In addition to using the system for permit validation, it has also been used for a variety of other functions. Specifically, using Reflections, we have downloaded data to MS Word and sent out contractor renewal notices, newsletters, ordinance changes and a variety of other correspondence. This feature has allowed the user the flexibility to design and run reports independently.

Integration of this system with the vendor's software was accomplished without problem. We simply sent the vendor a copy of the contractor's schema. We also defined the edit criteria that we expected for acceptance of permit applications and for issuance of permits.

STREET INDEX SUB-SYSTEM

The Street Index Sub-system was established to verify streets in the absence of a geo-file. The Street Index sub-system was simply designed to be what its name implies - a listing of street names. We designed the Index to store the streets in geo-file format, i.e, storing street name, street type and direction. This format will allow the City to make the transition to a geo-file/parcel system as resources become available. The "Map Name" that appears on the reverse side of the City's map was also stored in the Street Index. All data elements were designed for maintenance by the Engineering Department.

Since the City did not have the Streets resident on the 3000, we uploaded the names from a Lotus file, formatting the records via Speedware. We also developed the Street Index in Speedware as we had a limited time frame and we knew that the City intended to migrate to a parcel management system in the future. While the Index is a simple "add", "delete", "modify" and "inquiry" type of application, it has served the City quite well. Due to the overlap in school district boundaries, the City had experienced a problem in issuing permits outside it boundaries. The Street Index prohibited this error and provided uniformity in spelling street names.

We experienced no major technical problems in the Street Index integration, however, we encountered a minor problem in the handling of numeric fields in Speedware vs. COBOL. This problem has since been resolved; we circumvented the problem at the time by changing a particular numeric field to an alphanumeric field as the particular application did not absolutely require the use of a numeric field. We were then able to manipulate the data in the receiving COBOL program. As with the Contractor Sub-system, we forwarded a copy of the Street Index schema to the vendor. He made the necessary calls to the Index, and applied the desired edits for street validation.

TAX AND UTILITY BILLING INTERFACES

In order to issue a permit for an existing structure, Code Enforcement needed to validate the structure's existence within the City. While the City's Utility Billing System contained the address information on any structure receiving water in the City, Code Enforcement had no means to access the data. Code Enforcement also needed to determine the parcel's legal description so that the permits could be forwarded to the Central Appraisal District for tax purposes. While the Tax Master held this information, it was not readily available to any other department. Code personnel manually searched appropriate sub-division plat maps and recorded the lot, block and subdivision. This process was time consuming and resulted in permit processing delays.

These interfaces posed quite a challenge as Tax and Utility Billing were driven by disparate keys; the Tax key was a composite of the lot, block and subdivision - the very items that were unknown. Utility Billing's primary key was an account number that had no significance to any other City department. While its alternate key was street address, the streets in the system were not verified upon entry and therefore varied to exact nomenclature. The primary key to ACES was an application/permit number with an alternative street address key in the fixed geo-file format.

We could not seriously consider a conversion of either Utility Billing or Tax in the time frame that we had defined for ACES implementation. We solved the interface problem through the application of Speedware's "Speedex" to both systems. Since neither system had the security for outside inquiry, we built separate "lookup" screens and files for each system, selecting required data elements and applying "Speedex" to owner name and address. Admittedly, this approach required a 50 percent increase in data storage however, it markedly reduced the effort necessary to process permits. Additionally, permit turn around time decreased and data accuracy improved dramatically. The "lookup" files are rebuilt through weekly batch runs. The Utility Billing "lookup" extracted and rebuilt nearly 40,000 active utility accounts, running in less than two hours; the Tax "lookup" extracted and rebuilt nearly 60,000 real property accounts, running in less than three hours. Both these jobs run on the weekends and have exclusive access to the HP3000; if run during normal business hours, run time will increase substantially.

We encountered some minor, but time consuming problems in establishing batch run streams for the "Speedex" "rebuilds". Speedware was designed for on-line processing and its manual provided little in the way of documentation for batch processing. Running the "rebuilds" on-line required the presence of an operator, a costly and in this case, needless expense. These problems were repeated with the installation of Speedware's version 5.0.

"Speedex" allowed the user to obtain required information without knowing the record key. It also allowed the user to process data using a "wild card" character, i.e., if the user knew only a portion of the element to be searched, the "Speedex" software returned records that had matching entries. Specifically, the Tax "lookup" allowed multiple key access via owner, property location or legal description; the Utility Billing "lookup" allowed multiple key access via customer name, property location, driver's license number or social security number. While the "lookups" were designed specifically for Code Enforcement, they are now popularly used by many other City departments that have the need to process owner/ occupant information. These departments include Streets and Traffic, Solid Waste, Planning, Capital Projects, Engineering, Health, Police, etc.

The "lookups" have been further enhanced to allow the users to selectively download names and addresses via Reflections to Micro Soft Word where they have been used for a variety of notification purposes. This feature has saved the City many hours in addressing and preparing correspondence.

As time permits, we will add "Speedex" directly to the Tax master and eliminate the need for the duplicate data. This effort will require some reprogramming to ensure security regarding tax payments. There are no immediate plans to add "Speedex" to the Utility Billing master as the master is a KSAM file; Adding "Speedex" to the UB master would require a migration to IMAGE.

Although "Speedex" has proven to be of great benefit to the City, a word of caution is advised. "Speedex" was not a substitute for quality design. We still needed to strive for logical and optimum key paths. If an item was truly unknown, "Speedex" could help you find it. For example, we directly added "Speedex" to both the Contractor's company name and to the Contractor's owner name several months after we had implemented the Contractor/Sub-contractor System. We discovered that Contractors frequently sent representatives to obtain permits who did not know the Contractor's registration number. This was not surprising as the City did not actually require the Contractors to carry the registration card for permit issuance. We also discovered that representatives did not know the actual registered company name (an alternate key). "Speedex" allowed us to circumvent these problems and located the proper records quickly. We justified the "Speedex" overhead in this case to provide better service to the Citizen. Since the Contractor data base was a relatively small one, the added "Speedex" data sets did not consume a great deal of space.

INTEGRATION OF THE SOFTWARE MODULES

As discussed above, we encountered, no major problems in tying the Street Index and the Contractor sub-system to the vendor's software. The integration of these three modules was controlled through standard calls to IMAGE.

We did encounter a problem in pulling the software together under a single menu. The vendor's software contained a "Main Menu" that was to coordinate all automated Code Enforcement processes. We had the menu set up to call the Speedware modules when requested. For example, if the user wished to register a new contractor in the Contractor/Sub-contractor (Speedware) system he would enter the proper option in the V/PLUS controlled menu and depress ENTER. The on-line COBOL program would then execute the proper call to Speedware and return with the desired Speedware sub-menu.

We struggled initially to make the call successful. We encountered a problem with bounds violations; the problem was solved when we determined the correct account capabilities and applied them as necessary. We encountered a second problem with Speedware loosing a temporary file designation. The problem was solved via resetting the file designation in the specification file just prior to exiting Speedware to return to COBOL.

While we solved the problems in the COBOL/Speedware interface, we discovered the call required 15 - 25 seconds to complete. Over time, the users found the delay unacceptable. The timing problem was resolved when we added a Speedware "Main Menu" as the driver, and allowed Speedware to call COBOL. Although we never performed an in-depth analysis of why the Speedware to COBOL required less time, we conjectured that each time COBOL called Speedware, Speedware had to reinitialize its overhead. Apparently, if the system was designed with Speedware as the controller, the overhead would be executed a single time, with the initial call to Speedware.

Infocentre did not encourage the COBOL to Speedware interface. In fact, the Speedware manual contained no instructions on how to effect the call. The manual did however, contain instructions for a Speedware to COBOL interface.

BENEFITS TO THE ORGANIZATION

The primary benefit to the City was increased speed of implementation. As described above, we had calculated 245 days to complete a detailed design document and calculated an additional 645 days to code, test and document the software. We needed 3.4 man years to complete the project and we simply did not have the time.

Secondly, the approximate cost of in-house development was calculated at $159,330. The projected cost of a co-developed project was approximately $61,545. This figure included the actual salary costs for the Data Processing personnel during the implementation period. The City saved an estimated $97,785.

A third benefit to this type of arrangement, was reduced long term cost to the City for maintenance. While in-house maintenance required in-house staff, it did not require the monthly cash outlay to a third party vendor. While organizations have paid monthly maintenance fees to outside vendors for software usage, these fees do not negate the need to have staff on hand to support the software releases. In theory, support for software releases should require less time than in-house maintenance however, many programmer/analysts earn their living doing just that.

A fourth and perhaps most important benefit to the City was the increased flexibility this type of arrangement allowed.

As contractually stated, the City received the vendor's source code. The City did not have the right to sell the code, nor could it be given away under the purview of public domain. However, after a six month warranty period, the City had the right to make any modifications or enhancements that it deemed necessary.

ACES has by no means remained static. As of March 31, 1988, 17 months after implementation, ACES programming statistics have increased significantly. The City has added 1000 lines in on-line programming, yielding a total of 21,245 lines of on-line code. 38,332 lines of batch COBOL programming have been added, yielding a total of 77,599 lines of batch code. 29 Speedware modules have also been added, yielding a total of 81 Speedware modules.

The City has retained COBOL as the batch reporting language as the added programs handle data in much the same manner as the original 20 supplied by the vendor. A major sub-system supporting Zoning Enforcement was developed during this time period and accounts for most of the added Speedware modules.

A fifth benefit to the City in this type of arrangement was the sense of security that source code provided. The City received four bid proposals in response to the RFP. The high bid provided for a vendor to design and implement a customized Code Enforcement system. The next two bids were submitted by vendors that no longer exist under the same legal identity as they did when the bids were submitted. If the City had accepted either bid, a new contract would have been necessary. A renegotiated contract might not have been favorable to the City. The low and selected bid provided the City an excellent system as well as the source code for future growth and development.

Lastly, the City has derived great benefit from the Tax and Utility Billing "lookups". Few departments in the City do not use one or the other of these modules. As a result of their popularity, the City will be able to justify the cost of reformatting the Tax and Utility Billing keys to provide automatic access through a common key, namely street address. This conversion will eliminate rekeying of needed data. The conversion will also serve a second goal of preparing both Utility Billing and Tax for interface to the postal tape, allowing the City to implement zip plus four.

The reformatting of Tax addresses will be a step towards the City's goal of a parcel management system. Additionally, Code Enforcement has requested support for the storage of permanent parcel data in the forthcoming fiscal year. A permanent parcel data set, combined with a reformatted Tax Master, will provide the City with a geo-base cornerstone.

BENEFITS TO THE VENDOR

While the benefits of co-development to the City entity were somewhat obvious, the vendor also derived benefit from this type of arrangement. The major benefit to the vendor was reduced development costs. The City purchased software that had been running in test mode at another site. The vendor programmed and unit tested the contracted enhancements. However the City, as the first live site, worked with the vendor to correct any problems and implemented the software into production.

A second benefit to the vendor was another happy customer. While Plano's situation was somewhat unique, I am sure we were not the first HP site that did not wish to enter into a long term maintenance agreement with an outside vendor. We have been very satisfied with our arrangement and when asked, we will provide an excellent vendor's reference.

Lastly, the vendor's product was enhanced considerably during the implementation process. On-line inspection requests, and automatic calculation of fees were missing from the system that was originally demonstrated. While the vendor programmed and tested these changes, the City had defined a need that would be of benefit to any governing body issuing building permits. The vendor was now free to market these enhancements.

IS IT FOR EVERYONE?

The long and short terms goals of the organization need to be evaluated in answering this question. If the contracting entity did not have a trained HP staff and had no intention of hiring such a staff, then this method of development would not be appropriate. However, if the entity wished flexibility to grow and to add to its systems in a logical and productive environment, then this method would be of obvious benefit.

In conjunction with the organization's goals, would it be willing to provide the programming and analysis time that was obviously required in this type of development? Approximately 13 man months of Data Processing support were required during the 13 month period. At one point during the project's development, four members of the Data Processing staff were dedicated to the implementation. This was a large commitment of resources, requiring total management support.

Secondly, integration of third party and in-house developed software assumed that a third party had the software available to meet a reasonable percentage of the entity's needs. If the software was not developed, would the vendor add the code to bring it to a satisfactory level? This was a tough question as it involved judgment and compromise so that a reasonable solution could be met. Resolving this issue could also cost money and a decision would have to be made as to who would pay the costs.

The third question that had to be answered was, could the entity find a vendor willing to engage in a joint development process? Part of the answer to this question goes back to my discussion on communication. Correspondingly, the vendor might have viewed this type of development as too expensive. In a large software development environment, the vendor must sell many systems to remain profitable. Many vendors consider customization too costly.

Lastly, could the contracting bodies resolve the legal issues arising in a co-development environment? Naturally, the vendor wished to protect his investment in systems development. He obviously wanted to retain the sales rights to his product. Also, would he be willing to relinquish the maintenance fees frequently associated with purchased software? On the other hand, could the contracting agency provide the assurance to the vendor that it would protect his rights as well?

IN SUMMARY ...

In writing this paper on the integration of third party and in-house developed software, I have merely described how this method of development worked in my organization. The success of this method is dependent on many factors, only a few of which have been noted. Each organization is unique, complete with its own set of idiosyncrasies. Suffice to say, that in the proper environment, co-development can be a cost effective and productive process.

ATTACHMENT 1
DETAIL DESIGN STRATEGY

MAJOR FILE MAINTENANCE AND REPORT PROGRAMS	RELATIVE DEGREE OF DIFFICULTY
MAIN MENU	H
APPLICATION DATA ENTRY VERIFICATION & UPDATE	H
PLANS EXAMINING ENTRY VERIFICATION & UPDATE	H
APPLICATION INQUIRY	H
PERMIT DATA ENTRY VERIFICATION & UPDATE	H
PERMIT INQ	H
CONTRACTOR PERMIT INQ	H
CONTRACTOR INQ	M
INSPECTION REQUEST	M
INSPECTION REQUEST PRINT & RPT	M
INSPECTION POSTING	M
DAILY REPORT OF INSPECTIONS MADE	M
PERMIT FINALIZATION	H
OTHER FILE MAINTENANCE AND INQUIRY PROGRAMS	
FEE SCHEDULE FILE MAINT & INQ	H
PERMIT TYPE FILE MAINT & INQ	M
CLASS OF WRK FILE MAINT & INQUIRY	M
TYPE USE FILE MAINT & INQ	M
INSPECTION TYPE FILE MAINT & INQ	M
PERMIT STATUS FILE MAINT & INQ	M
PLANO STREET INDEX FILE MAINT & INQ	H
INSPECTOR NUMBER/NAME FILE MAINT & INQ	M
OFFICE PERSONNEL NUMBER/NAME FILE MAINT & INQ	M
INSPECTION STATUS FILE MAINT & INQ	M
OTHER FILE MAINTENANCE REPORT PROGRAMS	
FEE SCHEDULE RPT	S
PERMIT TYPE RPT	S
CLASS OF WORK RPT	S
TYPE USE RPT	S
INSPECTION TYPE RPT	S
PERMIT STATUS RPT	S
PLANO STREET INDEX RPT	S
INSPECTION NUMBER/NAME RPT	S
OFFICE PERSONNEL NUMBER/NAME RPT	S
INSPECTION STATUS RPT	S

OPERATIONAL REPORT PROGRAM

DAILY PERMIT DETAIL RPT	M
WEEKLY PERMIT DETAIL	M
MONTHLY RPT OF INACTIVE & FINALED PERMITS	M
ANNUAL REPORT OF ARCHIVED PERMITS RPT	H

MANAGEMENT REPORT PROGRAMS

MONTHLY PERMIT ACTIVITY RPT	M
MONTHLY CENSUS RPT	M
DAILY RPT OF INSPECTIONS MADE	M
MONTHLY SUMMARY OF INSPECTIONS MADE BY INSPECTOR	M
MONTHLY SUMMARY OF INSPECTIONS MADE BY DISTRICT	M
MONTHLY RPT OF COMMERCIAL NEW CONSTRUCTION & APARTMENTS	M
MONTHLY RPT OF APPLICATIONS SUBMITTED W/O AN ADDRESS	M

```
TOTAL # OF "H" (HARD)   = 11 X 10    110
TOTAL # OF "M" (MEDIUM) = 23 X  5    115
TOTAL # OF "S" (SIMPLE) = 10 X  2     20
                                     ---
                                     245   TOTAL NUMBER
                                           OF DAYS FOR
                                           DETAIL DESIGN
```

ATTACHMENT 2

SYSTEM: CITY OF PLANO

PROGRAMMING ESTIMATES

FNCT	EASY	TIME	OLD	NEW	LANG	** MAJOR FILE MAINTENANCE
	TO	REQ.	PGMS	PGMS	USED	& REPORT PROGRAMS **
	DO					
Y/N	Y/N	DAYS	#	#	C/S	
N	N	15	-	1	C	MAIN MENU
N	N	30	-	1	C	APPLICATION DATA ENTRY VERIF/UPD
N	N	30	-	1	C	PLANS EXAMINING ENTRY VERIF/UPD
N	N	15	-	1	C	APPLICATION INQ
N	N	30	-	1	C	PERMIT DATA ENTRY VERIF/UPD
N	N	15	-	1	C	PERMIT INQ
N	N	15	-	1	C	CONTRACTOR PERMIT INQ
N	N	10	-	1	C	CONTRACTOR INQ
N	N	10	-	1	C	INSPECTION REQUEST
N	N	10	-	1	C	INSPECTION REQ PRINT/RPT
N	N	10	-	1	C	INSPECTION POSTING
N	N	10	-	1	C	DAILY RPT OF INSPECTIONS MADE
N	N	15	-	1	C	PERMIT FINALIZATION
=========================			**OTHER FILE MAINT AND INQUIRY PGMS.**			
N	N	30	-	1	C	FEE SCHEDULE FILE MNT & INQ
N	Y	5	-	1	S	PERMIT TYPE FILE MNT & INQ
N	Y	5	-	1	S	CLASS OF WORK FILE MNT & INQ
N	Y	5	-	1	S	TYPE USE FILE MNT & INQ
N	Y	5	-	1	S	INSPECTION TYPE FILE MNT/INQ
N	Y	5	-	1	S	PERMIT STATUS FILE MNT/INQ

FNCT	EASY	TIME	OLD	NEW	LANG	** OTHER FILE MAINTENANCE
	TO	REQ.	PGMS	PGMS	USED	REPORT PROGRAMS **
	DO					
Y/N	Y/N	DAYS	#	#	C/S	
N	Y	15	-	1	S	PLANO STREET INDEX FILE MNT/INQ
N	Y	5	-	1	S	INSPECTOR NAME/NO FILE MNT/INQ
N	Y	5	-	1	S	OFC. PERSNEL NAME/NO. FILE MNT/INQ
N	Y	5	-	1	S	INSPECTION STATUS FILE MNT/INQ
N	N	15	-	1	C	FEE SCHEDULE REPORT
N	Y	5	-	1	S	PERMIT TYPE REPORT
N	Y	5	-	1	S	CLASS OF WORK REPORT
N	Y	5	-	1	S	TYPE USE REPORT
N	Y	5	-	1	S	INSPECTION TYPE REPORT
N	Y	5	-	1	S	PERMIT STATUS REPORT
N	Y	30	-	5	S	PLANO STREET INDEX
N	Y	5	-	1	S	INSPECTION NAME/NO. REPORT
N	Y	5	-	1	S	OFFICE PERSONNEL NAME/NO. REPORT
N	Y	5	-	1	S	INSPECTION STATUS REPORT
==========================			**OPERATIONAL REPORT PGM.**			
N	N	15	-	1	C	DAILY PERMIT DETAIL REPORT
N	N	15	-	1	C	WEEKLY PERMIT DETAIL PERMIT
N	N	15	-	1	C	MTHLY RPT OF INACTIVE/FINAL PRMTS
N	N	15	-	1	C	ANNUAL RPT OF ARCHIVED PRMTS RPT

FNCT	EASY TO DO	TIME REQ.	OLD PGMS	NEW PGMS	LANG USED	** OTHER FILE MAINTENANCE REPORT PROGRAMS **
Y/N	Y/N	DAYS	#	#	C/S	
==========================			**MANAGEMENT REPORT PGMS.**			
N	Y	5	-	1	S	MTHLY PERMIT ACTIVITY REPT
N	Y	5	-	1	S	MTHLY CENSUS REPT
N	Y	5	-	1	S	DAILY REPT OF INSPECTIONS MADE
N	Y	5	-	1	S	MTHLY SUMRY OF INSPS MADE BY INSP
N	Y	5	-	1	S	MTHLY SUMRY OF INSPS MADE BY DIST
N	Y	5	-	1	S	MTHLY RPT OF CMRCL NEW CONST & APTS
N	Y	5	-	1	S	MTHLY RPT OF APPS SUBMITTED W/O ADDRESSES

NUMBER OF DAYS NEEDED (AT 6 EFFECTIVE HRS PER DAY): 640

PROGRAM COUNT:

NEW COBOL PROGRAMS TO WRITE:	19
NEW SPEEDWARE PROGRAMS TO WRITE: (THESE ARE LOGICAL, NOT PHYSICAL PROGRAMS)	29

ATTACHMENT 3

RECAP OF ALTERNATIVES

OPTION	TIME MAN YEARS	TOTAL COST
VENDOR A	N/A *	$293,205
IN-HOUSE DEVELOPMENT	3.4	$159,330
PUBLIC DOMAIN A	3.1	$150,870
PUBLIC DOMAIN B	3.0	$145,230
PUBLIC DOMAIN C	2.6	$139,617
VENDOR B	N/A	$116,870
VENDOR C	N/A	$ 68,045
SELECTED VENDOR	N/A	$ 61,545

* implementation times were not calculated for vendor options as times would vary by contract.

DISAPPEARING DIAL-UP

James D. Ham
Southeastern Public Service Authority
of Virginia
723 Woodlake Drive
P. O. Box 1346
Chesapeake, Virginia

Are you planning to move your data processing center? No doubt you have ordered the raised floor, halogen fire suppression system, environmental unit, patch panel, ups, isolator/regulator, vacuum cleaner, and microwave oven, but have you installed and tested the new dial-up service from your local telephone company? This simple "standard" element of your system has tremendous problem potential. Our recent experience with a deteriorating communications network may help you to avoid the pitfalls which we leaped into.

Our organization has a dial-up network of HP150 microcomputers located at widely scattered transfer sites. Each of these 150s call our HP3000 twice each night: once to upload the day's transactions then later to download a newly updated customer file. We use ADVANCELINK as our communication software. Recently, after much preparation, we moved our data center to an adjacent municipality. Prior to the move, our electrical engineer checked the power supply and the environmental systems, and we had verified that the new dial-up lines were live, (ie: we had used a telephone to make calls to and from the new site.

The move was accomplished, and the computer was set up and operational the same day. Our administrative users, who did not move with us, were up and running at 9600 baud via leased line and 8 channel muxes that same day. The first week the success rate of uploads and downloads on the dial-up network slipped from its' previous high percentage, but we did not become alarmed. There were many other issues related to the move occupying us, and we were accustomed to missing a site now and then due to excessive line noise, cut cables, or operator error. However, after two weeks the success rate dropped rapidly, and by the end of six weeks was around ten percent!

The symptoms varied, but the essence was that the HP150's could not maintain the line connection long enough to complete the data transmission. Most of the time the ADVANCELINK error message was "connection failed". Occasionally we would get a "no carrier" message, or the transmission would stop mid-file with the 150 just sitting there and the modem at the HP3000 remaining in a busy state.

A data analyzer could have shown us just what was coming across the line, but as we were not experiencing bad data, just no data, we did not try to obtain one. Well, intuition told us that the problem had to lie with the new telephone lines. They and the AC power supply were the only new elements in the system, and the AC had checked okay. However, after the telephone company technician tested our lines, he informed us that our lines were within specifications for "voice" grade service (dial-up) and there was nothing he could do. We checked with the phone company business office and received the same response. There was nothing they could do to improve our service.

Since relief from the phone company did not appear imminent, and the data had to be moved daily, we devised a three part plan to attack the problem: (1) implement an alternative method of moving the data for immediate relief, (2) find a way to use the dial-up lines as they were for the short haul, and (3) pursue with determination convincing the phone company to improve our service as a permanent solution.

In our case, the alternative to the telephone lines was a "sneaker net" with 3.5" diskettes as the medium. Our courier visited all but two of the sites daily, and transfer trucks visiting those two sites could deliver diskettes to a common location where the courier could pick them up. We quickly produced operating procedures for the diskette upload and download processes and installed corresponding command files on the remote computers. One problem with this solution was that the sites presented a hostile environment to the 150's, and some of the diskette drives were no longer operable. Success with the telephone network had made the expense of maintaining the diskette drives no longer seem necessary.

Our bridges hadn't been burned, just allowed to decay! Replacement and repair of the drives was accomplished as rapidly as possible. In the interim, some sites had to be visited daily to collect the previous days' transactions. A fixed drive was connected to the transfer sites' 150 and the transactions were copied from their fixed disc to the portable drive. The collected data was then uploadaed to the HP3000 back at the Data Center.

In trying to make the degraded phone service work we discovered that if the transfer site operator executed ADVANCELINK and typed in the telephone number, instead of letting a command file establish connection, we could get a 20 to 30 percent connect rate. Once connected, the file transfer usually completed successfully. This confirmed our belief that our hardware and software were not the source of these problems. This operator intervention required us to transmit during working hours, which often resulted in customers waiting in line at the site. Still, it was better than driving to the site with a fixed drive! Trial and error in modifying the command files disclosed that eliminating having the 150 wait for the HP3000 to respond with the terminator character (ctrl/Q) and slowing the 150s down improved the transmission success rate to about 60 percent and made daytime transmission unnecessary. Pauses were inserted after commands such as "HELLO" and "BYE", extra newlines sent to the 3000 before and after "HELLO", "&DSCOPY", "BYE", etc., and the character delay times increased to achieve the improvement.

The third area of effort was directed at the telephone company. We called the business repair office repeatedly with complaints of a degradation of service compared with that at our previous location. Their technician was called back to check our lines again, with the same results. A former telephone company employee had informed us that different levels of service existed for each line type, and suggested that we inquire as to our service level. Oh yes, we had consulted many persons and organizations during the course of the problem. Part of our problem solving procedure is to search the available experience bank. While doing so we received one solid lead, much sympathy, and two offers of a complete communications evaluation; for a fee. We followed the lead and escalated our complaints to the manager of the local central office with requests for a service upgrade. Suggestions by the phone company that we install leased data circuits were rejected with the insistent request that they provide us dial-up service equal to that which we had previously. We did not want the monthly expense of multiple data circuits, or the vulnerability of one multipoint circuit linking sites in eight cities and counties and crossing two telephone companies at that time. In addition, all of our modems were dial-up and would have to have been replaced with non-dialing units. By this time the telephone companys' business office, maintenance office, engineering section, and public relations office had all been brought into the discussions with our organization.

In support of our efforts with the phone company, and in devising workarounds, we went through the standard problem identification steps, evaluating the hardware, software, procedures, and environment. One element at a time every link in the dial-up network except the HP3000 was replaced. On our order, the phone company installed one of their data jacks on our line, with no results. From the phone company entry panel, we ran new twisted pair to the computer room via a different route, replaced the remote modem, the local modem, the modem and telephone cables at each end, and replaced the HP150. We also tried different versions and different copies of ADVANCELINK and the ADVANCELINK command files. We discovered that dialing any other computer from our remote sites worked fine, and that dialing out on one of our new lines and back in on the other compounded the problem so badly that we could not use an HP150 in the data center to test modems, etc. Testing had to be done from a remote site. Of course, all of these efforts confirmed our belief that the dial-up service was the problem, but more importantly, the tests supported our negotiations with the phone company, and assured management that every avenue of relief was being explored. Some improvement in connectivity was gained by using the same make of modem at the host site and the remote site. Previously we had not done this. We also found that MNP error correction modems and modems with adaptive equalization will not solve all ommunications line problems. Modems with these features were obtained on approval from vendors for some of our tests.

The phone company had measured our lines several times and reported to us that the decibel (power) level and slope (decibel loss), while within their "voice" grade specs, definitely would not support "high speed" data transfer, (1200 baud). We were at the end of that particular service line, the wires were old, etc. They also told us that there was no tariff within their rate structure which would allow them to upgrade our service. Our next step was to request that they quote us a price for upgrading the equipment or devising a new tariff and to let them know that we were willing to bear the cost of correcting our problem. Persistence pays! After five months of discussion at various levels between our organization and the phone company they announced that they were going to fix the problem, and they did, within two weeks! In essence, the phone company solved our problems by installing MFT's, (metal frame terminators), on our lines in their central office. An MFT increases line frequency. We had been asking whether MFT's might help our situation since learning of their existence from their former employee. A simple solution at the end of a complex path. No special charges or rate changes were levied against us by the phone company!

In conclusion I would suggest that: (1) New dial-up service be thoroughly tested with your production configuration, or as close as you can manage, in advance of the move. We may have discovered our problem sooner if we had taken a 150 to the new center and tried communicating with a remote site. If your new service will not be ready prior to the move it may be feasible to test the phone service from another organization located near your new data center and on the same service line. (2) Try tweaking the communications software or command files while waiting for the phone company to correct the situation. Some service was better than no service in our case. And (3) Comprehensive testing, and presenting an organized case are helpful in dealing with the telephone company bureaucracy.

THE EVOLVING NETWORK

A. Jay Gross
Product Manager, Multiplexers
Paradyne Corporation
(813) 530-2785

Paper No. 0051

THE EVOLVING NETWORK

The ideal data communications network is invisible to the user and appears to be no more than a piece of wire or direct connection between the originating and destination points. And like the local electric power company, it should be so reliable and easy to use that the user takes it for granted. Today, networks have the ability to approach this ideal. The array of existing tools and emerging technology makes this possible. In many respects, the future is here today.

The beginning of modern data communications can be traced back to the 50's. It was then that American business and industry began a trend towards decentralization that created new problems and set the stage for the migration of centralized data bases out to remote locations. It became apparent that new technology would be required to transfer information from one location to another. The pressure was on for business equipment manufacturers to develop methods and systems for moving the massive volumes of information being generated. Solutions to the communications bottleneck had to be found.

The modem, which became the foundation for an entire new industry, ushered in the era of data communications in the 60's and distributed processing in the 70's. Most early concerns were easily addressed even though available options were few. Simple point-to-point and multidrop configurations predominated. Most applications required less than 2400 bps with 110 bps and 300 bps dial being used extensively into the mid and even late 70's.

The network planner's goal for providing the user a cost-effective and efficient data communications system was always met. After all, as traffic and locations grew, users could add more cheap lines and drops, along with modems that had started decreasing in price. In many cases, the best use of resources or most effective network design had very little to do with actual system implementation. Multiple lines to single locations were common.

Interestingly, at that time network control and diagnostics were considered to be of little value. By process of elimination, a problem usually could be narrowed down to one of two network components--lines or modems. Besides, no more than two points of contact for service were usually required - the telephone company or the modem vendor. Many times, one call to Telco took care of everything since both the modems and the lines were leased from them.

The good old days - less demand from users, reasonable costs, few products and services to deal with, simple problem determination and resolution. Uncomplicated network solutions that worked!

Well, the 80's changed all of that. It seems that almost overnight, users had more applications than ever, and of course, they all required better response times. Remember what happened when everyone bought a personal computer and wanted it on line.

Or, when the local telephone company increased their access charges dramatically, as well as their lead times for lines. And loss of a single point of contact for service made life unbearable as users screamed about uptime or the lack of it! Increased product offerings from multiple vendors became a double-edge sword. Often, older equipment was not compatible with newer technologies. And, of course, service offerings from the Telco's based on voice standards only added to both the confusion and frustrations. The network could grow no further using existing hardware.

The communications evolution has now reached its next logical step. The changing technology, regulations, pricing and user requirements are the causes of both today's problems and opportunities in network design. We have entered an era where the network has become as critical as the information source itself! Many businesses in our increasingly service-oriented society have discovered that the network is the tool that gives them the edge over competition. Most have discovered the time value of information.

Attention must be given to understanding the nature of the network from a business perspective - the products, markets, competition, and underlying management philosophy. Ask where did we come from, where are we today, and where are we going. Thus, as planners look toward growing the network, their goals must be to:

- decrease costs
- increase profitability
- increase productivity
- increase network availability
- seek new markets and business opportunities

Modems alone can no longer meet these goals, although they continue to play an important role in the network of the 80's. The network planner must now seek consolidation by acquiring technology that will add value in terms of both cost savings and increased function to existing systems. Thus, the trend is towards consolidation and integration (figure 1). A broad group of devices has emerged as a powerful way to meet these objectives: multiplexers, concentrators, and nodal processors.

Their primary function is to concentrate a large number of low-speed incoming lines onto one or more high-speed transmission facilities. This functionality, coupled with highlevel, built-in intelligence, not only reduces line costs but provides a uniform means of dealing with the total communications requirements of the organization. Enhanced service offerings, increased network availability, improved diagnostics, and a migration path for future applications are only a few of the benefits.

Simple, low-end concentrators include modem-sharing devices, point-to-point Statistical Time Division Multiplexers, telephone central office D4 channel banks, and point-to-point T1 multiplexers. Advanced designs range from networking statistical multiplexers and T1's to packet switches. These intelligent multiplexers are usually referred to as communications processors or nodal processors.

Unifying network architecture through the use of multiplexers takes the comprehensive approach to consolidation and integration. But, implementation of such a design must be well thought out. In addition to the business goals just mentioned, there are obviously technological issues. Everything from type and volume of traffic to applications, network control, and geography must be taken into account.

Three basic technologies have emerged with capabilities that can meet the needs of different situations. Statistical multiplexers, packet switching, and T1 are the choices to be considered.

Statistical multiplexing is used primarily in situations where asynchronous communications predominate. Because this method dynamically allocates bandwidth as needed, the short character-at-a-time transmission used in asynchronous communications can produce tremendous efficiencies. The ratio of aggregate channel speeds to the link speed can easily be 10-to-1 or greater. In other words, ten devices of 4800 bps each, for a total aggregate 48,000 bps, could be statistically multiplexed over a link running at 4800 bps (figure 2). Not only is the cost of nine lines saved, but the speed of the modems used between the two multiplexers is much less than one might expect!

This is not to say that the laws of physics are altered. Instead, a statistical multiplexer takes advantage of the fact that all of these devices probably won't be transmitting at the exact same moment in time. Even in a "heads down" order entry application, an asynchronous terminal is sending data only about 10 or 15 percent of the time during an eight-hour day. Besides, how many operators do you know that can type at 4800 bps? That would be the equivalent of about 36,000 words per minute!

To look at it another way, if each operator were typing at 40 words per minute, which is about 5 bps, 900 devices transmitting simultaneously would be needed to fill up 4800 bps worth of bandwidth! Of course, there are other considerations which, from a practical point of view, make this compaction ratio much higher than we could really obtain. The point is, all users are virtually guaranteed a time slot on demand. That's why the apparent throughput in this example is 48,000 bps even though the real rate of the link is only 4800 bps.

Here's how its done. The data from each device forms a frame that is sent to the remote statistical multiplexer. Included with this frame is additional information for error control, addressing, flow control and signaling. Most multiplexers use an international standard protocol for framing called HDLC (high-level data link control). This is the same transport protocol used in X.25 packet switching networks (figure 3).

Each frame can contain up to 256 bytes of information, although most statistical multiplexer vendors use 128 bytes. Using this technique, all users will get at least several bytes of data into a given frame if requested. If there is not enough data to fill up the 128 bytes, the frame will automatically adjust down. In our example, the ten terminals will have at most 40 bytes per minute times 10 devices, divided by 60, or a total of about seven bytes of data to send per frame.

As efficient as this is, another 20% can be gained by stripping the start, stop, and parity bits of the asynchronous data at the sending statistical multiplexer and then adding them back at the receiving end.

Another important feature of a statistical multiplexer is its ability to buffer data. Should all operators manage to hit a key at the exact same moment in time, data will not be lost. A buffer size of 4 to 16K will usually suffice, depending on the number of terminals and their respective speeds. This situation will change as printers and batch devices are added. Compaction ratios of 10-to-1 are no longer valid. Instead, a more adequate rule of thumb would be ratios of 4-to-1 for printers (up to a few pages at a time) and 2-to-1 for batch devices.

Instead of increasing the buffer size in order to handle the demand created by these bandwidth hogs, flow control is used. When about 80% of the buffer is full, a command is issued by the statistical multiplexer to both the local DTE's and the remote statistical multiplexer. The remote multiplexer will, in turn, signal the sending device to stop sending data. The most common method of doing this is an in-band scheme known as Xon/Xoff. As the name implies, the proper software flow control signal is sent to start and stop normal data flow from the originating device to its destination. Xon will usually be given when the buffer empties to about 40% of its capacity. Because this activity is in-band and automatic, the operator will have no knowledge of the occurrence.

Hardware flow control can also be used. EIA control signals such as Clear to Send (CTS) and Data Set Ready (DSR) are used.

As most users already know, there is not much in the way of error control in the asynchronous world. At least not until statistical multiplexers arrived on the scene. Like most synchronous protocols, HDLC uses a redundancy check algorithm in conjunction with an Automatic Request for Repeat (ARQ) for end-to-end error control. So, not only is every device virtually guaranteed a time slot, users can count on the information arriving at its destination error free!

Today's network planner can utilize the intelligence inherent in a statistical multiplexer to take this one step further. A comprehensive network solution can be implemented by using building blocks available from one of a handful of networking statistical multiplexer vendors. From four channels to 240 channels, an integrated family of products will allow an asynchronous network of any size to be designed. Some vendors have even integrated multiplexers into their network management and control centers. Now a single point of control exists for multiplexers, modems, and DDS equipment such as DSU/CSU's (figure 4). The ultimate approach for consolidation and control of the asynchronous network would certainly seem to be statistical multiplexing.

What happens when we consider synchronous data? In a mostly asynchronous environment with some synchronous traffic, today's statistical multiplexers are the way to go. But, as the percentage of synchronous traffic approaches 20%, alternatives should be considered. And, when this number exceeds 40%, the statistical multiplexer becomes inadequate very quickly. Packet switching may be a better answer.

A packet switch is no more than a very sophisticated statistical multiplexer. As stated earlier, both use HDLC as their protocol which is considered to be the transport layer of X.25, known as X.25 level II. Packet switching utilizes the next layer which is X.25 level III. In addition to the HDLC protocol, this level contains an advanced addressing structure resulting in only one channel being assigned to one frame. The result is block oriented (figure 5).

This multilayered addressing structure, combined with the enormous power built into packet switches, gives it the ability to support diverse vendor-specific protocols. Thus, packet switching seems to better satisfy synchronous network design applications, particularly large networks with multiple synchronous protocols.

Packet switching also differs from statistical multiplexing in its hardware makeup. There are packet nodes and packet assemblers/ disassemblers (PADs).

The node forms the network backbone. Its software enables the node to communicate with adjoining nodes so that traffic information is constantly exchanged and updated. In other words, it acts as a trunk-interfacing tandem switch. Like a telephone company central office, call requests are properly directed to the destination node over the path with the shortest delay.

The PAD connects remote terminals directly into the network nodes. Typically, they will packetize several devices of a specific protocol into a single X.25 network interface. The PAD can be located locally to the node or remotely via a pair of modems and a dedicated line. At the host site, a PAD can be installed so that, in effect, the packet network becomes transparent by interfacing at the port level, or instead, one of the node trunks could support a connection to a single host port configured to de-mux multiple virtual circuits at the X.25 level (figure 6).

Another important feature of packet switching is the sophisticated network management tools available. In addition to network control and diagnostics, a packet network control center can provide capacity management as well as usage accounting and billing.

Capacity management ensures that packets are sent to the receiving party by the most efficient available route, based on network traffic at the time of call setup. This operation automatically smooths peak load that otherwise would require additional capacity.

Usage information can be important to organizations that charge network time to departmental cost centers or customers. The advantages of packet switching are becoming clear. Its ability to provide a vendor independent solution for large diverse networks is the key. Packet thrives on multiple hosts, protocols, applications, and locations.

The third area of multiplexer or concentrator technology is T1.

In recent years, the term T1 has been used to refer to any digital arrangement operating at 1.544 Mbps. T1 has been used in this country for almost 25 years, yet for many, it continues to be a vague and mysterious technology. And, indications are that T1's emergence as the transport technology of the future is only now beginning.

The first T-carrier facility was introduced into the otherwise analog telephone network in the early 60's to interconnect central offices. By digitizing voice signals and multiplexing them using time-division techniques, the T1 link permitted 24 voice-frequency channels to be carried over just two pairs of wires. The first generation of T1 central office termination equipment was the D1 channel bank.

By the late 70's, this evolved into the D4 channel bank. Although voice digitizing and channelling techniques had changed, basic technology remained the same; a very large time division multiplexer (TDM) with 24 channels and a link speed of 1.544 Mbps (figure 7a).

In 1984, several things happened to change the way T1 was perceived and utilized. First, AT&T made T1 more accessible by increasing the number of facilities installed and lowering the cost. This became known as Accunet 1.5. With the advent of divestiture, other carriers, including the Regional Bell Operating Companies (RBOCs) increased competition and availability by installing even more high-speed circuits, much of it being fiber.

It was at this point that vendors began to see the real potential T1 offered for voice and data integration. An updated version of the D4 channel bank was introduced--second generation equipment simply known as T1 multiplexers. By adding microprocessors for intelligence, the bandwidth could be subdivided into more than 24 channels. Supervisory capabilities were added and limited networking was achieved (figure 7b). At about this same time, the idea of a voice and data integrated services digital network (ISDN) started to emerge with T1 playing a key role.

What seemed like a stable and known technology with interface and framing specifications cast in concrete suddenly became complex and overwhelming primarily due to a lack of understanding in the data communication marketplace. Even a fundamental grasp of these concepts will lead to advanced networks able to take advantage of T1 economies of scale and newer equipment designs.

What does it mean to be T1 compatible? Today, the answer is not a simple one, as there are different levels and framing formats. Obviously, to start with, the multiplexer must at least transmit and receive a signal at 1.544 Mbps. This is DS-1 compatibility and includes electrical characteristics.

When a DS-1 signal is used for D4 service, it consists of frames of 193 bits each. This represents an 8-bit byte for each of the 24 sub-channels, plus an extra bit for framing. Sampled at 8,000 times/second, each of these 24 slots represents 64 Kbps of bandwidth or a total of 1.536 Mbps of usable bandwidth. The 193rd bit, also sampled at 8,000 times a second, accounts for the remaining 8 Kbps of the total 1.544 Mbps bandwidth. Twelve of these 193 bit frames are known as D4 superframes and represent the framing level of compatibility. (See Table 1)

The next level is D4 Channelization, required of T1 circuits that terminate at the central office. A DS-1 signal made up of 24 subchannels, each taking 8 bits at a time, is now needed. These are numbered DS-0 1 through DS-0 24. For premises-to-premises transmission, the user must only maintain the 193rd bit pattern--everything else is transparent.

The third and final level is signaling compatibility. Here, the 6th and 12th frame of every superframe have the 8th bit of the user's information robbed. These signaling bits are used for basic telephone control, such as on-hook/off-hook indications.

It is only when each of these three levels of compatibility are met that a T1 multiplexer is truly D4 compatible; framing, channelization, and signaling. It is at this point that the user can take full advantage of new and existing tariff offerings.

Since they are TDM's, T1 multiplexers can be bit or byte-interleaved. Bit-interleaving, being the most efficient of the two, involves transmitting each bit as it is received from the incoming channels. The advantage is that, unlike statistical multiplexers and packet switching, only a small amount of buffer is needed--on the order of several hundred bytes per port or less. Since the multiplexer does not need to wait for an entire character to arrive before putting information on the T1 pipe, bandwidth is maximized and delay is minimized. However, bit-interleaved multiplexers cannot be used to interface with many of AT&T's

service offerings such as Customer Controllable Reconfiguration (CCR), also known as DACS (digital access and cross-connect system). The ideal T1 multiplexer is programable as either bit or byte-interleaved. As a practical matter, bit-interleaving can and should be used in most situations.

Today's T1 multiplexer has become so powerful that it is correctly referred to as a nodal processor, supporting multiple links, large channel capacities, and features such as automatic alternate routing and bandwidth contention. Virtually any application can be supported, including synchronous, asynchronous, voice, video, and high-speed data needs right up to 1.536 Mbps. Integrating these applications appears to be the key to using T1 effectively (figure 8).

Table 2 compares the relative merits of each of the three technologies discussed.

CONCLUSION

Because there are choices, it is tempting to pick the one direction that seems to offer the best overall fit. But, do network planners have to lock themselves into one way of thinking or one type of technology? Most corporate networks are really a collection of different applications and even separate networks based on common business interests. Thus, it may make sense to examine the integration of these technologies.

Integration can take place by simply sharing high-speed trunks to common locations. This allows smaller, independent networks to remain autonomous while reducing the overall line costs. But, as each individual network grows, reallocation of the backbone trunk bandwidth will be necessary, resulting in performance constraints. Also, sharing is fixed to the extent that traffic on one network cannot borrow bandwidth from the others during peak traffic periods or in times of link failures.

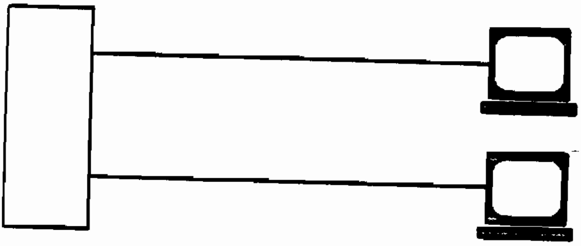
A better and more fully integrated approach is the hybrid design. This network also uses high-speed backbones, but combines private and public facilities as needed. This mix and match approach tailors the architecture to traffic characteristics.

The mechanism for interconnections are multifunction gateways. More than protocol converters, gateways are intelligent access points that integrate one network into another. As applications and requirements change, so does the personality of the network. Gateways enable the network planner to take advantage of existing service offerings from AT&T. Currently, these include CCR, M24, and Software Defined Network (SDN). Public and private packet networks can also be accessed.

Then there's ISDN. Touted as the ultimate solution for integrating all forms of information, ISDN is being carefully designed to retain compatibility with existing switching and transmission equipment, most notably T1. In fact, the number of rapidly growing T1 based private backbone networks are in effect private ISDN facilities. Thus, a network solution incorporating T1 will allow migration towards ISDN as it becomes available.

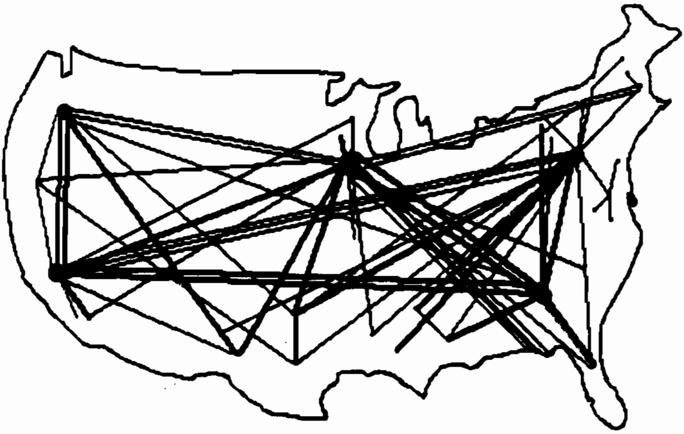
The best solution for network needs is based on hard questions and even harder decisions. Those willing to invest the time and energy in this process will develop a comprehensive communications strategy for today <u>and</u> the future.

POINT-TO-POINT LINES



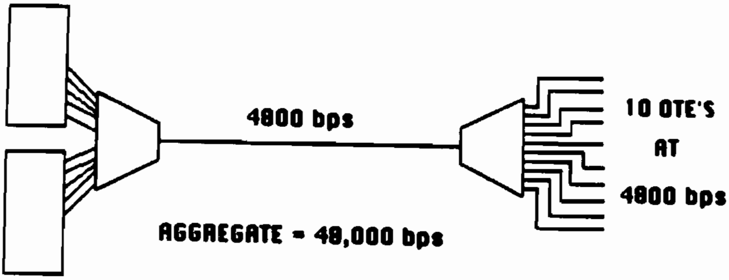
paradyne

CONSOLIDATED NETWORK ?

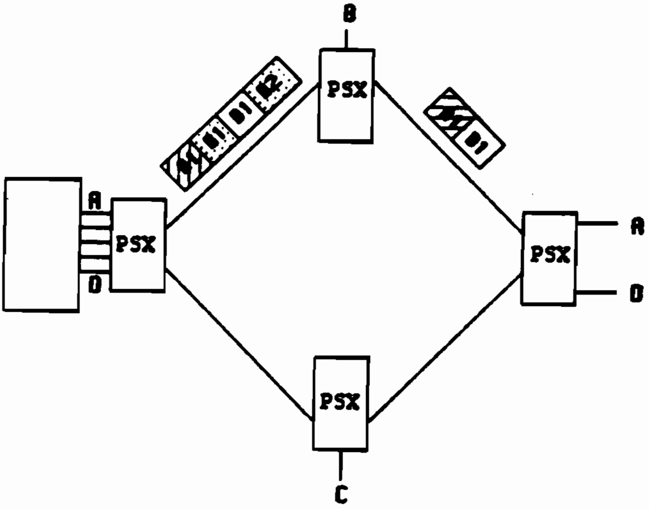


paradyne

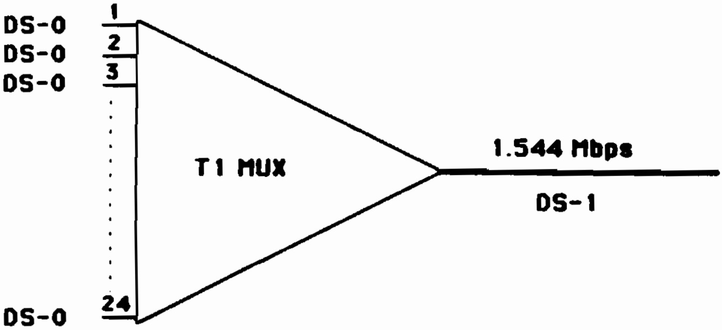
STATISTICAL MULTIPLEXING BACKBONE



PACKET SWITCHING NETWORK



T1 TECHNOLOGY



8 BITS X 24 CHANNELS = 192 BITS

192 BITS X 8,000 SAMPLES/SEC = 1.536 Mbps

8 BITS X 8,000 SAMPLES/SEC = 64 Kbps

paradyne

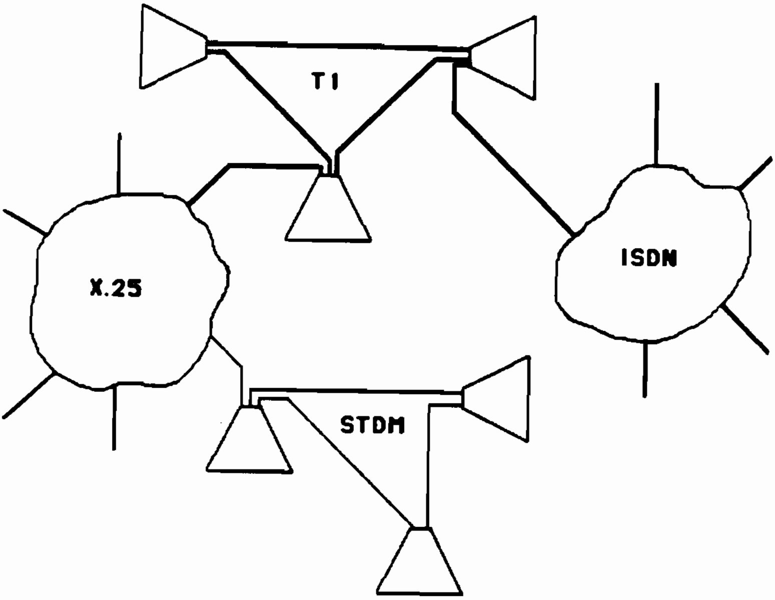
TABLE 1

STATISTICAL MULTIPLEXING
Vs.
PACKET SWITCHING

	STAT MUX	PACKET
ALTERNATE ROUTING	GOOD	EXCELLENT
ASYNCHRONOUS	EXCELLENT	GOOD
SYNCHRONOUS	FAIR	EXCELLENT
INTERACTIVE	EXCELLENT	EXCELLENT
BATCH	FAIR	FAIR
ERROR CONTROL	EXCELLENT	EXCELLENT
MANAGEMENT	GOOD	EXCELLENT
HIGH SPEED/VOLUME	FAIR	FAIR
COST	LOW	HIGH

paradyne

THE HYBRID NETWORK



The Spectrum Instruction Set, A 3000 Hacker's View

By Robert M. Green
(c) 1988, Robelle Consulting Ltd.
8648 Armstrong Road, R.R. #6
Langley, B.C. V3A 4P9 Canada
(604) 888-3666

I have been writing and debugging code for the Classic 3000 instruction set for over 15 years. Like most of you, I have been impatiently awaiting Hewlett-Packard's new line of computers, code-named "Spectrum". Now that actual Spectrum machines are spewing forth from Hewlett-Packard factories in ever-increasing numbers and going into production in my customers' computer rooms, it is time to start learning about these CPUs.

By the way, Hewlett-Packard prefers us to refer to their new computer family as the **HPPA** (*HP Precision Architecture*), not as the **Spectrum**.

I called HP's Direct Marketing group at (800) 538-8787 and used my VISA card to order a manual on the HPPA instruction set:

Precision Architecture and Instruction Reference Manual,
HP part number 09749-90014.

Like everyone else, I had read numerous articles from Hewlett-Packard about the objectives of the Precision Architecture, but I found clean, solid facts about the MPE XL machines hard to pin down. I needed some basic information to build on and the hardware instructions themselves seemed like a good base. Whatever software we eventually run on the HPPA machines will all be coded in the HPPA machine instructions.

Starting from the HP manual, I set out to compare the HPPA instructions with the Classic 3000 instructions and see what interesting differences I could uncover. Whether you ever personally write machine code for the HPPA or not, it can't hurt you to know something about these basic building blocks.

Goals of the Spectrum Project

A computer is a tool to execute programs built from sequences of simple "instructions". A typical instruction is something like "Add these two numbers together". The Classic HP 3000, designed in 1970, has a "complex instruction set", meaning that the instructions which programmers use are not the real hardware instructions. Each complex instruction is implemented by a hidden microprogram written in the real instructions.

The HPPA is a RISC machine, a "Reduced Instruction Set Computer", meaning that the microprogrammed instructions were removed. The programmers use the machine's real hardware instructions. Any task too complex for the RISC hardware is done by executing a series of the basic machine instructions, either as in-line code or by calling a subroutine.

The HP manual describes the observations that led to the idea of RISC computers:

> "Extensive research into patterns of computer usage reveals that general-purpose computers spend up to 80% of their time executing simple instructions such as load, store, and branch. The more complex instructions are used infrequently. On architectures with large, complex instruction sets, the simple, often-executed instructions incur a performance penalty caused by the overhead of additional instruction decoding, the use of microcode, and longer cycle time resulting from increased functionality..."
>
> "The RISC features implemented with the HP Precision Architecture include:
>
> Direct hardware implementation; no microcode.
> Fixed instruction size, one word in length.
> Small number of instruction types.
> Small number of addressing modes.
> Reduced memory access -- only load and store."

A primary goal of the HPPA is to complete the execution of one instruction in each machine cycle, and to keep that cycle time as short as possible.

General Structure of Spectrum Machines

According to the HP manual, the HPPA machines have 32 general registers available to the programmer, each with 32 bits. They are referred to as GR 0 to GR 31. Only GR 0, GR 1, and GR 31 have a hardware-defined special purpose, although other registers may be reserved by software convention. GR 0 is the bit bucket; when used as a source of data, it always provides zeroes. When used as a target, it throws away the result. GR 1 is used as the target in the Add Immediate Left instruction and GR 31 is used in the Branch and Link External instruction.

This general-register organization contrasts strongly with the stack organization of the older 3000. In the Classic 3000, the programmer has access to a push-down stack and to 16-bit registers whose hardware function is highly specialized (i.e., Q points to local variables, DB to global variables, etc.). The closest thing to a general register is the Index Register, and it is used for special functions in many instructions.

All HPPA instructions are 32 bits long, with the first 6 bits reserved for the Major Opcode. This allows 64 major opcodes, allocated as follows:

23 opcodes are currently illegal (allowing HP lots of room for expansion),
27 opcodes are single instructions (e.g., 1 is Load Byte, LDB),
12 opcodes are instruction groups (e.g., 0 is the System Operation group),
1 opcode for up to 4 tightly-coupled Special Function Assist processors,
1 opcode for Co-processors, including 15 floating-point instructions.

The 27 major opcodes that invoke a single instruction provide load, store, branch, and other functions (e.g., LDB, STB, LDW, STW, etc.).

The 12 instruction groups expand into 108 unique instructions, as shown below. I show the opcodes for the 12 groups as Hexadecimal (base 16) values with a $ prefix. You should brush up on your Hex arithmetic, because the Spectrums are definitely Hex machines.

Opcode	Instruction Count / Type	
$00	11	system control instructions
$01	19	memory management instructions
$02	31	arithmetic/logical instructions
$03	15	indexed and memory instructions
$3A	4	unconditional branch instructions
$25	6	immediate arithmetic instructions
$2C		"
$2D		"
$34	4	extract/deposit instructions
$35		"
$09	8	co-processor load and store instructions
$0B		"

Floating Point

The floating-point co-processor follows the IEEE standard. It provides 15 functions, including square root, on three sizes of number: 32-bit, 64-bit, and 128-bit. Please note, however, that MPE XL versions of HPPA also support the Classic 3000 format for floating-point, via software emulation. The two formats are not compatible. For maximum performance I assume that you must convert your data files to the IEEE standard and also convert all of your application programs and third-party or contributed tools.

Another source of HPPA information, the book **Beyond RISC!** published by SRN, describes the floating-point problem this way:

> "The HP 3000 uses 9 bits for the exponent and 22 bits for the mantissa... The IEEE format defines a single precision number to contain 8 bits for the exponent and 23 bits for the mantissa. While the difference is small, it affects the size of the numbers that can be stored... The floating point format can be a problem in migration if the format is used extensively in disc files and databases."

The Fortran/XL and Pascal/XL compilers have options to force use of the old Classic 3000 floating point, but remember: this uses software instead of hardware for arithmetic. There is no good way that I am aware of to tell which format is being used for the floating point numbers in a particular database or file.

Total Instruction Count

If my arithmetic is correct, that makes a total of 155 instructions for the HPPA. Many of these are minor variations or are of interest only to low-level systems programmers.

How Does the Spectrum Multiply?

When comparing the Classic 3000 architecture with the HPPA, two of the most obvious deletions are the Integer Multiply and Integer Divide instructions. Since these instructions are used less than 1% of the time, neither the HPPA nor the Classic 3000 has the expensive hardware needed to do fast Integer Multiply and Divide. On the Classic 3000, the designers use a complex microprogram that is hidden from the customer and may vary from model to model. The HPPA designers wanted to multiply with reasonable performance without the microprogram or special hardware.

There is an interesting article about this problem in the Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS II), October 1987, published by The Computer Society of the IEEE. Whew! The paper is:

Integer Multiplication and Division on the HP Precision Architecture
Daniel J. Magenheimer, Liz Peters, Karl Pettis, Dan Zuras.

In their search for a fast method to multiply and divide, the HPPA team started out with general algorithms for subroutines to multiply and divide two integers. The usual algorithm for integer multiply, called "Booth encoding", involves looping through the multiplier two bits at a time replacing strings of zeroes or ones with a constant.

The HPPA stores numbers as 32-bit words, each bit or binary digit being capable of storing a one or a zero. If two bits are processed per loop, the Booth algorithm requires 16 such loops. To support this clever and tricky algorithm, machine designers often include a Multiply Step (and Divide Step) instruction to perform the two-bit processing. However, the HPPA designers found that the special Multiply Step and Divide Step instructions would have made their CPU more complex and increased the basic cycle time. This was not acceptable because it conflicted with the goals of a RISC architecture.

The HPPA designers attacked this problem in the same way that we often attack performance problems with database systems. They did a frequency analysis of actual multiplications in real user programs (and divisions -- the strategies in the IEEE article apply to divide as well as to multiply). They found that 91% of multiply tasks include a constant operand known at compile time, that the constant operands tend to be small, that non-constant operands also tend to be small, and that standard results occur more frequently than extended results (i.e., 32-bit answer versus 64-bit).

> "In the spirit of the philosophy espoused by RISC architects we should atttempt to optimize the most frequent cases, even at the cost of decreasing performance of the less frequent cases... By recognizing the inherent non-uniformity and special cases of operands (and results), we may be able to increase the overall performance."

Rather than implement a costly Multiply Step instruction that would slow down the basic cycle time for <u>all</u> instructions, they decided to take advantage of some hardware that the HPPA needed for another purpose. Remember, the HPPA is a byte machine, meaning that the byte (8 bits) is the basic unit that it can manipulate. All of memory is referenced via consecutive byte addresses, such as byte 1, byte 23, byte 4,567,890. However, most numbers are stored as words (4 bytes, 32 bits), half-words (2 bytes, 16 bits), or double words (8 bytes, 64 bits).

If you have a table of words starting at byte address 4000 and you want the 30th word in the table, you must compute the byte address of that word. The calculation is simple: 4 times 30 + 4000 = 4120. The position of the 30th word after byte 4000 is "4 times 30" because there are 4 bytes per word.

The HPPA has basic instructions to access elements in tables of words, half-words, and double-words. To ensure that these instructions execute in one CPU cycle, the HPPA includes hardware called a "pre-shifter"; it can multiply any register by 2, 4, or 8 prior to adding it with another register. It is called a shifter because it shifts bits to the left; in binary arithmetic, shifting one bit to the left is the same as multiplying by 2, shifting two bits is multiplying by 4, and shifting 3 bits is the same as multiplying by 8. (Computers like powers of 2, such as 4, 8, 16, 32...)

Since this "pre-shifter" was already needed, the HPPA designers could inexpensively include nine instructions to make the shifter accessible to the programmer. Each instruction shifts any of the 32 registers, adds it to any register, and stores the result in any register. The shift count can be one, two, or three bits and the add can be INTEGER, LOGICAL, or INTEGER with TRAP on OVERFLOW.

SH1ADD	Shift One and Add
SH1ADDL	Shift One and Add Logical
SH1ADDO	Shift One, Add and Trap on Overflow
SH2ADD	Shift Two and Add
SH2ADDL	Shift Two and Add Logical
SH2ADDO	Shift Two, Add and Trap on Overflow
SH3ADD	Shift Three and Add
SH3ADDL	Shift Three and Add Logical
SH3ADDO	Shift Three, Add and Trap on Overflow

Overflow - An Aside.

> When the result of a multiplication won't fit in the 32-bit word that the HPPA uses to hold numbers, an "overflow" occurs. Some programming languages insist that overflow errors be detected and handled in a special way. Other languages, such as C, have no special requirements. Most computation instructions on the HPPA have a form that traps for overflow and another form that does not trap.

Now they have very fast instructions that allow both a shift (i.e., a multiply by two, four or eight) and an add. This is a powerful building block for fast multiplication.

Multiplying By Constants

Let's look at multiplying by a constant first, since that happens so often in practice. Multiplying a number by 10 is the same as adding the number together 10 times, or multiplying by 5 twice. The key to implementing constant multiplication is that any multiply operation can be converted to a series of additions and smaller multiplies:

```
120 = 10 times 12
120 = (5 times 12) + (5 times 12)
120 = (4 times 12 + 12) + (4 times 12 + 12)
```

The only tricky step above is the last one. We converted to this format because computers like to multiply by powers of 2 (e.g., 2, 4, 8). Remember, our Shift-and-Add instructions give us the ability to multiply any register by 2, 4 or 8 in a single instruction. The compiler can convert any multiplication by a constant into a series of specific Add and Shift-and-Add instructions. Generally, the HPPA can do such a multiply in four or fewer instructions, often in only one or two.

To multiply register R by 10, we would use the following two instructions:

```
SH2ADD R,R,R ; shift R two (multiply by 4),  add to original R, store in R.
ADD R,R,R ; add R to R (double R) and store back in R.
```

The first instruction leaves us with (5 times R) in register R and the second instruction doubles R, leaving us the (10 times R) in register R.

In our example, R would have the value 12:

```
SH2ADD   12 * 4 = 48 (shift two), 48 + 12 = 60 (add to original).
ADD      60 + 60 = 120
```

Multiplication by small constants occurs often in programs because of the need to index into tables. In order to compute the byte address of the table element, the program must multiply the element number by the element size in bytes (usually small!).

Multiplying By Variables

Examine the following statement from a Pascal program:

```
ExtendedPrice := Quantity * UnitPrice;
```

It multiplies "quantity" by "price" to give the extended price. Neither the quantity nor the price is known when the program is written, since this statement will be executed for many different parts on many different orders. In this case, the HPPA needs a multiply method that can handle any possible values for the operands.

When neither of the operands is known at compile-time, we have "multiply-by-variable". You can construct a very simple, but slow, method of multiplying by an arbitrary variable using just three basic computer functions: addition, divide by two (arithmetic right shift), and bit test. This "naive" algorithm loops 32 times, processing one of the 32 bits each time. A simple optimization is to exit the loop as soon as the shifted multiplier is zero. Since most operands have many leading zero bits, this improves the "average" multiply by at least 40 percent. For example, the number 40 is represented in the computer as 26 leading zero digits, followed by 101000.

Here is the algorithm to multiply A times B giving Result, represented in a high-level language:

```
Result := 0
while A not zero do
  if A odd then Result:=Result+B  {add if last bit = 1}
  B := B + B    {double B; multiply by 2; shift left 1 bit}
  A := A / 2    {shift right 1 bit, get next bit to test}
```

Each time through the loop, we add the current value of B to the Result if A has a one in the last bit (i.e., is odd), then we double the value of B and shift A right one bit to bring up the next bit to be examined. Although this may seem obscure, it is actually the same process you apply when you do decimal multiplication by hand. The only difference is that we are using Base 2 instead of Base 10.

The Spectrum's Shift-and-Add instructions allow us to examine several bits at a time from both the multiplier and multiplicand, reducing the worst case and the average case. But this is just the start of optimizing based on frequency analysis:

> "It is rarely the case that both operands are large, say larger than 16 bits, because the result will be an overflow... If the multiplication does not result in an overflow, at least one of the operands must be representable in less than 16 bits. With a simple test and possible swap, we have reduced the maximum number of times through the loop to four and the average to two."

The HPPA team then noticed that each time through the loop multiplies the multiplicand by a number between 0 and 15. Because the HPPA can do any small constant multiply in three instructions or less as described above, they introduced a CASE statement to include in the loop the explicit code for each possible constant multiply. They also observed that one of the operands is less than 16 over half the time, so they optimized for single loop multiply. Finally, they added a quick exit for values of zero and one, and special checks for positive operands (which occur 90% of the time).

Here is how they describe their optimization results:

> "Multiplication by compile-time constants can generally be performed in four or fewer (single-cycle) instructions; multiplications by variables, including full overflow checking, can be obtained in an average of 20 (single-cycle) instructions... on the Precision Architecture the average multiply requires about six cycles."

The HPPA designers chose to concentrate on speeding up the most-often requested multiplication tasks at the expense of the least-often used tasks. They used a shift-and-add algorithm to accomplish this, because the hardware for this algorithm was "free". When the desired multiply operation includes large values, loops similar to those on a Classic 3000 are executed. When the operands include a constant or a small variable value, the multiply is executed in a very few of the HPPA's fast instructions. The result is better overall performance, with the proviso that the operations will be harder to spot during program debugging. The corollary is that frequency analysis is the first step in optimization.

What does this mean to you? If the compiler writers use these ideas fully, the code generated should give excellent performance. However, when you are debugging programs with the System Debugger, don't expect to see a simple MPY instruction, nor even an obvious subroutine call. You may see one of many methods of doing an integer multiply, depending on the operand values, on the compiler's intelligence, and perhaps on the level of optimization you requested!

Spectrum Instruction Format

The basic HPPA instruction format is 6 bits for the major opcode and 10 bits to specify the source registers (if any).

```
------------------------------------------------------------
|  Opcode   |  Source2   |  Source1   |  Rest of Instruction|
------------------------------------------------------------
   6 bits      5 bits       5 bits            16 bits
```

Source2 and Source1 select input registers for the instruction. The position of the target register (if any) can vary, but is often at the end of the instruction.

The only memory reference instructions are Load and Store; there are no instructions to add memory, as in the Classic 3000. Let's look at a typical memory reference instruction, Store Word (STW):

```
                         STW - Store Word

------------------------------------------------------------
|  $1A       | Base-Reg | Data-Reg | S |    Offset          |
------------------------------------------------------------
   6 bits      5 bits     5 bits   2 bits  14 bits
```

The function of STW is to store 32 bits from a specified General Register (i.e., Data-Reg) into memory. The effective byte address is computed by adding the 14-bit Offset value in the instruction to the 32-bit Base address in another General Register (i.e., Base-Reg). The computed address for a Word load or store must be at a multiple of 4 bytes, just as the address for a Half-word load or store must be at a multiple of 2 bytes.

Storage Boundaries on the Spectrum

The HPPA is a byte-address machine, but as the manual says...

> "All addressable units must be stored on their naturally aligned boundaries. A byte may appear at any address, halfwords must begin at even addresses, and words begin at addresses that are multiples of 4."

The Classic 3000 usually stores 16-bit values at even-byte addresses, like the HPPA, but does not force 32-bit values to be on 32-bit boundaries. In this matter, the two machines are incompatible. If you restore a Classic-3000 file or database onto the HPPA, the machine will not be able to use the Load and Store Word instructions to reference mis-aligned fields. Instead, it will have to use pairs of Load Halfword and Store Halfword instructions. You will use fewer instructions if you convert your files to be HPPA-aligned and re-compile your programs, but we can't say whether this improvement will even be measurable until we can run some performance tests.

If you have an MPE V file (i.e., aligned on 16-bit words) and you access it via a native-mode MPE XL program, you must be certain that your compiler is aligning fields in data buffers on 16-bit boundaries. Otherwise, your program will read and update the wrong fields. The Native-Mode compilers have options to cope with alignment issues and the details vary from compiler to compiler. This issue is well explained in the **Beyond RISC!** book from SRN.

Delayed Branches

We have seen that one of the primary goals of the HPPA is to complete the execution of a useful instruction in each machine cycle: 80 ns on the 950. Note: the "official" machine cycle of the 950 is subdivided into four sub-cycles, in order to implement the different phases of an instruction. This allows HPPA to work on several instructions at once. Thus it actually takes a minimum of three machine cycles to execute an instruction, but the effective throughput is one instruction completed every cycle. We will tend to ignore these details, except where necessary.

The branch operation is a serious stumbling block to these HPPA goals. Branch instructions are difficult to implement in one cycle because they must first compute the branch location, then actually retrieve the new instruction at that location.

The HPPA is a pipelined computer, meaning that it has a pipeline of instructions that it is preparing to execute while it is actually executing the current instruction. As long as the instructions execute in sequence, the hardware to pre-retrieve the following instruction is fairly straightforward. When we branch, we cannot have the instruction at the branch destination in our pipeline, because we don't know in advance which instruction it is.

The choice seems to be between using two cycles to execute the branch or stretching the length of the basic cycle time to allow for retrieving the branch location from memory. Neither choice seems attractive.

What the HPPA does is ingenious and profoundly disturbing: the HPPA delays the execution of the branch for one cycle. This strategy is frequently used in microcode (see the HP 3000 6x and 7x machines, for example), but this is the first time I have seen it in an official instruction set.

Here is how the Precision Architecture manual describes the concept of Delayed Branching:

> "All branch instructions exhibit the delayed branch feature. This implies that the major effect of the branch instruction, the actual transfer of control, occurs one instruction after the execution of the branch. As a result, the instruction following the branch (located in the 'delay slot' of the branch instruction) is executed before control passes to the branch destination."

When you look at assembler listings of MPE XL native-mode programs, you will sometimes see instruction sequences like this:

```
BL  opencarton        ;branch and link
NOP                   ;delay slot
```

The compilers and/or optimizers could not find anything useful to do in the cycle after the branch ("the delay slot"), so they inserted a NOP (no operation). Effectively, this sequence is a two-instruction branch.

Or, you may see an instruction sequence like this:

```
 BL closecarton        ; branch
```

```
        LDW 26 ...              ; load word in the delay slot
```

The BL branch instruction is executed before the LDW instruction, **but does not take effect until one cycle later.** The LDW instruction that comes <u>after</u> the BL instruction is actually executed while the BL completes. The LDW in this case is probably loading one of the parameters needed by the 'closecarton' subroutine.

Let me see if I can make this a little more clear with an analogy. If you are taking an airplane trip to Minneapolis, you follow a program with instructions like these:

```
1.      book flight
2.      reserve hotel
3.      pay for ticket
4.      reserve rental car
5.      pack bags
6.      fly to Minneapolis
7.              (wasted time, read a book perhaps)
8.                          collect baggage
9.                          get rental car
10.                         check into hotel
```

Everything takes about the same time ("one machine cycle"), except for the actual flight. This takes more than one machine cycle, leaving a wasted time period when you catch up on reading or napping.

Now, imagine that you are a HPPA computer, determined not to waste a single machine cycle. How would your trip be programmed?

```
1.      book flight
2.      reserve hotel
3.      pay for ticket
4.      reserve rental car
5.      fly to Minneapolis
6.              (pack baggage during the delay slot)
7.                          collect baggage
8.                          get rental car
9.                          check into hotel
```

Using the HPPA airplane, our trip took only 9 cycles instead of 10. How did we shorten the time? By packing our bags while we are flying to Minneapolis. If this sounds like "being in two places at the same time" to you, you're right! The HPPA can work on several tasks at the same time due to the power of pipelining. The HPPA compilers try to take the last step in your program before the branch and move it to after the branch. In this way, it can be executed during the time that the branch is delayed.

Of course, there are a few catches.

If the branch is a conditional one, such as "branch if register two equals zero", then we couldn't move any instruction that might change the value of register two. We would have to find an instruction that could not have any impact on whether the branch would be taken.

Another wild and unbelievable implication of the delayed branch is spelled out very dryly in the instruction set manual, as follows:

"Consider the situation in Figure 4-2.

Figure 4-2. Branch instruction in the delay slot

```
Loc.  Instruction                                      Reference#
100  STW
104  BV r0(r7)    branch vectored to location 200            I1
108  BL r4 r0     IA relative branch to location 400         I2
10C  ADD r2,r6,r9 next instruction in linear code sequence

   ....

200  LDW  0(r3),r11   target of BV instruction I1            I3
204  ADD              next instruction, never executed!

   ....

400  LDW              target of BL instruction I2            I4
404  STW                                                     I5
```

"A taken branch instruction, I2, is executed in the delay slot of a preceding taken branch, I1. When this occurs, the first branch I1 schedules its target instruction, I3, to execute after I2, and the second branch, I2, schedules its target instruction, I4, to execute after I3. The net effect is the out-of-line execution of I3, followed by the execution of I4. Also, if I3 were to be a taken branch, its target, I5, would execute after I4, and I4 would also have been executed out of its spatial context."

How can we translate this strange computer situtation into a real life analogy? It would be as if, on our flight to Minneapolis, we were high-jacked to New York, but the high-jackers were unable to keep us from stopping in Minneapolis just long enough to lose our bags (i.e., one cycle).

When I first saw this example in the manual, I was at a loss to think of any practical use for it. But it is never wise to assume that anything in the HPPA is by accident. On further reflection, I thought I could see a way to use two branch instructions in a row to good advantage. Classic 3000 hackers are aware of the XEQ instruction, which allows you to execute another instruction, one that is not known until execution time. The HPPA does not have an XEQ instruction. But, by combining the above example with the fact that the HPPA does not distinguish between code and data as strongly as the Classic 3000 does, you might be able to produce a reasonable facsimile of the XEQ instruction.

The HPPA delayed branch is not just an intellectual curiosity that you can ignore. It has may practical implicatons. When you are using the system debugger to set a breakpoint near a branch instruction, it is very difficult to remember that the instruction after the branch will be executed before the target of the branch. Deciding where to set the breakpoint can sometimes seriously strain your imagination.

Memory Addressing

One of the goals of the HPPA project was to allow programs to address enormous amounts of memory, many times more than the current hardware technology can deliver. The memory Load and Store functions compute an effective byte address by adding an Offset value to a 32-bit Base address held in one of the 32 registers.

The effective address, however, is not a real memory address, nor is it an address in the stack Data Segment (as in the Classic 3000). It is an address within a Space. The HPPA can have thousands of spaces active and the program can access eight of them at any instant using eight hardware Space Registers. Within the memory reference instructions, the S field tells how to select the Space Register. If S is 1, 2, or 3, the instruction uses the space defined by Space Register 1, 2, or 3. If S is 0, it uses the space defined by 4 plus the upper two bits of the address in the Base-Reg value (i.e., 4, 5, 6, or 7).

Spaces are similar to Segments in the Classic 3000 architecture, but can be <u>much</u> larger. Only SR 0 has a hardware-defined purpose; it is used for the return address of interspace calls. Software conventions define SR 4 as the code space, SR 5 as the stack space, SR 6 as a space for shared data, and SR 7 for public operating system code, literals, and data. SR 1, SR 2, and SR 3 are available to the programmer as temporaries for the construction of 64-bit long pointers (i.e., pointers into any space).

Quadrants

When the Space field in the instruction is 0, the target Space Register is selected by adding 4 to the upper two bits of the 32-bit address. Two bits allows four possible values: 0 selects SR 4, 1 selects SR 5, 2 selects SR 6 and 3 selects SR 7. However, to keep the hardware simple, the entire 32-bit word is the byte address within that space, not just the 30 bits after the space selector. This is called short pointer addressing since only 32 bits are used to select both the space and the offset within the space. With long pointers, a full 32-bit word is used for the space number and another 32 bits for the offset within the space.

Because the upper two bits that select the Space Register are included as part of the address, you can only address a quarter of the potential addresses in a space using a short pointer. HP describes the restriction as follows:

> "Only one fourth of the space is directly addressable by the base register with short pointers and the region corresponds to the quadrant selected by the upper two bits. For example, if a base register contains the hex value $40020000, space register 5 is used as the space identifier and the second quadrant of the space is directly addressable."

In 32 bits, we can address the first quadrant of the space pointed to by SR4, the second quadrant of SR5, the third quadrant of SR6 and the fourth quadrant of SR7. This makes for some very large memory addresses, and can be quite surprising to an old-time 3000 programmer. To get at the other 3 quadrants of those spaces we must use a long pointer where the desired space register is not encoded in the 32-bit offset value.

Bounds checking of subroutine parameter addresses is a more complex issue on the HPPA than on the Classic 3000. On the Classic, you check a parameter address to see if it is between the DL and S registers. On the HPPA, a legitimate parameter address may not even point to the user's data space, and read/write security within a space can theoretically vary from page to page (a page is 4096 bytes). Anyone who understands this issue is invited to correspond with the author.

Pipelining and Register Interlock

In order for the HPPA to complete the execution of an instruction in each machine cycle and to have that cycle be as fast as possible, it has pipelining. It has a pipeline of instructions that it is preparing to execute while, at the same time, it is actually executing the current instruction. A major obstacle to completing an instruction in every cycle is the Memory Load operation.

What happens with memory loads on the HPPA is similar to what can happen with your checking account. You deposit a check to your account on Monday, but if you try to withdraw that amount on Tuesday, you may be blocked by the bank. They claim it takes two cycles for the check to clear.

The same thing happens with memory load instructions. It takes one cycle just to compute where in memory you want to load data from. Then it takes another cycle to retrieve that data and put it in the desired register.

The machine designers faced a choice: they could either make the basic cycle time longer to allow for both the computation of the effective address <u>and</u> the load from main memory. Or, they could go on to the next instruction before they had finished with the load instruction.

This second choice is what the HPPA does. When you load from memory, the pipeline has already prepared the next instruction for execution. The HPPA goes ahead and starts executing that instruction, even though it has not completed the preceding load operation.

Question: What happens if the next instruction requires the data that is being loaded?

Answer: Register Interlock.

You should not refer to the target of a Load instruction in the instruction that follows a Load instruction. If you do, you get a **Register Interlock**. This pauses the program for one cycle, allowing the load from memory into the register to complete.

The compilers and optimizers on the HPPA attempt to re-order the machine instructions to avoid Register Interlock. For example, a sequence that does Load-Store, Load-Store and causes two Register Interlocks, can be converted to Load-Load, Store-Store with no interlock delays.

```
A := B;     LDW 31 load B      LDW 31 load B
C := D;     STW 31 store A     LDW 30 load D
            LDW 30 load D      STW 31 store A
            STW 30 store C     STW 30 store C
             six cycles         four cycles
```

The first code sequence with alternating LDW and STW instructions takes two cycles longer than the second code sequence, which does both LDWs first, then both STWs.

The ability of the compilers to reorder instruction sequences to avoid register interlocks is important to attaining the theoretical speed of the HPPA.

The 930 Register Interlock is not as smart, nor as expensive, as the 950. Rather than check the type of the next instruction to see if it could contain a reference to a register, the 930 just looks for the 5-bit pattern of the register number in the position in the instruction where register numbers usually appear. Of course, if you are executing an instruction like LDIL (load immediate left), with its 21-bit constant operand, the 5-bit pattern of the register may occur in the constant operand by chance. Unfortunately, you get a register interlock anyway. The 950 avoids this trap with extra AND and OR logic.

The only reference to Register Interlock in the HP manual is on page 5-16:

> "Execution is faster if software avoids dependence on register interlocks. Instruction scheduling to avoid the need for interlocking is recommended. This does not restrict the delay a load instruction may incur in a particular system to a single execution cycle; in fact, the delay will be much longer for a cache miss, a TLB miss, or a page fault."

Register Arithmetic Vs. Stack Arithmetic

The Classic 3000 has 19 **arithmetic** instructions that operate on the top of stack values. The functions provided are Add, Subtract, Compare, Zero, Multiply, Divide, And, Not, Or and Xor.

The Classic 3000 provides Add, Subtract and Compare for 16-bit and 32-bit integers and for 16-bit unsigned integers. Zero pushes a 16-bit or 32-bit zero value onto the stack. And, Not, Or and Xor (exclusive OR) are provided for 16-bit values only. In all cases, the operands are taken off the top of the stack and the result, if any, is pushed onto the stack. All instructions compute and set Overflow, Carry and Condition Code fields in the status register, whether you need them or not.

The HPPA has similar arithmetic functions, but the operands are always 32-bit values. If you want to Add two 16-bit values together, you actually do a 32-bit Add and then manually check the result for overflow of 16 bits using another instruction. The arithmetic functions take two of the 32 general-purpose registers as input and one of the registers as output. The Multiply and Divide functions are not included in the HPPA instructions set and were discussed in detail earlier in this article.

Addition

There are numerous variations on each of the basic arithmetic functions. For example, here are the ways you can Add two registers and put the result in a third:

ADD	Add (32-bit signed addition)
ADDO	Add and Trap On Overflow
ADDC	Add with Carry
ADDCO	Add with Carry and Trap on Overflow
ADDL	Add Logical (32-bit unsigned addition)

The regular ADD instruction does not trap on integer overflow; there are variations on the instruction for trapping. Languages differ in their treatment of overflow conditions. Pascal has very precise requirements for overflow traps, while the C language explicitly says that it does not trap on overflow - the result is undefined. "With Carry" means that the carry bit from a previous add function is included in computing the answer. ADDL is an unsigned 32-bit addition, where all numbers are treated as positive values.

Nullification

On the Classic 3000, arithmetic functions adjust bits in the status register which you can then test in a branch instruction. For example, you might subtract one number from another and branch if the result is zero. On the HPPA, arithmetic instructions allow you to "nullify" the next instruction if a certain condition, such as zero result, occurs or does not occur. Rather than have numerous conditional branch instructions, the HPPA puts the conditional logic in the arithmetic instructions. This is similar to many other machines that have "skip" conditions in instructions (if something happens, skip the next instruction). Experienced programmers who have worked with the HPPA at the machine-language level report that having Nullify in so many instructions is a big aid to writing tight code.

Subtract

There are even more variations on Subtract than there are on Add:

SUB	Subtract (32-bit signed subtraction)
SUBO	Subtract and Trap On Overflow
SUBB	Subtract with Borrow
SUBBO	Subtract with Borrow and Trap On Overflow
SUBT	Subtract and Trap on Condition
SUBTO	Subtract and Trap on Condition or Overflow

Subtract "with Borrow" is similar to Add "with Carry": it means that the borrow bit from a previous subtract operation is to be included in computing the answer. As with Add, there are versions of Subtract to trap or not on overflow. The "trap on condition" instructions give you the ability to trap to an error routine if a certain condition results after doing the subtraction. Having a trap instead of a skip on the condition means that you can test the condition in one instruction instead of two, but the usefulness of this is limited to tests that will likely abort the program when the condition occurs. These are probably included for doing efficient bounds checking of array indices.

Compare Function

COMCLR is the Compare and Clear function. The two input registers are compared and the next instruction can be conditionally skipped, based on the result. In addition, you have the ability to clear another register in the same cycle. Some day you may appreciate being able to compare and clear in the a single instruction. For example, using the Nullify field in the COMCLR instruction with a following Add Immediate Instruction, you could compare two registers and leave a zero or one in another register that reflects the result of the compare.

Logical Functions: And, Not, Or, Xor

The HPPA has basically the same logical functions as the Classic 3000, except that the operands are any of the 32-bit registers rather than the 16-bit values currently at the top of the stack. There is one unusual instruction:

```
ANDCM   And with Complement
```

ANDCM complements the value of one register, then ANDs it with another register, leaving the result in a third register. The ANDCM instruction can produce the same result as the Classic 3000's NOT instruction; **ANDCM** `r,r,r` flips the bits in register r, ANDs them with the register r and stores the result back in register r, effectively a NOT of register r.

Immediate Functions

The HPPA has 19 machine instructions with constant or "immediate" operands:

Opcode	Operand-Size	Function
LDO	14 bits	Load Offset
LDIL	21 bits	Load Immediate Left
ADDIL	21 bits	Add Immediate Left
ADDI	11 bits	Add to Immediate
ADDIT	11 bits	Add to Immediate and Trap On Condition
ADDIO	11 bits	Add to Immediate and Trap on Overflow
ADDITO	11 bits	Add to Immediate and Trap on Cond/Ovfl
SUBI	11 bits	Subtract from Immediate
SUBIO	11 bits	Subtract from Immediate and Trap on Overflow
COMICLR	11 bits	Compare Immediate and Clear
COMIBT	5 bits	Compare Immediate and Branch if True
COMIBF	5 bits	Compare Immediate and Branch if False
MOVIB	5 bits	Move Immediate and Branch
ADDIBT	5 bits	Add Immediate and Branch if True
ADDIBF	5 bits	Add Immediate and Branch if False
DEPI	5 bits	Deposit Immediate
VDEPI	5 bits	Variable Deposit Immediate
ZDEPI	5 bits	Zero and Deposit Immediate
ZVDEPI	5 bits	Zero and Variable Deposit Immediate

The Classic 3000 has only 10 immediate instructions and they all have an 8-bit constant operand. The Classic 3000 has MPYI and DIVI, but lacks the Deposit Immediate instructions. The HPPA has more immediate instructions, but they are all "basic" one-cycle instructions (i.e., no multiply immediate instruction).

The size of the immediate operand on the HPPA varies: 5 bits (-15 to +15) for branches and bit deposits, 11 bits (-1K to +1K) for most immediate functions, 14 bits (+8K to -8K) for Load Offset which can also add, plus Load and Add Immediate Left (they take a 21-bit operand that is shifted left to have 11 zeroes after it). Using LDIL and LDO together, you can load any 32-bit constant into any register in two machine cycles. The LDIL instruction has another interesting use: to embed statement numbers in the object program. When the compilers want to leave road markers in the code, they use LDIL to register 0, which is a null operation. The constant parameter refers back to the source code statement number or line number!

The Classic 3000 has one Add Immediate Instruction (ADDI). The HPPA has at least 6. The HPPA can add an 11-bit signed constant to a register and store the result in another register, with four varations: no trap (ADDI), trap on overflow (ADDIO), trap on one of 16 conditions such as less than zero (ADDIT), or trap on overflow or a condition (ADDITO). ADDI and ADDIO can also specify one of 16 conditions for nullifying the next instruction (i.e., never, <, =, odd, etc.). Two other instructions, ADDIBT and ADDIBF, allow you to add a 5-bit signed constant to a register, test a condition, and branch if the condition is true or false. The conditions that can be tested in the Add Immediate instructions (and in most HPPA arithmetic instructions) replace eleven separate branch instructions on the Classic 3000 (e.g., BRO, BRE, BCC, and so on).

Conclusions

The design of the Classic 3000 instruction set was heavily influenced by programmers. Their desire was to avoid the programming problems they had encountered writing systems software in Assembler on HP's earlier 2100 computer line. As such, the instruction set represented their practical feel for what would be nice to program in.

The HPPA instruction set is striking in the degree to which it is based on measurements, not feelings. Everywhere you look, it shows signs of hard engineering tradeoffs. If measurements showed that few real programs used a particular function, that function didn't have much chance of making it into HPPA. Conversely, if you find a particular function in the instruction set, it probably has a valuable use in some large class of programs (no matter how obscure it looks to you).

The Classic 3000 code is more compact and easier for people to read and write, but the HPPA code is more powerful. Not only can the HPPA instructions access vastly more memory, but they are at least a hundred times more flexible than their Classic 3000 equivalent. The HPPA instructions resemble the internal microcode of the Classic 3000 much more than they resemble the official "machine" instructions of the Classic. Because of the many options available, the HPPA depends on adroit compilers to select the best instruction for each situation.

When Hewlett-Packard announced their intention to build a RISC machine, the theoretical papers on RISC led me to expect a machine with 30 to 50 instructions. When I first looked at the HPPA, I was surprised to find over 150 instructions. As I studied HPPA, however, I started to get a feel for it. The instructions have incredible complexity and power, but they also have "reduced complexity" from a computer engineer's point of view. They are easy to implement in fast, elegant hardware. Although this strategy has shifted many complex problems from the hardware to the software, the HPPA designers appear to have provided the programmers with the power they need to solve their problems.

PERFORMANCE MONITORING
AND
CAPACITY PLANNING ON MPE XL

Robert S. Apgood
Strategic Systems, Inc.
10502 11th Ave. NE
Seattle, Washington 98125
(206) 525-3309

I. INTRODUCTION

A vast number of excellent papers and seminars have been presented in recent years describing the steps to and complexities of Performance Monitoring and Capacity Planning in the classic HP 3000 environment. While most users acknowledge that the ideas presented in those papers and seminars would certainly be useful, not too many shops have actually taken steps to implement an on-going function of monitoring and analysis to evaluate current utilization levels and project trends of resource demands.

The intent of this paper is not to re-hash how it's done, what the numbers should look like, or what spiffy things our products do to make it easy for you to accomplish this task (even though that is true). Rather, we are going to look a common scenario and discuss why this needs to be done in the classic HP 3000 shop and, even more importantly (is that really a word?) why it must be done in the MPE XL environment.

This paper can be viewed as an "introduction" to the concepts and concerns of performing monitoring and analysis. It is the first in a series of papers to be presented by Strategic Systems, Inc. on the subject and will be followed in later conferences with progressively more technical and complex presentations.

II. TWENTY-TWENTY HINDSIGHT

Historically, in the MPE/V environment, most shops haven't really experienced the need for trend tracking. Why? The basic answer to this question is that they haven't really needed to. For the majority of its life, the HP 3000 hardware has been able to exceed the capabilities of the MPE operating system. Then, as the processing demands of most shops increased, the capabilities and performance of MPE kept pace with those demands and the user was lulled into a false sense of security that "trusty Old Clem" was actually able to handle the throughput demands all along. This feeling was further compounded by the fact that most shops experienced a need for greater connectivity (more terminals or higher capacity discs) and upgraded to boxes that allowed this increase before (and in many cases well before) they ran out of other, more fixed in capability, resources (memory, CPU and channels).

Not heeding the indirect warnings coming from HP and users who had experienced similar phenomena in DP environments such as IBM shops, resource utilization tracking seemed unnecessary and no plans for implementing this tracking were made in the average HP shop. And then one day...

"HI! I'm your friendly neighborhood BRICK WALL!"

A look with your favorite monitoring tools shows you:

CPU UTILIZATION	=	99%
I/O UTILIZATION	=	87%
MEMORY UTILIZATION	=	98%
RESPONSE TIME	=	(Well, feel like taking an early lunch? You certainly have the time!)

So, now what? You call your HP sales rep and he tells you, "We can upgrade you to a Series 70!"

"Great!", you reply, "When can I take delivery?"

"September!", he cheerfully informs you, "I know it's a little ways away, but we've had quite a few orders for them recently."

Considering that today's May 13 (a Friday, wouldn't you know it), it seems that you're not the only ones who've had this little problem.

Sound familiar? Take solace in the fact that it's a frequent occurrence. But "what if" (poignant pause) you'd known for several months that you were going to run out of CPU sometime in May and you'd long since placed your upgrade order, and HP was installing it tomorrow...

What we're talking about here is nothing new. This same type of situation has been occurring in DP shops for years. In the HP world, in the IBM world, in the DEC world, in shops with every type of computer hardware ever made. The only difference is that very few of us in the HP world are truly prepared for it.

III. THE MINIMUM INFORMATION NEEDED

To effectively track resource utilization and response trends, a minimum set of information must be maintained over a period of time. First, we must understand what the primary components of our systems are:

1) CPU
2) Disc I/O and Capacities
3) Memory

The effectiveness (or lack thereof) of the performance of each and all of these components are the contributors to the overall effectiveness of processing efficiency. The most apparent measurement of this effectiveness is the user's perceived response time. If the CPU is experiencing an abnormally high demand from the various processes executing, the perceived response time is lengthened. If an abnormally large number of disc I/O's are being required at the particular point in time, the requests are "queued" for servicing and the user must wait until the disc can service his/her particular request(s). A similar situation occurs when a process requires memory and none is currently available. When this happens, the Memory Manager is "awakened" and it is its duty to make the required space available to the process. If any (or commonly, a combination) of the above demands occurs, the user process is "stopped" until the request has been serviced.

So, it becomes rapidly apparent that tracking resource utilization over an extended period of time is required if we want to "see" how our processing demands are growing.

CPU:
Monitoring CPU utilization is probably the most critical and important of the major system resources. The reason for this is quite simple...it costs the most to replace. Luckily, except in rare circumstances (adding a new, major application, for example) does this utilization rate climb at a drastic rate. Therefore, planning for a CPU upgrade can easily be done in a time frame that allows for budgeting and ordering so as not to be caught off guard by the requirement. As stated before, however, this is the priciest of the major system components to enhance and, as such, close scrutiny to the performance of the other components should be made to determine that the bottleneck encountered or anticipated is, indeed, the processor.

DISC:
Disc is the second costliest resource to enhance. Unfortunately, it is frequently quite true that when the determination is made to add more disc to the system is required, the error factor is generally fairly low. Face it, if you need more disc space, you need more disc space. No surprises there. If you're looking for a rule of thumb, then if you've hit 85% disc space filled, you'd better call your sales rep. The assumption, of course, is that you've already purged @.@.GAMES, @.LASTYEAR.ACCTING and @.BACKUP.REFLECT.

DISC I/O:
Disc I/O is a frequently overlooked bottleneck. Unfortunately, Disc I/O bottlenecks have a nasty habit of disguising themselves as Memory Shortages. The last thing you want after waiting two weeks for your memory upgrade is to discover that your performance problem didn't go away. The common misconception is that since disc I/O rates are high, and Disc Caching is turned on, then Disc Caching must be the culprit and adding more memory will solve the problem (especially since that memory sales rep told you it would)! Now, don't take me wrong, more memory has its place, as I will discuss further, but right now we're looking at disc I/O. Entire papers and products have been written about disc caching, so I won't go into an extended discussion about it here. However, just as often as memory is the problem with disc I/O, so too is the caching configuration.

A second common problem with disc I/O is that the load placed on the discs is just too great for your current hardware configuration. This is particularly true with Series 70's. More than once has a system been configured with six 7933's all daisy-chained together nice and pretty and then hung on one GIC on one IMB. Sure, it's pretty...and pretty slow. As with Disc Caching, a multitude of papers has been presented on hardware and disc load balancing, so pursuing the details involved doesn't fall within the scope of this paper. The point being made is that before a decision to upgrade hardware is made, information must be gathered over an extended period of time and a careful analysis of the existing environment is mandatory.

MEMORY:
A shortage of memory will, without argument, cause a serious system performance degradation. If application programs are segmented poorly, or a large number of distinct processes are competing for the memory available, or the current disc caching configuration is causing a large number of write hits to dirty pages or a large number of cache domains are resident, then memory manager activity will definitely cause performance problems. However, most machines being sold today are generally intelligently configured with memory and another cause will frequently lead one to believe that a memory shortage exists. True, additional user load and/or new applications can create a need for additional memory. Unfortunately, not nearly so often as is suspected. Yes, more memory will hide a caching problem in many instances. However, a close look at and "twiddle" of caching fetch quantums will often "solve" a "memory shortage" problem. Again, more knowledgeable authors than I have addressed this syndrome in great detail and I refer you to their expertise.

IV. HOW THE "BIG BOYS" DO IT

Albeit new to the HP world, resource tracking and trend analysis is not new technology.

In one form or another, disc caching has been around for many years on other vendors hardware and fairly exacting methods for monitoring its effectiveness have been developed. The same holds true for memory utilization reporting, disc utilization reporting and response time analysis. What appears to many to be new technology, as some of you "Big Blue" expatriots well know, just isn't so.

So, what do we need to see?

RESPONSE TIMES vs. USERS

The best place to start is to determine what the "average" response time is based upon the number of users on the system. This type of trend analysis encapsulates the effectiveness of all the systems resources in the manner in which it affects the users. This historical information is valid on a daily, weekly, monthly, quarterly and annual basis.

As the "Response Times vs. Users" diagram indicates, it is possible to get a good feel for system efficiency just by tracking the average response time and mapping it into a predefined "acceptable" limit. The "acceptable" limit is a subjective response level defined by the system administrator. This is the level that the system administrator assigns as the maximum time in which a transaction must occur in order to accomplish the tasks of his/her environment.

Also provided by the system administrator is a "projected" response level, based upon past performance, that he/she anticipates the users will experience subject to the number of current users.

The "observed" response times for users are then tracked and mapped to this projection thereby providing the system administrator with a "snapshot" view of historical system performance. When a significant deviation is noted in the "observed" vs. "projected" slopes, the system administrator is quickly made aware of the fact and a more in-depth analysis can be made to ascertain the cause. Frequently, a deviation can be ignored if it still falls within the "projected" or "acceptable" limits. However, when this is not the case, subsequent reporting must be available to track the cause.

CPU UTILIZATION

Another, critical piece of information is an historical view of CPU utilization over an extended period of study. As with Response Times vs. Users, this report is useful for indicating peak CPU usage hours (daily), peak CPU usage days (weekly and monthly) and, to a lesser degree, is useful on a quarterly and annual basis.

Tracking CPU utilization quickly indicates the peak demand periods for this critical resource and provides the system administrator with the information he/she needs to determine what steps can be taken to balance this demand.

As the "CPU Utilization" diagram indicates, tracking this information over the period of interest and computing an average CPU utilization demand gives invaluable insight into growing use of the processor. Tracking the "average" information on a quarterly or annual basis shows trends in increased CPU usage and allows for proactive steps in meeting a CPU saturation point as described in the above scenario.

DISC UTILIZATION

This is the most easily understood report and is self-describing. We note that disc utilization (disc space filled) is fairly static on most systems from day-to-day. In our example "Disc Space Utilization" diagram, we note that at the end of the month, a significant increase in space utilization occurs. In our example, we've added a new, large application and it demands a fair amount of all resources. Referring back to CPU Utilization and Response Times vs. Users, they support this observation. The rule-of-thumb here is, "If you try to put ten pounds of dirt in a five pound sack, it just doesn't fit. Period."

DISC I/O RATES

Whereas, Disc Space Utilization is an important subject of monitoring, Disc I/O Rates give a more informative description of the "pulse" of your disc I/O demands. Unfortunately, there are no "magic formulas" or rules that can be followed. This is due to the fact that virtually every machine has its own unique configuration. So, what is the ceiling on one Series 58 may well be the floor on another. This is particularly true in the Series 6x, 7x environment where you can have multiple IMB's, and a significant number of GIC's, which all can be processing disc I/O's effectively, simultaneously.

The diagram "Disc I/O's Per Second" shows that we've established an "acceptable" limit of 70 I/O's per second. This configuration is on a Series 48 with two 7933 drives each on their own GIC. At an average rate of 35 I/O's per second each, an "acceptable" rate of 70 I/O's per second total is within limits. If, on the other hand, You have a Series 70 with two IMB's, at a supported limit of two high-speed GIC's per IMB and a single Eagle drive on each of those GIC's, you could multiply the 70/sec rate by a factor of 2.5 and have a more-than-reasonable "acceptable" level.

So, you can see that this is certainly an environment specific mandated report. The concept, however, is applicable to any business computing environment.

MEMORY UTILIZATION

Memory Utilization is, probably, the most misunderstood concept. Each vendor has it's own method of memory management and in some environments, the actual percentage of memory currently occupied is a valid measurement of memory requirements. This holds true on the classic HP 3000 for those systems that do not have disc caching nor AUTOALLOCATE turned on.

In our diagram "Memory Utilization", we note that memory utilization starts at approx. 91% and rapidly hits the 100% mark. This, in and of itself, doesn't necessarily indicate that we are experiencing a "memory pressure" situation. An initial glance at this report leads the experienced system manager to believe that AUTOALLOCATE is in effect and that further study is required. If, on the other hand, AUTOALLOCATE is not in effect, and a look at SHOWCACHE doesn't alarm us, then we probably need to assess our memory capacity and seriously consider adding more memory to the system, or modify our caching configuration to better utilize the memory currently available.

An excellent (excellent=inexpensive) method of assessing caching effectiveness is to use CDTMGR (contributed by Bryan Carroll, HP) found on the VEGAS swap tape. This little tool provides a real wealth of cache performance and lets you monitor the effectiveness of any cache configuration changes you make.

MEMORY MANAGER ACTIVITY

Probably the most accurate method of analyzing memory effectiveness is by monitoring Memory Manager activity. This means that you wish to observe how often the memory manager is required to "make room" for a particular structure (code, data, etc.). Although the measurement varies from vendor to vendor, the concept is the same regardless of the type of hardware you have. Whereas, on a FIREBLAST 6000, memory pressure is measured in nibble-faults-per-footpound; on a classic HP 3000, memory pressure is measured in cycles-per-second. Conceptually, the activity is the same; some indicator of memory management is used to measure how much time the Memory Manager is spending servicing the memory requirements of a particular process.

As a brief synopsis, on the classic HP 3000, when a request is made by a process for memory, the memory manager "cycles" through memory looking for available space to place the required structure in memory. The greater number of times (per second) that the Memory Manager needs to look through all of the memory available to find (or create) that space, the longer the process has to wait to continue.

As in the case of our example ("Memory Manager Cycles"), for the majority of the month, Memory Manager "found" the space it needed with a minimum of cycling required to satisfy the request. However, we note, at the end of the month (after loading our new application) the number of times that **MM** had to cycle through memory increased drastically before a particular request could be completed. So, although Memory Utilization certainly didn't indicate that we were experiencing memory pressure, Memory Manager **Activity** certainly did!

V. HOW IT APPLIES TO MPE XL

Now that we have an historical perspective of the concepts of resource utilization tracking, along with an indication of how it applies to the classic HP 3000 environment, let's turn to the applicability of these concepts to the MPE XL environment.

In the MPE XL environment we have the following resources that contribute to the processing of a particular task:

1) CPU
2) DISC
3) MEMORY

So what's new? ...Nothing.

The only **change** in that processing is the **manner** in which certain tasks are performed and in the **capabilities** of the resources. CPU is faster. Disc I/O is faster. Memory is more **abundant** and addressability is **greater**. But the concepts are <u>exactly the same</u>!

RESPONSE TIMES vs. USERS

We still have online users who still have a perceived response time from the system. Therefore, we still need to track (and project) what that response time is based upon the number of current users.

CPU

We still have a CPU that is capable of performing a finite (albeit greater than the classic HP 3000) number of tasks in a given period of time. Therefore, we need to track the utilization of that processor over a period of time to determine what our historic needs have been, what our current needs are, and to project what our future needs will be. This becomes particularly important in the HPPA/MPE XL environment due to the cost and availability of upgrades. They're inevitable, so we need to anticipate when they must occur and plan and budget for them accordingly.

DISC UTILIZATION

The only surprise we may experience here is that we use up disc space faster than we did in the classic HP 3000 environment.

This is no surprise. In a RISC based architecture, the compilers generate more numerous instructions to accomplish a task than do their CISC counterparts. Additionally, we've grown into the HPPA environment, basically, due to the fact that the shear volume of our processing and data demands have increased. Additionally, with the advent of virtual mapping of data structures, among other items (such as the size of MPE XL, itself), our "virtual memory" requirements have increased.

DISC I/O RATES

Certainly, Disc I/O rates will increase due to the fact that we now enjoy the benefit of a much more efficient I/O facility. However, we can't ignore this aspect of resource monitoring. The concept of load balancing (both file locality and physical hardware configuration) doesn't change. If a channel or disc is busy, it's flat busy, and the hardware can only perform a single item of work at any single given instant.

MEMORY UTILIZATION

For the exact same reasons mentioned above, if there is memory available, it will be used. The only real change that we will note here is that the Disc Caching Facility (actually, the conceptual process) is now an integral part of the I/O and Memory facilities. To "change caching configuration" becomes a meaningless concept in that the function is, essentially, performed dynamically. The MPE XL experts out there, I'm sure, will take issue with the "imbedded caching" analogy. For their benefit, I confess that the technical implementation is far more sophisticated and pervasive than the analogy leads one to believe. Please bear in mind that the intent is to convey the concept and not to discuss the technical aspects of the implementation of that concept. So, to them, I extend my apologies and challenge them to make a presentation describing how, from a technical perspective, my analogy is meaningless.

MEMORY MANAGER ACTIVITY

As in any other vendor environment, measuring Memory Manager activity is still the best method of evaluating the effectiveness of the memory resource. If Memory Manager activity is high, something needs to change. A close scrutiny of the HPPA implementation clearly reveals that memory is a critical (!) resource to efficient throughput. I now start to support the memory proponents to a greater degree in the assumption that "More Memory Makes Bottlenecks Disappear." Although, as is true in any other environment, this is not an absolute, in the MPE XL environment it is an excellent **first** assumption. Series 9xx buyers take note: BUY LOTSA MEMORY!

VI. WHY YOU NEED TO START NOW

It has been the intent of this paper to make the reader aware of the necessity and benefits of performing an on-going function of resource monitoring and utilization trend analysis.

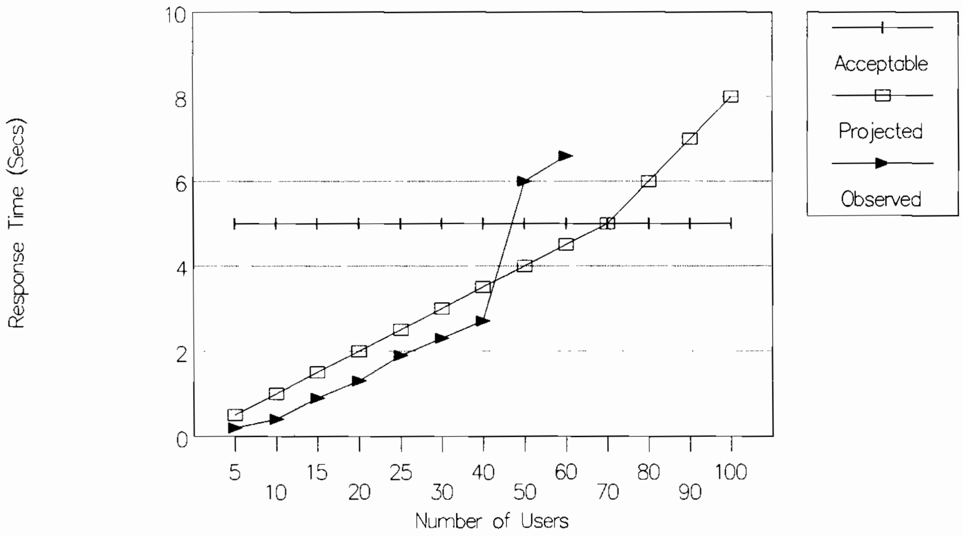
As has been pointed out, this function is applicable in any computing environment (yes, even on micros, to a certain extent). The idea is to implement a vehicle for informing the user of the use and abuse of available system resources and to have the information available that assists him/her in planning for future growth.

As our examples have shown, a fairly minimum set of information can provide enough data to the planning process to inform you in a timely manner as to when you can expect to increase the quantity or capability of a resource that is approaching a saturation point.

For those of you who are planning to upgrade to the HPPA environment, this can be of exceptional benefit. On the first hand, it may well show you that enhancing a particular resource may enable you to defer the upgrade for an extended period of time. Just as important, this information can give you concrete data for presentation to management to justify the upgrade for which you have been pushing. Either way, **knowing** what is happening on your system ensures that you're prepared to make whatever changes you will need to make in an informed, intelligent manner.

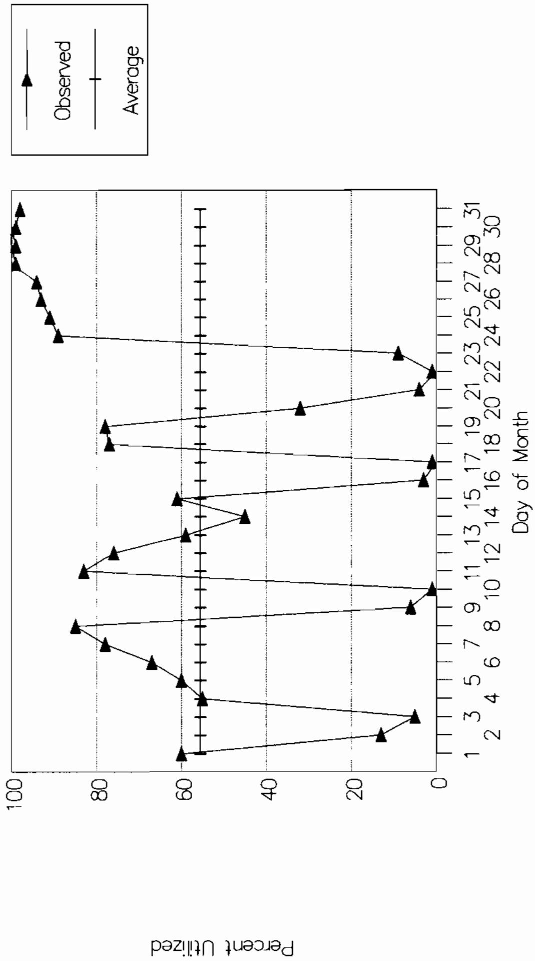
Response Times vs. Users

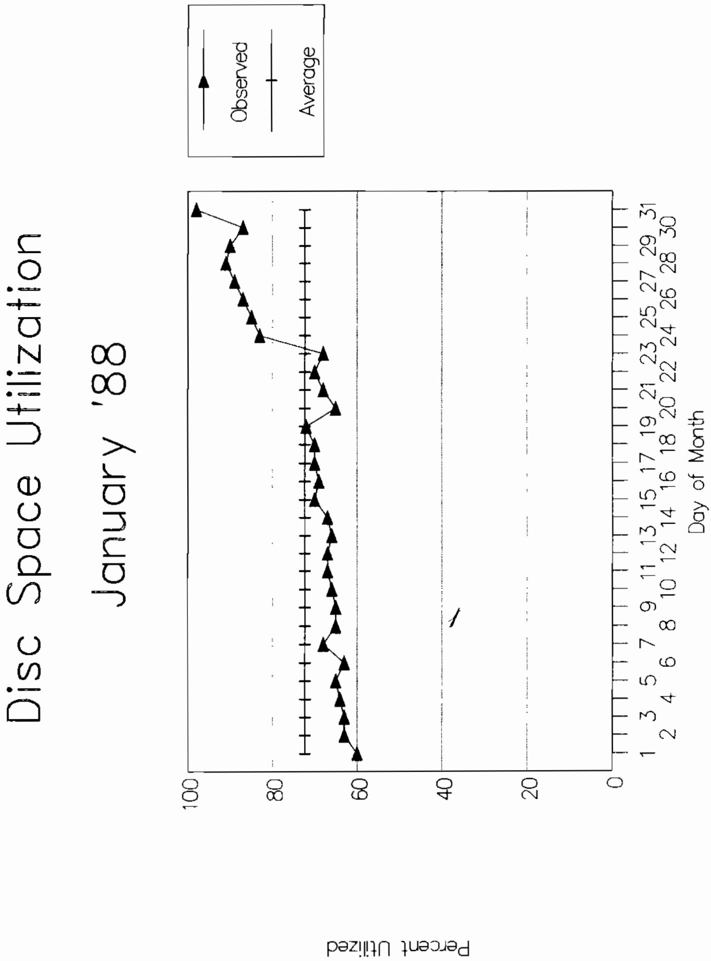
January '88



CPU Utilization

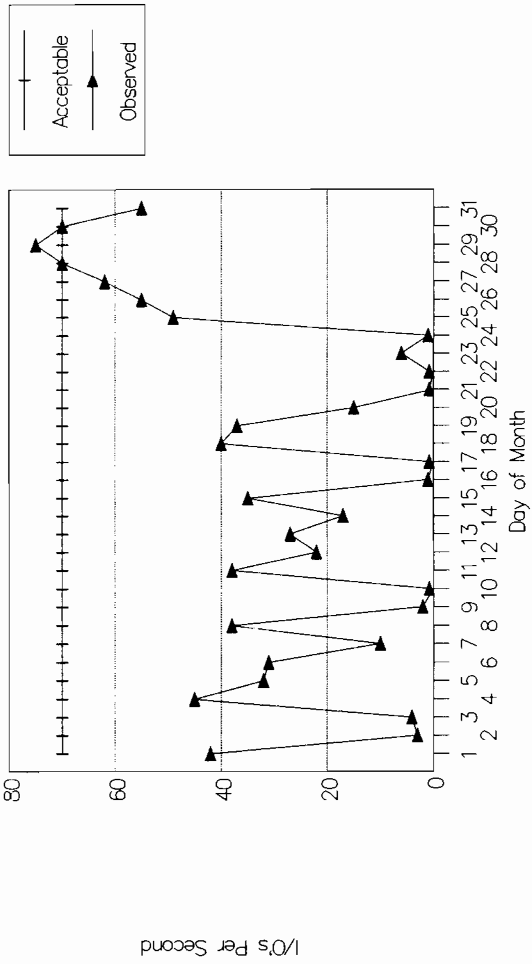
January '88

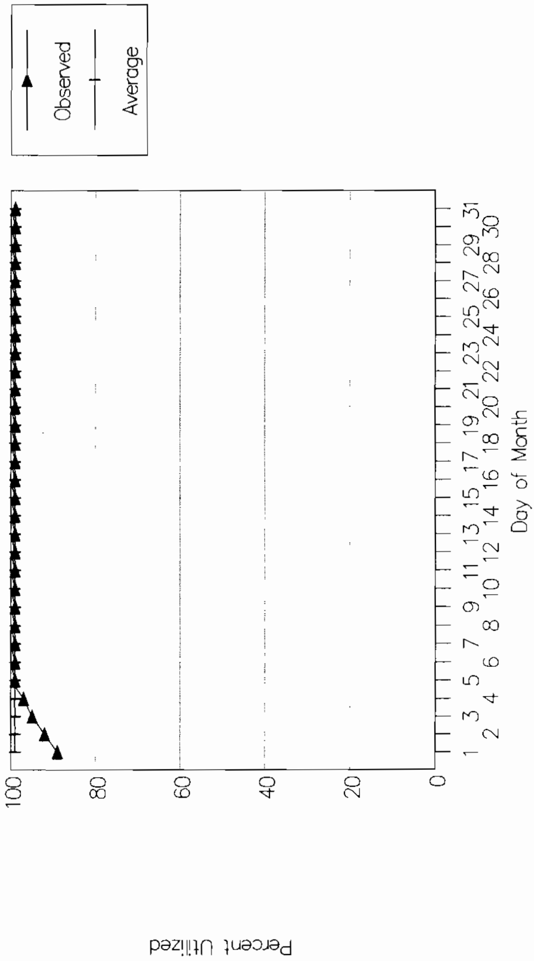



Disc Space Utilization
January '88
Observed
Average
100
80
60
40
20
0
Percent Utilized
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Day of Month

Disc I/O's Per Second

January '88

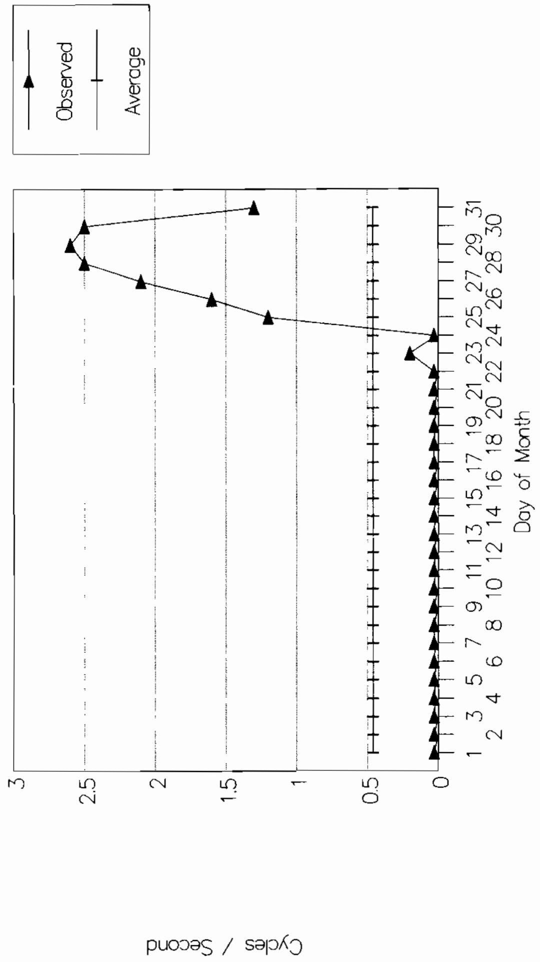




Memory Utilization

January '88

Memory Manager Cycles
January '88



SALES FORCE AUTOMATION: A Case Study

Nolan M. Alexander
Bepex Corporation
P.O. Box 880
Santa Rosa, CA 95402

BACKGROUND --

Bepex Corporation is an international company with facilities in Chicago, California and Minneapolis in addition to sites in Scotland, Holland and West Germany. Bepex manufactures process equipment for the food and chemical processing industries and maintains a domestic (U.S. and Canada) sales force of nine people.

PROJECT HISTORY --

Bepex Corporation's entry into the area of Sales Force Automation began in 1985 when the company president tasked the Information Services Department (ISD) to investigate the possibilities of placing micro computers in the offices of our field sales representatives. At the time the specific goals of the project were vague but centered around electronic mail and improving the processing of call reports. Information on several systems was collected and forwarded to the president but no action was taken and the project was placed on hold in favor of more pressing needs.

In mid 1986 ISD was again tasked to evaluate possible systems for the field sales offices. At this time we examined the HP Portable, the HP150, the Telecompaq and the HP VECTRA. The VECTRA was recommended for its ability to be a HP terminal (via AdvanceLink 2392) and its IBM software compatibility. By now the project was defined as trying to improve communications with the field sales offices. Specifically, to get call reports on a more timely basis and respond to requests from the field and possibly, to allow direct access to manufacturing systems and order status in the HP3000 by the salesmen. Estimates of the hardware and software costs were provided to management along with a schedule for implementation.

In late 1986 the project suddenly moved into high gear. Could ISD acquire at least one sample system and prepare a presentation/demonstration for the annual sales meeting the following February? Simultaneously, Hewlett-Packard was introducing Sales Force Automation to the installed base. Our HP salesman was aware of Bepex's interest in this area and urged us to attend the Sales Force Automation seminar. The success of HP test program and the transformation of the Portable to the Portable Plus caused us to recommend to the company president that we scrap the plan to put VECTRA's in the field and use the Portable Plus and HPDESKMANAGER IV. This last-minute change caused the president to seriously doubt if we really knew what we were talking about but he did agree.

The system configuration settled on was:

HP PORTABLE PLUS with 512K and built in modem
Think jet printer
Portable disk drive
Advance Mail with Reflection software
Lotus 1-2-3
Memo Maker with Time Manager
Executive Card Manager

After shopping around several sources we entered into a purchase agreement with a local HP dealership for eight systems and took delivery of one system right away for prototype development and as a demonstrator for the sales meeting.

The entire project was now more clearly defined. It would consist of two principal phases.

PHASE I:

- Get the portables into the field sales offices.
- Install HPDESKMANAGER IV on the HP3000.
- Start the flow of communications between the company offices and the field sales offices via electronic mail.
- Give the sales force additional tools in the form of the software on the Portable Plus.

PHASE II:

- Additional Portable systems available for executives, managers and salesmen to use when away from their offices in California and Minneapolis.
- A general inquiry subsystem to be written for Bepex users that will allow them to log onto the HP3000 and, in a single log-on, access inquiry screen for any Bepex software system. A kind of executive level inquiry.
- Custom basic program for the portables that would contain proprietary formulas to be used as sales tools.

Now we had a plan and a prototype; the only other thing we needed was a couple of salesmen with the willingness, or even better -- enthusiasm, to be the beta test group. It was decided to present the project at the annual sales meeting and hope that at least two volunteers would surface. Taking a page from Hewlett-Packard's book, the presentation was based on a series of "What if's". What if a salesman could directly access the order database or what if he could have direct access to the desk of the company president when a $500,000 order hung in the balance? What if he could prepare customized charts and graphs based on his customers' requirements? What if he could take word processing, electronic mail and spreadsheet capability on the road with him in a machine that could withstand the everyday bumps and grinds of traveling? The response to the presentation was much more than our expectations; out of eight eligible candidates we had eight eager volunteers.

TRAINING --

Given the response from the field sales force we decided to go ahead with a group of four, instead of two, for our pilot group. The four systems were delivered to the California facility where they were assembled by ISD and configured. Then the systems were shipped to Chicago where the training would take place.

The training program is eight hours of introduction to personal, portable, computing and was segmented into three sessions. The first session was four hours and covered the following:

Hardware (setup and basic operations)
System configurations
Memomaker
Time manager
Introduction to Advance Mail

At the end of the session the students were given reading assignments and exercises to complete for the next day. The second session was two hours and covered Advance Mail exclusively. Most of this session was spent generating messages, transferring them via modem to the 3000 in California and picking up mail from the 3000. At the end of this session the students were given additional reading assignments and exercises. The final two hour session consisted of an hour's review of Advance Mail and an introduction to Lotus 1-2-3 and Executive Card Manager.

The training sessions were very carefully designed to keep in mind that the students had little or no background in personal computing. Careful attention was given to terminology, specifically to define new terms and to avoid using interchangeable terms that tend confuse new users.

Now that the field personnel were trained it was necessary that they have someone to communicate with. The training of HP3000 users in HPDESK was started. Users were trained individually or in groups of two. In review, this was probably a mistake. It used a lot of time on the part of the trainer and produced results too slowly. Another problem was encountered involving the terminals. Most of the key personnel that we needed to get into the electronic mail network were using 264X series terminals. These terminals are supported by DESKMANAGER IV but not for all features. The most critical feature not supported was the function key labels. These labels are critical to new HPDESK users because they remind the user what his or her options are in any given function of HPDESK. The 2640B terminal (we had four) doesn't support the full screen edit function for entering or editing text and the users found that the subset of Editor was difficult to learn and use.

CURRENT STATUS --

At this time, seven systems have been distributed to the field sales personnel. Two salesmen do not yet have machines but they operate out of offices in California and Minneapolis where they have access to HPDESK via terminals. All HP3000 users have been trained and older model terminals have been replaced with units that support all the functions of DESKMANAGER IV. Phase II of the project has been postponed in favor of more pressing requirements for manufacturing operations. Phase I of the project is complete and is considered by all involved to be a success.

All personnel who are part of the network are required to check their electronic mail at least once each day. Portable users tend to perform this function more on the order of twice each day as they generate the most traffic.

Portable Plus users are also reaping additional benefits from the additional software packages available on the Portable Plus. MemoMaker is allowing them to perform light word processing duties and with the recent announcement of ink for plain paper this feature has become even more valuable. Executive Card Manager allows the salesmen to maintain a personalized database of their customers and contacts. Lotus 1-2-3 provides a means to calculate and plot technical data for presentation to clients and sales prospects.

USER EVALUATIONS --

The following are excerpts from evaluations of the Portable Plus, Advance Mail and other software by the salesmen.

"Basically, the HP-Portable Plus System is a welcomed addition to my office. The packaged programs have given me flexibility in my one man field sales operation... ...my written response time to customer requests was slower because most responses were hand written prior to giving the information to the secretary. This doesn't mean I don't need a secretary, but my need is reduced by using form letters saved on the discs and getting urgent letters out quickly."

"Telephone tag is a way of life today. The system has allowed me to send timely messages to our various offices without wasting time trying to get the party on the phone..."

"The system is excellent and easy to use with the exception of the users manuals. They are totally confusing. I found the Commodore manuals and software manuals more easy to use."

"Lotus 123 has been useful for Field Test Reports. Graphing techniques and spread sheets have been helpful in preparing these reports. However, it was absolutely grueling to learn how to graph from both the tutor and manual. "

"Time Manager: I use Calendar for important business followup calls. The name & address files are used for direct dialing of important customers with info such as the secretary's name and her extension."

"After several days of operation I would like to offer the following comments.

The only program I've tried is AdvanceMail. This system of transferring messages is excellent and I think it eventually will prove to be much more than adequate in meeting our goals of improving communications.

I know I'll put much more in writing than I tended to in the past because I don't have to deal with a trip to the Post Office and the subsequent 3-4 day delay in transit."

"The system meets all expectations I may have held. If anything, it is easier to use..."

"... system meets all expectations - is "user friendly" - and, as a complete neophyte with computers, I've had minimum trouble picking up operation. One area of shortcoming - some of the manuals are hard to learn and confusing - makes for a lot of frustrations in mastering a feature, particularly ECM manual."

"Most useful to date - Time Manager - also like it best..."

"Time Manager: Most useful - I work more efficiently, get more tasks done in one day, etc. helps with all planning for each day, week, and long term, etc. This feature/product can organize anyone, and make them more efficient - should never miss a followup date..."

"Executive Card Mgr: Most useful - One disk stores so much. Like the ability to call up key fields for trip planning, or installation data with customers. This feature greatly enhances efforts for sales calls in key areas - shortens planning time. Also great for customer contact with autodial feature."

ISD EVALUATION --

Phase I of this project must be counted as a success. The users of the the system, those on the HP3000 and those using the Portable Plus, are enthusiastic and all agree the system has enhanced our ability to communicate between the factory and the sales offices. There were however, some areas that could have been improved upon.

1. The Portable Plus users were trained and equipped before the HP3000 users. This was clearly an error in timing. The portable users were left with all that new capability and potential but had no one to communicate with except each other. We were fortunate their enthusiasm carried them through the times they had to wait for someone to be trained to receive their messages.

2. We didn't understand the level of difficulty that would be encountered by trying to use HPDESK on 264X terminals. The delay in replacing these units hurt the project by preventing key personnel from becoming part of the system sooner. We also feel HP could have made a stronger point of just how much more difficult it is to use HPDESK on these terminals.

3 Installing and configuring the Portables was straight forward and greatly aided by the manuals and installation guides except for one area. The Reflection software bundled with the Advance Mail product is totally dependent on the batch and configuration files that are used to accomplish the various modes of mail transfer. Yet, there is only one

documentation file on the utility disc along with several sample files. This documentation is clearly insufficient! An experienced user and system analyst with more than fifteen years in data processing had a great deal of difficulty in putting together files that would do the job. The final result was a combination of educated guesswork, trial-and-error, calls to the response center, calls to other users and luck. An inexperienced user with little or no data processing or datacomm background hasn't got a prayer! Generally speaking, the user isn't supposed to have to do anything with these file anyway. It's the job of the HPDESK Administrator. Unfortunately, the HPDESK administrator manuals don't provide any relief either.

It is our opinion that if HP is going to sell the Reflection software as part of a bundle that they ought to bundle the documentation with it too. Many useful features of Reflection remain unknown to us until we come across them by accident. For example, configuring Reflection on the Portable Plus to emulate the HP 2624B instead of the HP 2392A will enable forms cache and, at 1200 baud over the modem, that is a significant feature.

4. When we first put this project together we had leaped at the chance to dazzle our customers with our high-tech capability by using the Portable Plus from their offices to check on order status or product availability. We have since moved away from this policy, indeed we now tell our salesmen to never use the units in the customer's office for communications. In most remote transfers the call goes through on the very first try but, through no fault of the user or the computer hardware, there are times when the call will not go through at all. Data communications via voice grade telephone lines has improved much in the last few years but it is still far from perfect. To avoid the acute embarrassment of being unable to make the connection after pumping up the customer we insist that our salesmen not take the chance. We do however, encourage them to use the stand alone features of the system in the customer's office if it will help the sale.

FUTURE PLANS --

Company management and Information Services are currently redefining Phase II of this project. Early in Phase I we planned for a time when Portable users would use the units to call up the system and log-on as a regular interactive session to interrogate the manufacturing and order databases for information concerning their orders. We no longer view this as a desirable and, interestingly, neither do the salesmen. They prefer not to have to log-on and look up the data but would rather have a order status report, for just their orders, sent as a message on Monday mornings.

Another feature we're investigating is setting up access for the other companies who act as our sales representatives. Using Security 3000 from VESOFT, we are confident about protecting our system and applications while providing these companies with the limited access they need. While these companies would probably not be using Portables, we feel that DESKMANGER IV, with its Foreign Service Connection, is the appropriate avenue this type of data exchange.

Bepex Corporation is satisfied with its entry into this area of data processing and the value of our investment in hardware and software to support it. We view the future of the project as open-ended, continuing to evolve in tandem with the other systems we use to support our manufacturing and marketing efforts.

Unorthodox IMAGE Accessing for Power

Joseph Berry
Kiryat Telz Stone 116A
D.N. Harei Yehuda, Israel

Noel Magee
Telephone Employees Credit Union
123 S. Marengo Avenue
Pasadena, CA 91101

Introduction

Traditionally, large databases are difficult to manage and manipulate. Datasets with millions of records cannot be unloaded and loaded whenever desired. Broken chains cannot, therefore, be easily repaired. Large numbers of records cannot easily be deleted.

For databases meeting certain logical and physical criteria, a technique is presented here for performing the almost instantaneous removal of millions of historical records from a detail dataset and moving them to an archive. This technique additionally allows access to as many of those archived records as the user wishes, subject only to the constraint of disc space. Modifications to application software is minimal. As many detail records as are present will be accessed by the application software.

Additional advantages to this technique include never having a fragmented detail data set and being able to produce historical reports in a very timely manner, even when millions of records need to be scanned.

Description of the Problem

At the Telephone Employees Credit Union we provide credit union services to communication industry employees in the Southern California area. We are, at present, the eleventh largest credit union in the United States. In addition to standard savings and withdrawal accounts, we provide access to over 10,000 Automated Teller Machines, lines of credit (both personal and equity based), Automated Voice Response, interbank/automatic funds transfer and other standard financial services. In order to provide these services in a timely fashion we use two Series 70s at a central site running Silhouette/3000 and Filepro from Carolian, MTS, RJE, NS/3000, DS/3000, and a variety of other third party packages. Our ten branches (located over the Southern

California area) communicate with the central site using 3270 protocol and remote 3274 look-alike controllers.

During the past seven years TECU has experienced a growth rate of 10-20% per year in transaction volume. In 1988 we averaged 40,000 online and 30,000 to 40,000 batch transactions per day. Analysis showed that the growth rate would problably not change in the near future. Additionally several new services needed to be implemented in the 1988 time-frame. These increases were reflected in two primary history data sets, HSTRY-SHARE-FILE and HSTRY-LOAN-FILE. From a base of 400,000 entries in 1979 the HSTRY-SHARE-FILE grew to over 4 million by September of 1987 and occupied some 1.5 million sectors of disc. This, of course, presented us with a file on the verge of all MPE and IMAGE limitations.

The principle difficulties with which TECU was attempting to deal were, first, the deletion time for inactive records and, secondly, the disc space required for so large a file. In the former case we had a deletion program which ran some 56 hours simply to delete the oldest thirty days of history. This amount of time was required due to the fact that the structure was a single large IMAGE detail. In fact, prior to Turbo-IMAGE, we could not do daily reporting from this due to the fact that a chain, keyed by date, would have been too long for IMAGE. Secondarily, the file was so large that, while one might have sufficient disc space in total, the extent sizes sometimes prevented a simple restore of the file. Often we ended up condensing volumes simply to acquire large enough blocks of free space to accomidate the extents; more often, we ran out of space. Also, since we updated the file on a daily basis, it was being stored on a nightly basis. This resulted in from one to three additional tapes (at 6250bpi) being generated per night and from 9 to 27 additional minutes of down time. Both of these problems became critical when, in 1985, we moved to 18 hours/day operation with the addition of proprietary and networked ATMs. In 1986/1987, the problems were compounded when we moved to a new headquarters building and started to look at full 24 hour, non-stop availability. Fumdamentally, we could no longer afford either the maintenance nor the disc space problems in our coming environment.

Proposed Solutions

Prior to the changes to be described, share history data was kept in the database, TECUDB, in the HSTRY-SHARE-FILE detail data set. This data set was chained off the manual master, SUB-SHARE-FILE.

```
      ------
       \  /   SUB-SHARE-FILE
        --
        |
        v
      -----
      |   |    HSTRY-SHARE-FILE
      -----
```

The decision was made to solve the problems associated with the large HSTRY-SHARE-FILE detail data set and a number of alternative solutions were considered. These included multiple KSAM files and multiple IMAGE data sets.

The primary advantage of KSAM as the file mechanism for storing the data was that since it was not inside an IMAGE structure, maintenance would be easier. It's disadvantage was that it did not have the nice item lists of IMAGE; thus entailing more "up front" work (i.e., more severe modifications to existing programs). This difficulty could be overcome by making the KSAM file self-describing and the appropiate user SL routines cognizant of that fact. The dynamic allocation of storage space which KSAM uses would save disc usage. The greatest disadvantage of using KSAM was the time required to build the key file whenever the data were moved from the TECUDB database.

Two IMAGE designs were also investigated. One design consisted of a number of data set pairs as shown here:

```
   ------    ------    ------
    \  /      \  /      \  /
     --        --        --

     |         |         |
     v         v         v

   -----     -----     -----
   |   |     |   |     |   |
   -----     -----     -----
```

This technique required that a separate database be built containing pairs of data sets for each month (or time period) in the past that is desired. Each master/detail would have the same format as the current master/detail.

A number of advantages were presented by this design. Each period's history data is defined as an data set pair. If there was not enough room on the system for all the historical data, then as many data sets as would fit would be maintained. To delete a month's worth of history data, the data set linkage for that month wouldn't really have to be erased. It is possible to physically purge the two data sets and rebuild them. This procedure would take a minimum amount of time. There is no reason to limit the number of time periods. Depending on disc space availability, many more months' worth of data can be stored. In all cases, the "active" number of data sets and which pair of data sets represents which time period would be determinable by inspecting a stand-alone control master data.

A major disadvantage of this technique is that the large master data set must be duplicated for each month. This would be a large waste of disc space. Furthermore, in order to profit from the purging and rebuilding of individual data sets, privileged mode (PM) access will probably have to be used.

An alternative variation to the above IMAGE design was to link all the detail data sets to one master data set. This solution eliminates all the excess disc space. However, similar to KSAM, an inordinate amount of time would be spent building (or updating) the pointer information in the master data set as historical data is added to it.

```
                ------
                 \  /
          ------  --    -------
          |       |          |
          |       |          |
          v       v          v

       -----    -----      -----
       |   |    |   |      |   |
       -----    -----      -----
```

Proposed Solution

Given the problems associated with the database TECUDB, we will describe an efficient and effective solution for "splitting up" the HSTRY-SHARE-FILE data set of the TECUDB database, making it considerably more manageable.

Our minimum requirements were to continue maintaining more than three months' worth of history data in order to process statements and quarterly postings. An important point here is that except for the current month, no information is added or updated (the only exception being by the TECU Information Services staff when repairing bad data).

The proposed enhancement required a change in the relationship between these two data sets. We changed the data set structure to the following new structure:

```
    ------        ------
     \  /          \  /     SUB-SHARE-FILE-A (AUTOMATIC)
      --            --
 SUB-SHARE-FILE     |
   (MANUAL)         v
                  -----
                  |   |     HSTRY-SHARE-FILE
                  -----
```

We took the SUB-SHARE-FILE data set and removed the connection to the detail data set (how this was accomplished is explained further). In addition, we added a new automatic master that essentially replaced one of the functions of the previous SUB-SHARE-FILE, i.e., chained access to the detail. The effect of this change on existing programs that access either of these data sets is interesting: no program changes were required! Note that if a chained read into HSTRY-SHARE-FILE is needed, the DBFIND references the name of the detail data set and not the master (therefore, no change). If a by-key access is required into the master to retrieve stored information, the DBGET references the master data set name, SUB-SHARE-FILE, which is now the name of our standalone master.

Obviously, data sets SUB-SHARE-FILE and SUB-SHARE-FILE-A have the same key, ACCOUNT-SUFFIX. Due to changes in the blocking factor of the SUB-SHARE-FILE, the resulting structure with three data sets actually consumes less disc space than the old structure consisting of two data sets.

With respect to performance degradation (i.e., increasing numbers of I/Os due to the additional data set), there is a very slight increase in the number of total I/Os. Specifically, adding a record to the HSTRY-SHARE-FILE data set with the existing schema took approximately seven physical I/Os (not including logging). With the three data set structure, the number of I/Os increases to eight.

In addition to the changes in the structure of the TECUDB database, a new, additional database, called HISTDB, was added to the system having the following structure:

```
    CNTL-MSTR   A-MSTR    B-MSTR    C-MSTR

     ------     ------    ------    ------
      \  /       \  /      \  /      \  /
       --         --        --        --

                  |         |         |    . . . .
                  v         v         v

               A-DETL    B-DETL    C-DETL
                -----     -----     -----
                |   |     |   |     |   |
                -----     -----     -----
```

In an operational mode, the automatic master and detail data set from the TECUDB database will be moved into this new database when the data in the original detail data set fills beyond a certain limit. Each pair of master/detail data sets will represent one arbitrary length of history data. The master data set is a duplicate of SUB-SHARE-FILE-A (which is now much smaller as an automatic master data set) while the detail is a duplicate of HSTRY-SHARE-FILE. Database TECUDB will only contain data for the current period. The CNTL-MSTR data set, pictured above, contains various kinds of identification information.

A number of advantages result from this solution:

1. When deleting the history data, the data in HSTRY-SHARE-FILE doesn't really have to be erased. The two data sets are simply purged and rebuilt. This procedure takes almost no time. In reality, instead of purging this pair of data sets, we rename them into the database that stores the historical information. We then create a new pair of empty data sets for TECUDB. Therefore, there is never any deleting of the records in HSTRY-SHARE-FILE (more on this bit of magic later).

2. There is no reason to limit the number of time periods. Depending on disc space availability, many periods' worth of data can be stored. In all cases, the "active" number of data sets and which pair of data sets represents which period is determined by inspecting the stand-alone control master data data set.

3. If a broken chain is found in the master/detail data set pair of the active TECUDB database, the repair can be accomplished as soon as there is a period turnover. This data set pair is moved to our new database and is only read accessed. A duplicate of the data set pair can be repaired on one of the backup computers and when corrected, these two files would replace the original ones.

An interface procedure, HDBGET, was written to allow all the applications to access either the normal data set pair or one of the data set pairs from the new HISTDB database in a transparent manner. The interface procedure accepts almost the same calling sequence as DBGET. It is responsible for calling the real HSTRY-SHARE-FILE data set in the TECUDB database. If, during a backwards chained read we arrive at the beginning of the chain in this data set, we must then access the most recent data set pair in the new database and return its information to the user. We must then store information reminding us of where we were and in which database in order to continue our chained read. This information must also be stored for serial access reads.

All programs that perform DBGETs on the HSTRY-SHARE-FILE needed to be modified. Instead of calling DBGET directly, these programs called HDBGET. The size of the status array was slightly enlarged and two parameters were added.

Structure of the HISTDB Data Base

The HISTDB database, when in normal operation, consists of zero or more (up to 24) physical pairs of SUB-SHARE-FILE-A masters and HSTRY-SHARE-FILE details. These data set pairs are moved into HISTDB via "RENAME"s of the data sets (see the HISTUTIL program below). In addition, there may be zero to 24 pairs of data sets associated with loan information (this is not currently implemented except in HISTDB and procedure HDBGET).

In order to fool IMAGE into believing that this database actually has so many data sets (48 pairs x 2), the schema file was actually configured with the 96 data sets.

In addition, there is a stand-alone master data set that controls access to the various data set pairs. This data set marks which data set pairs actually exist and what their date ranges consist of.

Below we see the layout of the CNTL-MSTR manual master data set. This data set contains two records: one for transaction history information (the detail data set being HSTRY-SHARE-FILE) and the second being for loan information (the detail being HSTRY-LOAN-FILE). Within each record, the remaining fields are replicated (indexed) 24 times. This represents the 24 possible data set pairs that can be present in the HISTDB database.

Each index represents one data set pair. The indices are maintained in backwards chronological sequence (i.e., the most recent date is in index 1, the first index). Since the data in the detail data set (as it came from the TECUDB database) was never modified, the data is always in physical chronological order. Therefore, by examining the first and last records of the data set, the date/time range can be quickly determined.

There is really no connection between the data set pairs and where they are located. The data sets of HISTDB were arbitrarily given the logical names HISTDB02, HISTDB03, ..., HISTDB97 (the master data set was HISTDB01). Thus, the logical and physical names of the data sets are identical. When a data set pair are moved from database TECUDB to HISTDB, information is inserted into the control data set according to the BEGIN-DATE date. The name of the detail data set that is assigned to this pair is inserted into variable DS-NAME. In this way, the assignment of the data set pairs can be managed, controlled, and accessed by the data contained in the CNTL-MSTR data set.

Layout of CNTL-MSTR file:

```
  key
------------------------------------------------------------------
|SET|BEGIN-DATE | END-DATE | BEGIN-HOUR | END-HOUR | DS-NAME|
|ID |    x24    |   x24    |    x24     |   x24    |   x24  |
------------------------------------------------------------------
```

Elements in CNTL-MSTR:

Element	Type	Size	Remarks
SET-ID	I	1	Data set number of the data set from the TECUDB database (i.e., there will be one for the HSTRY-SHARE-FILE and one for the HSTRY-LOAN-FILE).
BEGIN-DATE	I	2	YYMMDD of beginning date of dat set pair. 24 subitems.
END-DATE	I	2	YYMMDD of ending date of data set pair. 24 subitems.
BEGIN-HOUR	I	2	HHMMSS of beginning time of data set pair. 24 subitems.
END-HOUR	I	2	HHMMSS of ending time of data set pair. 24 subitems.
DS-NAME	X	16	Name of the detail data set that contains the records for this date/time range. 24 subitems.

HISTUTIL Utility

HISTUTIL was designed to perform two basic functions: (1) To take HSTRY-SHARE-FILE data set pairs from the TECUDB database and move them to the HISTDB database, updating its CNTL-MSTR data set. (2) To store/restore portions of data from the HISTDB database to and from tape. We designed the syntax of HISTUTIL to always relate to the HISTDB database. This was necessary because we needed a consistent point of reference for all commands.

This program, written in PASCAL, performs privileged mode functions without actually going into privileged mode (PM). The main reason for requiring PM was to be able to rename the data set pairs from the TECUDB database into the HISTDB. TECU already owns a program that can perform this function: MPEX from VESOFT, Inc. A phone call to VESOFT gave us the technique for communicating program-to-program with MPEX. This is accomplished with the MPE MAIL intrinsics: SENDMAIL and RECEIVEMAIL. HISTUTIL, therefore, created the MPEX process as a son and transferred commands to it via SENDMAIL. The following two procedures demonstrate this (error branches have been removed and procedure calls simplified).

```
procedure create_mpex (var pin:smallint);
var
   progname : string[30];

   procedure  createprocess;  intrinsic;

    { beginning of procedure }
begin
progname := 'MPEX.PUB.VESOFT ';

createprocess(progname, pin);
end;



procedure talk_to_mpex(pin:smallint;var message:string);
var
   status, messlen : smallint;

   procedure activate;           intrinsic;
   function sendmail : smallint; intrinsic;

    { beginning of procedure }
begin
message := '!' + message + ' ';
messlen := strlen(message) div 2;
status := sendmail(pin, messlen, message, 1);
```

```
    activate(pin, 3);
    end;
```

Since all the PM code was isolated into a known program, MPEX, it was easier and certainly less "dangerous" to debug. The other function that MPEX performs is a copy function. When HISTUTIL renames the data set pair from being a part of TECUDB to HISTDB, an empty data set pair must be copied into that location. For this purpose, we created a special group called HOLD that held an empty version of those two data sets (i.e., two IMAGE data set files). The MPEX "FCOPY" command with the ",FAST" option was used to copy these two privileged mode files into the proper group. The copy is usually done automatically after the rename unless overridden with a HISTUTIL NOREPLACE control word.

The standard command for adding data to the HISTDB data base is as follows:

```
ADD HSTRY-SHARE-FILE FROM TECUDB
```

The second major function of HISTUTIL is to store (or archive) data from HISTDB to tape (particularly data that is older than a certain date). The user runs HISTUTIL and enters the command STORE. The program presents the user with a list of the current data set pairs (with their date ranges) which he/she then chooses. The program generates a job stream for storing the appropriate files. These files are first moved into a holding area (group HOLD) with a copy of CNTL-MSTR data set to be able to quickly identify the contents of the data set pairs being stored.

The data set pairs are then archived to tape. In a similar manner, HISTUTIL contains a control verb option that allows data set pairs to be reloaded into HISTDB from the archives. This is accomplished by restoring an archive tape into group HOLD and executing HISTUTIL with the following command:

```
ADD HSTRY-SHARE-FILE FROM HOLD
```

Two options that were added to HISTUTIL include displaying the contents of the current HISTDB database and initializing the CNTL-MSTR data set.

Procedure Hdbget

In order to minimize converting all the application software to the new HISTDB design, it was imperative that the new interface procedure would be as transparent as possible in order to minimize coding changes. We started with the syntax to DBGET and were, in the end, forced to add two variables to the parameter list for reasons discussed below. Our new HDBGET supports the following syntax and modes:

```
         A      A     I      A      A      A        A
hdbget(pribase, dset, mode, status, list, buffer, argument,

         I         A
       secbase, secstatus);
```

mode 1: re-read; like normal IMAGE

mode 2: serial read, with data-set switch capability

mode 3: backward serial read, with data set switch capability

mode 5: chained read, based on secbase and secstatus relative record numbers. With data set switch capability

mode 6: backward chained read, based on secbase and secstatus relative record numbers. Actual directed dbget with data-set switch capability

Since this procedure was to reside in an SL, all variables used by the procedure were locally defined (Q-relative). That is, it is not possible to save information on previous calls within the procedure itself. It was seen that two variables had to be added to the procedure call. Variable SECBASE performs the same function as PRIBASE, i.e., to identify the previously opened secondary database (HISTDB).

Variable SECSTATUS is a 15 word array. The first ten words are used as the status array for the IMAGE calls made within HDBGET. The last five words contain the following information:

word 11: database id of current database
12: data set number of current data set
13: unused
14-15: record number of current record

Let's take a look at how HDBGET processes some of the specific modes: Mode 2, or forward serial mode, assumes we start from the oldest records (the beginning of the data base) to the newer records. Therefore this mode starts by accessing the HISTDB database. If the database doesn't exist, it immediately goes on to database TECUDB. HDBGET starts with the oldest data set pair in HISTDB (as determined by CNTL-MSTR). It is accessed in forward serial sequence and the data is returned to the user. If HDBGET reaches the end of data set, the next oldest detail data set is accessed. When no more data sets are to be found in HISTDB, HDBGET continues with the appropriate detail data set in TECUDB until the end of data set is reached. At this time, the user is given an "end of file" error message.

The more difficult transaction types to emulate were mode 6 and mode 5. Mode 6, for example, is the backward chained read. Here we start with the latest information in the chain and go backwards in time. Processing begins with the TECUDB database. A DBFIND is performed to the detail data set, HSTRY-SHARE-FILE. The status array returned contains the address of the last entry in the chain. DBGET, mode 4 is executed using this address, retrieving the information for the user. SECSTATUS words 11-14, are updated to contain the next address to be retrieved. Each subsequent call to HDBGET uses the information in SECSTATUS to do the DBGET, mode 4, read. When an end of chain error is encountered, HDBGET determines which data set pair in HISTDB is to be used next (the next most current). A DBFIND followed by a DBGET is executed (as above) and the information is updated in the SECSTATUS array. When all data is exhausted, an "end of chain" error is returned to the user.

To initialize the item list (for successive accesses via "*;"), HDBGET is called when the HISTDB database is still closed (i.e., when SECBASE is still zero). This will force HDBGET to open the HISTDB database and properly initialize all of its data set lists. If the user wants/needs to change the default item list in mid-program, then the HISTDB database must first be completely closed (DBCLOSE, mode=1) and SECBASE set to zero.

The return condition codes have been slightly altered to logically reflect the functionality of the HDBGET call. For example, it is assumed that a call to HDBGET will succeed and return data (more correctly, it is assumed that the key exists). Therefore, an invalid key will return a condition code of 14 (on a mode=3 access), which is a beginning of chain. It is as if we had gotten to the beginning of the chain and had not found any data. In such a case, portions of the DBEXPLAIN message may not be accurate. Specifically, the data set name and the mode may display erroneous results. This is due to the various other database accesses that HDBGET performs on the HISTDB database. The only item that is really examined is the condition code. Further information can be gleaned (if one really wants to) by examining the SECSTATUS array.

Implementation of Solution

The implementation of the HISTDB database into an operational reality required careful and meticulous planning. The following steps describe the overall process that lead to the successful implementation the reorganization of the HSTRY-SHARE-FILE data set. Refer to the accompanying chart below.

[A] Test the process of converting the current TECUDB structure to the new TECUDB structure. Use the procedure described below.

[B] Design and build the HISTDB and place a working copy on the secondary computer system.

[C] Store a copy of SUB-SHARE-FILE and HSTRY-SHARE-FILE onto the secondary system. Using a contributed library program, SELCOPY, begin extracting historical data (sorted by date) from HSTRY-SHARE-FILE and store into data base HISTDB as a number of data set pairs.

[D] Identify all source code that accesses the HSTRY-SHARE-FILE with a view towards identifying the types of source code changes that need to be made with the new structure.

[E] Write and debug the procedure that accesses the new HSTRY-SHARE-FILE and the HISTDB database (procedure HDBGET). Once debugged, install into production system. (It should function in a pass-through mode since the HISTDB database won't exist yet.)

[F] Make any changes to existing programs (as identified in [D]) that access HSTRY-SHARE-FILE to also access the new HISTDB database. Use the procedure(s) written in [E].

[G] Define and setup, in detail, the procedure for switching the HSTRY-SHARE-FILE from TECUDB to the new database (on a history set interval changeover). This will result in a utility that will perform or invoke all the functionality automatically (HISTUTIL).

[H] Perform full system test of the online system and other programs that access the HSTRY-SHARE-FILE.

[I] When ready to make the changeover to the new database structure, store the latest copy of SUB-SHARE-FILE and HSTRY-SHARE-FILE from the primary system onto the secondary system. If there are space constraints, store the historical data temporarily to tape.

[J] On the secondary system, use SELCOPY to extract the remaining data from HSTRY-SHARE-FILE that have not yet been extracted, sort the data by date, and write it to HISTDB. Delete the SUB-SHARE-FILE and HSTRY-SHARE-FILE from the secondary system.

[K] Convert the production TECUDB database to the new structure

[L] Move HISTDB to the primary system.

```
               [A] ----------\
                              \
                               \
                                \
[B] ---------- [C] ------        \
   \                      \       \
    \-------\              -------\\
     \       \                     \\
      \ [D]---[E] -- [F] --------- [H]--[I] -- [J] -------[L]
       \                           /        \             /
        \                         /          \           /
         \                       /            \         /
          [G] ------------------/              -- [K] --
```

Prior to Conversion Day

(1) Build a new database using TECUDB's new structure.

```
                   SUB-SHARE-FILE  SUB-SHARE-FILE-A
     ------    -----      ------   ------        ------
      \A /     \B /        \  /     \  /          \D /
       -- 01    -- 02       -- 04    -- 05         -- 07
       \        /                    /            /   \
        \      /                    /           /      \
         v    v                    v           v        v
                03                  06            08      09
------    -----               -----          -----    -----
|root|    | C |               |   |          | E |    | F |
------    -----               -----          -----    -----
      Other data sets    HSTRY-SHARE-FILE    Other data sets
```

(2) Store the database root file and data sets SUB-SHARE-FILE (04), SUB-SHARE-FILE-A (05), and HSTRY-SHARE-FILE (06) to tape. Remember, that these data sets are empty.

<u>Conversion Day</u>

(3) Save the manual master data by performing a data set unload of SUB-SHARE-FILE from the live TECUDB database using program DICTDBU. This takes about 30 minutes on a stand-alone series 70.

(4) Store SUB-SHARE-FILE and HSTRY-SHARE-FILE from the live TECUDB to tape and RESTORE onto the secondary system where the HISTDB will be updated to the present date.

```
                        SUB-SHARE-FILE
      ------    -----       ------            ------
       \A /      \B /        \  /              \D /
        -- 01    -- 02       -- 04              -- 06
        \         /           |               /     \
         \       /            |              /       \
          v     v             v             v         v
                 03              05             07      08
------     -----            -----          -----    -----
|root|     | C |            |   |          | E |    | F |
------     -----            -----          -----    -----
   Other data sets   HSTRY-SHARE-FILE      Other data sets
```

(5) Working with the live TECUDB database, physically rename data set 08 to 09, 07 to 08, and 06 to 07 (i.e., all the data sets after HSTRY-SHARE-FILE). Then delete data set 04 and 05.

```
      ------    -----                         ------
       \A /      \B /                          \D /
        -- 01    -- 02                          -- 07
        \         /                           /     \
         \       /                           /       \
          v     v                           v         v
                 03                             08      09
------    -----                           -----    -----
|root|    | C |                           | E |    | F |
------    -----                           -----    -----
```

(6) RESTORE the root file and data sets 04, 05, 06 that were created above in step (2). We now have a database in the new structure.

(7) Load the SUB-SHARE-FILE data set with the data that was previously unloaded above in step (3). Using DICTDBL, this takes approximately 70 minutes.

(8) Save the new database to tape.

(9) During the testing stage, STORE the database to tape (define the database with small capacities). This will give an additional verification that IMAGE accepts this newly built data base.

Problems Encountered with the Implementation

Our original HSTRY-SHARE-FILE data set contained approximately 4 million records. We couldn't delete all this data when we converted to the new structure since credit union members inquire on their recent transaction history. Since this data represented approximately three to four months' worth of transaction history, we decided to extract the data on monthly bounderies, creating data set pairs (see step [C] above). We knew that the data needed to be sorted by date and time (since we had to do serial reads and extracts through the detail data set to find the appropriate date range) and, therefore, used SELCOPY's sort feature to sort this data (SELCOPY uses HP's SORT intrinsics). Unfortunately, we had had no idea that many of the transactions that had been processed by the online system were being completed in less than 0.1 seconds. This was the time resolution that we maintained in the data. In other words, it frequently happened that two transactions for a particular member had identical time stamps. It wasn't until our first day live with the new system that this problem was discovered. A look at the transaction history of a member showed the transactions to be out of order (imagine shuffling your checkbook entries -- the total is correct but the intermediate results are wrong). It required some fast hacking to repair the historical data.

Related to the problem of breaking the detail data set data into manageable chunks was the problem of disc space. While we had a second series 70 with six 7933 disc drives, this system was being used to silhouette the primary, production system and therefore already had one full copy of the database on it. Much data set shuffling was done via magnetic tape and a 7978B tape drive.

The disc space problem was aggravated when we tried to build the HISTDB database. The schema specified 24 data set pairs for the HSTRY-SHARE-FILE information plus an additional 24 data set pairs for the loan information (to be implemented in the future). These data sets were defined with "reasonable" capacities. We ran DBSCHEMA and error-terminated with the infamous file system error 46 (out of disc space). It was then that our back of the envelope calculations showed that we needed at least 14 million sectors of disc space to build HISTDB (we had nothing near that much disc space available). The real problem was that we didn't really need all those data sets, just the definitions in the root file so that IMAGE would recognize them when present. Our solution was simple: we created the database root file using DBSCHEMA. Using DISKED5, we changed the flag in the root file from a "virgin" database to a functional database. IMAGE was happy.

We encountered one problem that almost spelled the end to the entire system. After we had created one of the data set pairs (due to an extraction with SELCOPY), we inserted the data into the framework of the new HISTDB database. We then tested the structure of the new database by trying to access various data with QUERY and comparing it against the original TECUDB database. Much to our shock, QUERY's FIND did not work! In the original database, FIND retrieved the correct data. Using HISTDB, FIND did not find anything. What was wrong? Careful examination of the structures of the two databases eventually revealed up the difference: the blocking factors of the respective data sets. While we had known that the capacities of the master data sets must be identical in order for the hashing algorithm to work, we had forgotten our IMAGE internals knowledge that the actual record address is a function of the block number and record within the block. We had to build our data sets with the correct blocking factors. Unfortunately, there is no utility that sets the blocking factor to that which we wanted. Nevertheless, a technique was found. We built the data set with a BLOCKMAX specification slightly larger than necessary and with a capacity of one too large. The credit union owns ADAGER; this program was then used to reduce the capacity by one and to reblock the data set. The new blocking factor then became identical to the original one in TECUDB.

We tested the integrity of the data after moving a data set pair from TECUDB to HISTDB. While the FIND command within QUERY worked (after fixing the above problem) the FORM SETS command did not. The display of the current number of entries was incorrect, garbage. A little more internals' knowledge reminded us that the current number of records is a calculated value, based on the capacity. Our HISTDB database detail data set had been built with a capacity of one million entries. We had transferred HSTRY-SHARE-FILE from TECUDB with 4+ million entries. While the output from QUERY's FORM SETS was incorrect, no processing was affected by it so long as no attempt was made to add any data.

During the testing of the HDBGET procedure, we uncovered one further interesting problem. The online application software, as part of its start up procedures, automatically accesses every data set needed in the database in order to initialize the IMAGE list parameter (for later use via the "*;" construct). With the addition of the HISTDB database, this wasn't properly extended. We had to add code to HDBGET to perform the same functionality within the procedure.

General Conclusions for Future Power Users

We once told one of the analysts here that, "Anyone who says he doesn't need to know the machine code for the machine he's working on is blowing smoke ... somewhere." Unfortunately we still believe this. In order to accomplish the HSTRY-SHARE-FILE split we needed an understanding of the application, IMAGE, and MPE. One of the most critical elements was a knowledge of the dependancies between the root file, the master file, and the detail data sets. The isolation of details from root and master files provided us with the first of two bases for the entire project and almost trashed the project three quarters of the way through. A late-night discovery of the dependance between the root file and the master data set, an unintentional Adager reblock, and the problems associated with recovering these misfortunes certainly convinced us that an exact understanding of IMAGE internals was critical.

The second basis for the HSTRY-SHARE-FILE split is that of 'SL isolation.' By that we simply mean isolating the actual data structure behind an SL routine. The critical point to this is that you are able to isolate the data structure and debug existing code BEFORE the actual structual changes are ever made. This gives you a staged implementation rather than simply going for broke; essentially the old CYA principle. Over the past four years TECU has used SL isolation in three (now four) major cases with dramatically successful results. One such implementation saved 24 hours of run time per quarter and 10 hours per month, eliminated the down time caused in both those periods, and was accomplished with only one minor problem.

The driving idea behind this exercise is that of a 'logical data structure.' Certainly we have heard much of this with regard to relational DBMSs but very little has been said in the non-relational field. Logically speaking, TECU had a single data structure, HSTRY-SHARE-FILE, which contained multiple periodic entities. We split the periodic entities into separate physical parts while retaining the logical continuity; thus providing physical flexibility while retaining logical support for the extant data structure.

Perhaps we've been living right (dubious, at best) or perhaps we got lucky (highly probably) but we did obtain some unexpected advantages from the conversion. First, we are now able to size the details arbitrarily (so long as we retain the auto master and detail blocking factors) and we may have an arbitrary number of details. Secondly, because of the nature of TECU's applications, the physical sequence of the data now corresponds to the chronological sequence of the records. This allows for such minor items as a chronological binary search of a detail to find all records inside a given period. This in turn allows us to quickly find and access records in a given period where we previouly would have had to search the entirety of the detail data set. Of course, sometimes one neither lives right nor gets lucky and then structural damage to one's database may occur. Prior to the HSTRY-SHARE-FILE split there was no way to recover from something as simple as a broken chain. Now we can simply wait until the pair with the break rolls out of the HISTDB. This is made even more impressive when one considers the fact that TECU has not done an unload/reload on their primary database since 1980!

Finally, there is one overriding consideration to each and every step of this procedure; we are doing this for intermediate and end users. Thus we must look at the external interfaces from the users' perspective; in this respect we failed in one sense and succeeded in another. In the first case we failed to realize that operations does not care about data set pairs but, rather, about HSTRY-SHARE-FILE date ranges. Therefore we used the pair IDs for moves into and out of HSTRY-SHARE-FILE as well as HISTDB. Additionally, we overlooked the loss of access to the structure through Query, Inform, Report, etc. In the latter case we succeeded rather admirably with our programmers in that they noticed little or no coding/performance differences between the old and new calls.

Traditional techniques offered no solution to our problems of managing a large data set. Due to the particular characteristics of the TECU database, we were able to design a technique utilizing safe privileged mode access that surmounted these obstacles. While these techniques cannot always be used in every environment, recognizing the existence of such techniques can be helpful for other companies designing sophisticated applications. This design became reality in August, 1987, when the Telephone Employees Credit Union went live with the new HISTDB database.

COMPUTER TRAINING:
HOW TO TRAIN THE COMPUTER PHOBIC

Christine Dale
Kaiser Foundation Health Plan of Colorado
2045 Franklin St.
Denver, CO 80205

Resistance and Fear

Why are people afraid of using computers? There are many reasons why people are afraid of using computers, ranging from not knowing how to type to the fear that they will be replaced. As technology moves in the direction towards automation, people are very much concerned about their own positions being taken over by computers.

You and I know, just by working in computers, that the previous statement is just not true! How many times have we discovered that by freeing the people we teach from the every day manual repetition, it leaves those people more time for creativity in their jobs? Over and over I have encountered the resistance to using computers caused by the notion that saving time will reduce the amount of work to be done. This time savings translates into direct labor savings from management's perspective, which has the support staff fearing for their jobs.

For example, I have seen a group of secretaries very upset because management had said that by using computers, the secretarial staff would be cut. Each one of the secretaries was concerned about losing her job. Overcoming their resistance to training was very difficult. If the directive to learn to use computers comes from management with the assurance that jobs will be enhanced, not eliminated, the employees will be excited, not fearful for their future.

Many people are afraid to admit that they don't know how to type. In this case, I like to give them a lot of encouragement to learn to type. There are good typing tutorials for PC's on the market. As time goes by, more and more software is function key or window driven and the need to be a skilled keyboard user is diminishing. But, learning is more difficult for the person who is trying to learn how to use computers and to figure out where the keys are at the same time.

Motivation

Working with the training staff, management must sell the benefits of training. Adult learners need to be assured of a positive learning experience. They need to be motivated through measures which include the emphasis for new opportunities. Once adults are convinced of the benefits of learning, the barriers of resistance will come tumbling down. This must be a concentrated effort between management and trainers.

One method I found very beneficial to people who are going to be learning about a system is to demonstrate it in a non-threatening environment, such as a departmental meeting. To "sell" the system and its benefits before training is very important. By presenting the system a step at a time, everyone's comfort zone is preserved.

In one company, I had to train 20 managers and 40 secretaries. They either had no or very little knowledge of computers. I was implementing a variety of programs on PC's as well as on the HP3000, so I had a challenge ahead of me. Initially, I met with each of the managers and their secretaries to get input from them as to what they'd like to use their computers for. After we selected the hardware and software, I held demonstrations for the groups, one for the managers and one for the secretaries. I showed them what they were getting and allowed them time to sit down at the machines and play. I also included some games and encouraged each of them to play the games. The company was a real estate firm, so I gave them copies of Real Estate Baron and Type Attack. They had a great time playing the games and even had some competition among themselves. It was fun to see the managers teasing the president as he lost his money. Later they had their real training sessions. From this initial fun session they felt comfortable with at least the basics of using the computer before they began getting down to business.

In another company, we had installed their HP3000 just before Christmas. At the annual department Christmas party we had everyone play the games in GAMES.SYS such as Blackjack and Othello. Again, everyone had fun while learning the basics.

While you are making the training fun, you cannot lose sight of the objectives. That objective is to teach them to become comfortable (and thereby, productive) using computers in their jobs.

Teaching Methods and Tools

1. Class Environment

The class environment is very important, especially for the beginner. If the student is uncomfortable, then his/her span of attention is short. Do you want everyone jumping up for coffee every 15 minutes? Do start with something good to eat, (it does help) but if the room is crowded and stuffy, everyone will either be asleep or leaving all the time.

The size of class depends on whether or not you want to be a neurotic at the end of the session. The best (and therefore, most expensive) way to teach is individually. This way you know that you have their attention and they also have yours. I have taught a number of high-level management people this way. You have to adjust your teaching style to their learning style. Some will allow you to teach as though you were in a classroom. Others have told me to teach them exactly what they want to know and not anything else! I call this speed-teaching, something akin to speed-reading. It is good for getting the general idea, but not good for details. These people will usually be back for more.

The more commonly used method for teaching is in small groups. Each student can have his/her own terminal or have two students share the terminal, but no more than two per terminal. The advantage to having each student having his/her own terminal is that they have much more time for hands-on training during class. But if two people are sharing the terminal, they can help each other along and reinforce the lessons. I have no preference for either way. I usually have no more than 4 terminals at one time, for up to 8 people. Any more than that and the class goes too slow.

2. Team Teaching

Whenever possible, have a user be a part of the training team. At the real estate firm, there were two of us who taught the classes. I taught the technical parts, and the other person from the user area taught the application parts. We blended well with each other and monitored the progress of the students by monitoring each other. She called herself the "dummy". For the first few times we taught, she'd make me explain in more simpler terms if I became too technical. She drew upon the group's experiences to relate back to what was being taught.

Be sure to use plain English and define **ALL** terms. One helpful hint is to label all the parts of the equipment and have a picture to give them, with, of course, all the parts labeled.

3. Start at the Very Beginning

   If the user is being trained on a terminal or PC attached to the HP3000, give them a tour of the computer room. Again, have labels on the equipment in the computer room. Once a user understands that they are a part of the big picture, they can understand what goes on in the "big black hole" called the computer room.

4. Teach One Thing at a Time

   Give the students outlines and quick reference guides at the beginning of class. While teaching them, refer to the outline and the quick reference guide often. Even though you may have a reference book to give them, DON'T teach from that book.

   Apply the K.I.S.S. (Keep It Simple, Stupid) method to teaching. Have the student perform only one function at a time making sure you clearly define the end result. A step-by-step approach is a must. If you have the students doing multiple tasks, you will only confuse them.

   Also, if there is more than one way to do something, only show them one way at first. Once they have mastered the first way, then show them the other ways or the shortcuts. For example, if you teach a product like HPDESK, there are many ways to move around the desk. **ONLY** show them how to move around the desk by using the function keys. Then, after they are used to the function keys, have them move around the desk by using the numbers.

5. HELP-HELP-HELP

   Most software comes with a "help" facility. As a part of class, emphasize the use of "help". Practice going into and out of the help screens so they can become comfortable with using the feature. This way, after class, they can become self-sufficient.

   Give the students names of people who work around them who also know the software. Many users like being resident experts as long as it doesn't start being the primary part of their job.

At Kaiser, we have given out a brochure to all users of PC's and Office Automation software describing the Technical Support Services that we provide. One service we have implemented is a "HELP LINE" phone number. This number, along with a system ID, is attached to everyone's equipment. The "HELP LINE" is always staffed during working hours and everyone can get their questions answered immediately. We document the questions and can tell if a person needs further training in specific areas (or if the initial training didn't take). Not only do we answer questions, but we provide the personal attention neophyte users often need.

After the Class -- FOLLOW UP

It is very hard to have an effective training program without providing follow up on the training. I tell the students to go back to their offices and take time to go through the class exercises again and just practice. I also tell managers that they **MUST** allow their employees time to practice before giving them real work to do.

To get people started using Electronic Mail, I wrote messages to people to get them to write messages back to me. We scheduled training classes through Electronic Mail. I also encouraged everyone to write messages to each other. One manager was complaining he didn't get any Electronic Mail. When the other managers heard this, they started sending him messages just so he wouldn't feel left out. Try to find a champion among the users and this person will do more for spreading enthusiasm for using the system than all the Data Processing people combined.

A very good source for additional training after the class is the on-line tutorials that come with most software. I'm impressed by the quality of some on-line tutorials I've seen. Several years ago, there was no such thing. It's wonderful that software companies are realizing people need more than reference books! I particularly like the "learndesk" feature of HPDESK. If time is at a premium, or if you can't hold a formal class, the on-line tutorials can be substituted for classroom training.

Conclusion

I have been training people how to use computers for many years and I still learn new techniques daily. I find that making learning fun and non-threatening goes a long way in making a training program a success. When you see a person who has never used a computer become one of the most excited computer users around, it is worth all the effort!

A Guide To Breaching HP 3000 Security

Phil Curry
Carter, Schaefer & Company
Houston, Texas

Introduction

People have been intrigued by secret codes and security for years. Take for example the decoder pins given away as premiums during the 1930's. A kid with his Captain Midnight or Little Orphan Annie decoder pin could send a "secure" secret message to his or her pal and keep would be "spies" from deciphering the message.

In the 1980's people are no different. With a relatively cheap computer or terminal one can try their hand at breaking security on a computer system. Most of the time it's not that the computer system has anything of any real value, it's just the fun of breaking the security and finding the codes to get into the computer.

What I'm going to reveal about you Hewlett Packard 3000 computer systems will make most system managers weak and wonder why in the world am I telling everyone this. The answer is simple. To prevent you from having a breach of security. Chances are you have already had a breach and never knew it. I'm not necessarily talking about someone logging on and moving money into their checking account or changing their hourly pay rate. I'm talking about someone having access to your system and potentially having this capability without your even knowing it. Someone may know they can have access to your computer or system account anytime they wish and are waiting for the right time to use it, like one week after they are layed off work or fired.

What this paper will do is give you an idea of how a Hacker thinks and can gain access to your system. It speaks from the view of the Hacker and tells how you as the system manager can prevent the breach of security. In some cases, one can't. Also note that I'm not giving away all my secrets. Like a magician I must keep the mystique of knowing how to get into a system to myself. Their are ways that if told could do more harm than good.

There is a misconception of what a Hacker really is. A Hacker is not one of these kids seen in movies like War Games that attempts to call computers and break into them. The name they give themselves is "Phreaks". A Hacker is a one who really gets down to bits and bytes with the computer.

Real Hackers.....

1. patch the object code. It's much faster than the edit, compile, and prep process.

2. use the ASSEMBLE statement in SPL.

3. can perform Binary, Octal, Hex, and Decimal conversion in their head.

4. know at least 4 languages (at least 5 if you count Basic).

I'll now remove my system manager hat and put on my Hackers hat and reveal how you can potentially break security on an HP 3000. Again, note that this is done to tell you how to prevent a breach and NOT to let people know how to get into your system.

Chapter I - Gaining Access To An HP 3000

The first thing one needs to do to breach security on an HP 3000 is to get access to one. If you have access to a personal computer and an autodial modem, you can do this.

In the movie "War Games", a high school student uses his home computer (an archaic Imsai 8080) to dial successive numbers in a telephone exchange looking for modems. You can do this too! There are many "War Games Dialers" available on computer bulletin board systems. If you can't find one, their easy enough to write. Anyone can go to Toy's Are Us and buy a Commodore 64 computer and a modem and do exactly what was done in the movie to find modem numbers to computers. Looking at the figures in the March 30, 1987 issue of Infoworld magazine, IBM has sold over 7 million personal computers. In early 1988, IBM announced they have sold over 1 million PS/2s, which raises the number to over 8 million. This figure doesn't even count compatibles, such as Compaq. Considering the cost of these systems, imagine how many less expensive Apple and Commodore computers are in the marketplace. Any one of these computers with a modem can access an HP 3000.

Some system managers are crafty and will buy modems that will not run at lower speeds to keep cheap 300 and even 1200 baud modems from connecting to their system. Even harder to overcome is the use of dial-back modems. Whenever someone wants access to the computer they must enter their name or access code. Then the line is disconnected and the computer calls the phone number it has stored for the password and connects to the terminal or computer. This keeps one from hacking into the system since even though we know a password, the computer will hang us up and dial the number it has associated with the logon.

Chapter II - You Have A Colon Prompt, Now What?

Ok, you now have access to an HP 3000. Now you need to log on to it. You need to know a user and account that is on the computer system. Hear are some user.account combinations to try:

MANAGER.SYS	Password: HPONLY
OPERATOR.SYS	
MGR.TELESUP	Password: HPONLY
FIELD.SUPPORT	Password: HPONLY
MGR.MAINLIB	
MGR.CSL3000	
MANAGER.TECH	
MGR.INTX2	
MGR.SCRUG	
MGR.BWRUG	
MGR.DETROIT	
MGR.GAMES	

If you already have a valid account on the system you're way ahead of the game.

To keep you from finding out passwords, good system managers will never use default passwords, HP's or third party vendor's. Passwords may be hard to guess since the best passwords are meaningless, like license plates. Combinations of letters and numbers not the name of the user's child or dog. Also three passwords could be needed; account, user and group.

Chapter III - Your On The Computer, Now What?

If you didn't log on as MANAGER.SYS or FIELD.SUPPORT you will have limited access to the computer. You want as much access as you can get. You can do the following things to find other person's passwords.

I. The Fake Restore

The MPE Store format is documented in the Systems Managers Reference Manual. The STORE command will disallow one from restoring files from one account into another. However, what you can do is write a program to read the STORE tape and load the files into your account. Program files are no good. The good ones will need to be run in an account with PM capability anyway. Good files are CATALOG.PUB.SYS and any file that looks like a stream file (files in groups named JOB). Don't let lockwords scare you, they are just part of the data on the tape. If you have a SYSDUMP tape, that's even better since the system directory is on the front of the tape. Get a source code listing of STAN from the Contributed Library, it can help immensely.

After you've written your program, tell the operator that you accidently purged a file and need to restore it from the last backup tape. Tell him you'll do the RESTORE command when he mounts the tape. Then run your program and the unsuspecting operator will mount the tape and reply to it.

II. Intercept Terminal I/O

Look for the program called PEEKABOO. This contributed program allows one to monitor all terminal activity to a device. For example, you can run PEEKABOO on the console, device 20, or on the system manager's port.

III. The Fake Prompt Trick

Write a program to emulate the MPE command interpreter. The steps are as follows:

1. Open file on another terminal
2. Read device.
3. If input is not a HELLO command send appropriate error message and go to step 2.
4. Scan for missing passwords (user didn't enter password during logon)
5. Prompt for password(s)

6. Give fake MPE error message (such as account out of time, cannot open UDC catalog, etc.)
7. Close the terminal.
8. Write the passwords to a file to read later.

You can get as elaborate as you wish. You could even fake a logon and parse the users commands and any of them that are programatic (LISTF, REPORT, SHOWJOB, etc.).

IV. The Trojan Horse

Write a GREAT program that is a game or utility that alot of people will run. When the unsuspecting system manager puts it in an account where everyone can run it, you can get their passwords. The steps are as follows:

1. Do a programmatic LISTUSER to a RELEASED file in your account. Be sure the file equation for the LISTUSER accesses the file with append access. If the command fails, then the password is of little value since it doesn't even give you account manager capability, proceed to step 3.
2. Do a programmatic LISTACCT to the RELEASED file in your account. The command won't fail since the program got this far. Also, if the user has SM capability, you just got ALL account passwords.
3. Begin game or utility program.

V. The Terminal Emulator Trojan Horse

A good terminal emulator is hard to write, but you can write a simple one. Some are available with source code, like KERMIT, from local bulletin boards or personal computer user groups. Once you have the program, some simple modifications will help you get other user's passwords. What you do is give a copy of the program to someone who has a password to the system that you desire. Remember, the program you gave them sees EVERYTHING the user types. So, what you do is write code to look for the user's logon and the MPE welcome. You will need to write another program on the HP 3000 and release it. The HP 3000 program will perform a file transfer by doing the following:

1. Reads the terminal until some end of file indication.
2. Appends the HELLO command and accompanying passwords to a RELEASED file in your account.

On the PC end, your emulator will need to do the following:

1. Look for HELLO command
2. Look for MPE welcome with revision of the operating system and terminal read.
3. Issue command to MPE (remember, you have control) to run your program described above.
4. Transfer logon information to the HP 3000.
5. Begin normal terminal emulation.

VI. Use Unknown Bugs Or Features Of Programs

The program SLS is a widely used program from the Contributed Library. It allows one to stream jobs while interacting with the user for information before doing the STREAM. The program writes the stream file to a file named JXXXA then calls the COMMAND intrinsic with the command STREAM JXXXA then purges the file. The original source does not disallow file equations to the file JXXXA so by doing the following you can see the passwords in the stream file:

1. :FILE JXXXA=$STDLIST
2. Now do the normal steps to do the SLS job
   Example: :SLS BACKUP

The entire JOB stream, including passwords will be displayed on your terminal.

If the system manager has locked you out of using the compilers or system utilities such as FCOPY, PREP or RELEASE, find access to a utility that allows programmatic calling of these commands or allows running programs inside them. Good examples are EDITOR, QUAD, and SPOOK. Try putting a colon in front of the command if just typing the command does not work. QUAD gives you access to the compilers as well as MPE commands so it is a good choice.

There are some other undocumented features on the HP 3000 that are interesting.

1. In QUERY, after doing a FIND type the command NUMBERS. This will display the relative record numbers of the records you just found.

2. In EDITOR when saving a file and asked if you wish to "PURGE OLD?" you can enter "OK" in place of "Y".

3. Most HP utilities will let users with PM capability to type DEBUG at the prompt and will drop them into PRIV MODE DEBUG.

4. There is a program in PUB.SYS called IOCDPNO that is disguised as a card punch driver left there for SE's to be able to call ATTACHIO directly. If you enter HELP at the prompt, you can cause a system failure.

5. There used to be a back door in MPE. You could type =DEBUGG (yes, two g's) at the console at drop right into PRIV MODE DEBUG.

If programmers at Hewlett/Packard are allowed to leave undocumented commands in the software, there just may be some back door waiting to be found.

VII. Look For Files In PUB.SYS And Other Accounts You Can Run

Most system managers have no idea what is in PUB.SYS. Unless otherwise altered, most programs in this group can be run by everyone. Sometimes, the lazy system manager will put utilities in this group so several people can run 1 copy of the program and hope user ignorance will keep others from running the program. Some useful programs to look for include:

1. PEEKABOO
2. ALLOWME
3. STAN
4. GOD
5. JSPOOK

Also, look for any released program files in PUB.SYS. You can write a program with PRIV MODE capability and copy it on top of the RELEASEd file.

1. Write PRIV MODE program and compile it.
2. PREP without PM capability (you need PM to PREP with PM)
3. Patch the object code to give program PM capability. You can do this with DECOMP from the Contributed Library using the REPREP command.
4. FCOPY released program to your account.
5. Copy your program on top of RELEASED program file.
6. Run program.
7. FCOPY original file back (destroy the evidence).

Epilog - What Have We Learned?

Now I'll take off my Hackers hat and put my system managers hat back on and re-cap what we learned.

1. Be careful who is dialing into your computer system. If really necessary, use dial-back modems to verify the user.

2. Put passwords on ALL accounts no matter how little access they have. Once a person is on, they could access things you never thought of. And NEVER leave the passwords on the accounts SYS and SUPPORT at their default.

3. Do not mount SYSDUMP or STORE tapes for users to read. If a user lost a file, RESTORE it for him.

4. Don't keep PEEKABOO lying around.

5. When logging on, suspect that you have a fake colon prompt when logging onto the machine. Type :EOF: to close the device, then press RETURN to get another prompt.

6. Never take programs written by users and place them in accounts where all users can run them unless you have source code. Then, recompile the program and put THAT object code where everyone can run it. Sometimes this isn't possible, like Contributed Library programs. BEWARE of persons bearing gifts.

7. Limit access to the MPE command interpreter to users. As you can see, this is where all the trouble starts. Get a menu system and allow users to only run what is needed. You can pass it off as a "User friendly interface" and get away with it.

8. Look at all files in accounts everyone has access to. Make sure none of them are RELEASED. Make the access to the file minimal. For example, utilities like SLPATCH should have its access X:CR to allow access only by the creator, not everyone.

9. Don't place anything in PUB.SYS that didn't come from HP. Utilities should go in a different group and maybe a different account if no PRIV MODE is needed.

10. Security is a never ending battle.

Appendix

Listing Of A "War Games Dialer"

```
10 '*******  AUTOMATIC SEQUENCE DIALER (a la "Wargames") **********************
20 'Written for the IBM PC and HAYES SMARTMODEM by Steve Klein 9/25/83
30 'Modified (Steve wouldn't recognize it anymore) with enhancements (starting
40 'number, printer on/off option, abort/hang up) by John Siers 12/28/83
45 '************************************************************************
50 'This program will dial numbers in sequence looking for computer carrier
60 'signals. When carrier is found, phone # is listed to printer and/or screen.
75 '************************************************************************
100 CLEAR ,,2000:XY=2
110 KEY OFF:COLOR 0,7:CLS:LOCATE 10,25:PRINT "Wargames Dialer Program"
120 LOCATE 12,26:PRINT "Written by Steve Klein":LOCATE 14,26:PRINT "Modified by
John Siers"
125 FOR I=1 TO 5000:NEXT
130 A$="":AB$="":CLS:PRINT " PLEASE ENTER PREFIX DIGITS (IF ANY), THE AREA CODE
(IF ANY), AND THE":PRINT "FIRST THREE NUMBERS [hyphens may be used to separate
e.g.9-1-nnn-nnn]: ":INPUT "~~~>",A$
140 INPUT "START DIALING AT # (LAST 4 DIGITS)";SN:IF SN>9999 OR SN<0 THEN 140
ELSE N1=INT(SN/1000):N2=INT((SN-(N1*1000))/100):
N3=INT((SN-(N1*1000+N2*100))/10):N4=SN-(N1*1000+N2*100+N3*10)
160 PRINT "LIST COMPUTER CONNECTIONS TO <S>CREEN ONLY OR <P>RINTER AND SCREEN?"
170 PRINTON$=INKEY$:IF PRINTON$<>"S" AND PRINTON$<>"s" AND PRINTON$<>"P" AND
PRINTON$<>"p" THEN 170
200 '*** Begin dialing sequence ***
205 CLS
210 FOR E=N1 TO 9:FOR B=N2 TO 9:FOR C=N3 TO 9:FOR D=N4 TO 9:N1=0:N2=0:N3=0:N4=0
220 OPEN "COM1:300,n,8,1,CS,DS" AS #1:R=32:PRINT #1,"ATDT"A$;E;B;C;D
225 DIAL=E*1000+B*100+C*10+D:DL$=STR$(DIAL):IF LEN(DL$)=2 THEN DN$="000"+
RIGHT$(DL$,1) ELSE IF LEN(DL$)=3 THEN DN$="-00"+RIGHT$(DL$,2) ELSE
IF LEN(DL$)=4 THEN DN$="-0"+RIGHT$(DL$,3) ELSE DN$="-"+RIGHT$(DL$,4)
230 GOSUB 500:LOCATE 25,1:PRINT "DIALING ";A$;DN$;:LOCATE 25,35:
PRINT "TIME LEFT";R;"SECONDS: [A]=ABORT [H]=HANG UP ";
240 '*** Check for input to comm buffer (carrier) ***
250 A=LOC(1):IF A>(20+LEN(A$)) THEN 280
260 IF R>0 THEN 230
270 CLOSE:FOR I=1 TO 3000:NEXT:NEXT D,C,B,E
275 LOCATE 25,1:PRINT STRING$(79,32);:LOCATE 25,1:
PRINT "END OF DIALING SEQUENCE:";:INPUT " PRESS ENTER TO CONTINUE ---->";XX$:
GOTO 130
278 '*** Found One!!! ***
280 SOUND 150,5:XY=XY+1:LOCATE XY,1:PRINT A$;DN$:IF PRINTON$="P" OR
PRINTON$="p" THEN LPRINT A$;DN$
290 GOTO 270
500 '*** Countdown time and check for abort/hang up ***
510 LET R=R-1:FOR I=1 TO 1050:NEXT:AB$=INKEY$:IF AB$="A" OR AB$="a" THEN 520
ELSE IF AB$="H" OR AB$="h" THEN 530 ELSE RETURN
520 PRINT #1,"ATH":CLOSE:LOCATE 25,1:PRINT STRING$(79,32);:LOCATE 25,1:
INPUT "DIALING ABORTED: PRESS ENTER TO CONTINUE ---->";XX$:GOTO 130
530 PRINT #1,"ATH":R=0:RETURN
1000 '**********************************************************************
1010 ' Another helpful program from Steve Klein
1020 ' With enhancements by John Siers (who found the original on the
1030 ' Lehigh Valley BBS, Allentown PA -- 12/83)
```

Training a New Operator - Where Do You Begin?

Flo Barley
Pekin Memorial Hospital
Court & Fourteenth Streets
Pekin, Illinois 61554-5098

Introduction:

You're getting a new operator - whether replacing one that left or adding a new person - either way - they need to be trained. Where do you start? You've shown them the computer center, so now they know where it is - they're just wondering what to do with it. There's books and manuals that can bury them for months, but is that the place to start? There's operator classes at the nearest training center, but what else do they need to know about your operations and applications that they can't get anywhere but from you? How much can you throw at them and expect them to absorb?

If you live in an area where there aren't many HP shops, you could have a problem with finding anyone with HP experience - let alone anyone with operator experience on any machine. This leaves you with quite a job ahead of you. You have a capable body in front of you, eager to learn and ready to tackle anything that comes his or her way. You're wondering how to get them pointed in the right direction without scaring a resignation out of them on the first day.

I will attempt to set up a step by step guideline that any shop - no matter how small or how complex - can use, adding as they see fit to suit their particular needs. Using basic steps, a trainer can set up an outline to assure himself that he's covering all the items needed. This can be expanded as you go to include not only hardware, but also software. This should give your new operator a strong training base and you enough confidence that your department will continue to operate to it's fullest capacity.

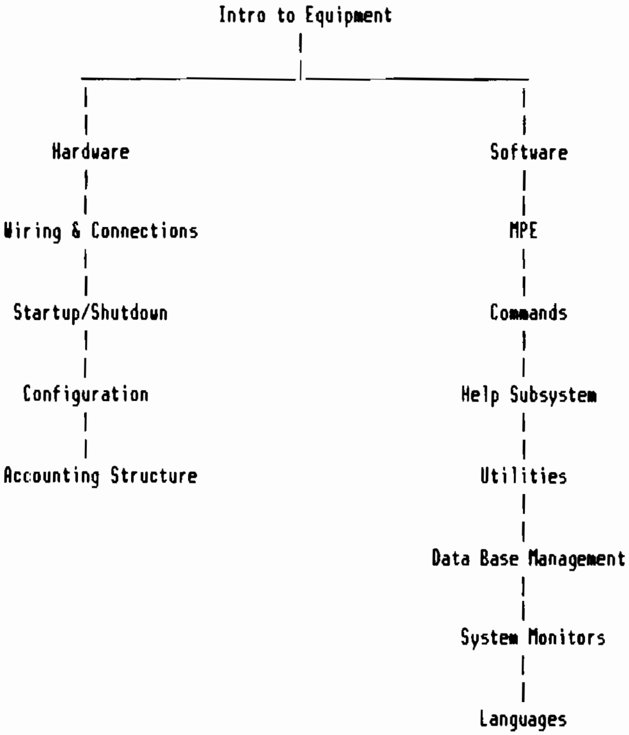
Operator Training can be and is a never-ending task. Each environment is different and must be approached from a different angle. You not only have your HP system with both hardware and software, but you also have various software packages ranging from financial system reporting to hardware monitoring tools.

There are well structured training classes offered at various locations in the U.S. by Hewlett Packard and other training facilities, along with the software training for the packages you have installed, but if it's not cost efficient for your organization to send your operators or the timing factor is not right for them to be gone, you need some type of organized training plan to get them started. Training must be an on-going process since modern technology never stops in this day and age and just keeping up with what's new in the world of computers is a never-ending battle. Once you get a basic outline of a training plan that will fit into your organization, you must keep it updated and each time it's used for new operator training, the operator's will be at an even level once they have finished.

All of my operators have been trained in-house and this type of training seems to work somewhat successfully for me. I say somewhat, because there still is the problem with keeping them updated on current issues. It is difficult to have initial training sessions with all the operators at once if you have a a small data center with varied shifts, but it can be done at a much slower pace with regularly scheduled sessions during non-peak hours. A slower pace may not be quite as effective for initial training if you have an immediate need for an operator, but one-on-ones with the operator for basics is a workable method to start, then having a more intensive training session for approximately 5 to 6 hours a month with all the operators together can cover a lot of ground and get them on a more consistent level with each other. I have tried this method since my training was catch-as-catch-can to start and have gotten great results from my operators who have responded to these sessions positively.

No matter what type of background your operator has come from, there will be some type of initial training needed. If they are from an HP environment, there would be minimal training as far as hardware except for the differences in the operating system and types of equipment you use as opposed to what they were used to. Someone not familiar with HP or any computer system would have to have more extensive training as far as hardware. I have not had the pleasure to train anyone who has come from an HP environment, so my training has basically been from scratch. Training can feel like an endless job and can be so overwhelming to the trainer let alone what it seems like to the trainee. It has to be looked at as part of the everyday routine and learning so much each day doesn't make it look quite so bad.

It was difficult for me to break down the training into a set pattern for everyone to use because the software will differ with each organization, but setting up a flowchart to begin with will allow you to branch off in whatever direction necessary to completely cover all the areas needed.



HARDWARE

The first step is to get them acquainted with the equipment. Let them know all the different types of equipment you have throughout the business, and then get them accustomed to the equipment they will be working with. Start with what it is, what it does and how it's used. Also include what type of maintenance it needs, such as ribbons, cleaning, paper, etc., and who to call or when to call HP for service and if they have that privilege.

Explain what the system hardware consists of including:

1. Stack Architecture
2. Virtual Memory
3. Disc Caching
4. MPE
5. Wiring and Connections
6. Terminals and Console
7. Disc Drives & Tape Drives
8. Modems and Multiplexors

Key terms to understand:

1. MPE
2. CPU
3. GIC
4. SIB
5. ALU
6. AIB
7. RAM
8. ROM
9. Caching
10. RS232
11. RS422
12. Input/Output

Understanding of Configuration would include:

1. Term Types
2. LDEV #'s
3. Device Classes
4. Memory Allocation
5. Virtual Memory
6. System Tables
7. Volume Table
8. Directory Info
9. Programming Info
10. System Logging
11. Scheduling Info

HARDWARE (cont'd)

Understanding of the Accounting Structure is necessary:

1. Groups
2. Users
3. Accounts
4. Capabilities
5. Homegroup
6. Lockwords/Passwords

SOFTWARE

What is MPE and what does it offer:

1. Configurator
2. Power fail/Auto restart
3. Backup/Restore Facility
4. Logging Facility
5. Utility Intrinsics
6. Loader
7. Segmenter
8. Process Manager
9. Job/Session Scheduler
10. Spooling Facility
11. Tape Labels Facility
12. Serial Disc Interface
13. Private Volumes Facility
14. Disc Space Manager
15. Virtual Memory Manager
16. Input/Output System
17. File Management System
18. Command Interpreter
19. Application Message Facility
20. System Console Manager
21. Initiator
22. Support for Disc Caching
23. Batch Processing

Show them how to use the help subsystem, how to use the MPE Quick Reference Guide and what HP manuals to read. Have them know where to look and find pertinent information regarding the MPE Message System, the configuration guidelines for setting up system tables, and the different subsystem utilities.

MPE Commands most commonly used:

1. ABORT
2. ABORTIO
3. ALLOCATE
4. ALLOW
5. ALTACCT
6. ALTGROUP
7. ALTJOB
8. ALTSPOOLFILE
9. ALTUSER
10. BREAKJOB
11. BUILD
12. BYE
13. CACHECONTROL
14. CONSOLE
15. DEALLOCATE
16. DELETESPOOLFILE
17. DISALLOW
18. DOWN
19. EDITOR
20. FCOPY
21. HEADOFF
22. HEADON
23. HELLO
24. HELP
25. JOB
26. JOBFENCE
27. JOBPRI
28. LIMIT
29. LISTF
30. OUTFENCE
31. PURGE
32. RECALL
33. REDO
34. RENAME
35. REPORT
36. REPLY
37. RESETACCT
38. RESTORE
39. RESUME
40. RESUMEJOB
41. RESUMELOG
42. RESUMESPOOL
43. RUN
44. SAVE
45. SETCATALOG
46. SETJCW
47. SHOWALLOW
48. SHOWCACHE
49. SHOWCATALOG
50. SHOWDEV
51. SHOWJCW
52. SHOWJOB
53. SHOWLOG
54. SHOWLOGSTATUS
55. SHOWME
56. SHOWOUT
57. SHOWQ
58. SHOWTIME
59. SPEED
60. STARTCACHE
61. STARTSPOOL
62. STOPCACHE
63. STOPSPOOL
64. STORE
65. STREAM
66. STREAMS
67. SUSPENDSPOOL
68. SWITCHLOG
69. SYSDUMP
70. TELL
71. TELLOP
72. TUNE
73. VINIT
74. WARN
75. WELCOME

Utility programs, standard with each system and their use:

1. Edit/3000
2. FCopy/3000
3. Sort-Merge/3000

What types of Data Base Management used:

1. Turboimage/3000
2. Query/3000
3. Ksam/3000
4. VPlus/3000
5. Adager

System Monitors:

1. OPT/3000
2. APS/3000
3. SOO5E
4. Tuner
5. Surveyer
6. TERMDSM

Languages used:

1. COBOL II
2. RPG
3. FORTRAN
4. BASIC
5. PASCAL
6. SPL
7. C
8. TRANSACT/3000

They will need to know how to monitor the system and manage jobs, sessions and spoolfiles. This would include the use of SPOOK, UDC's, STREAM, file types and equations.

It will be necessary for them to know how to start and shutdown the system and what processes are necessary in your environment. This area would also include what to do when there is a system failure or halt, ie: is there a downtime log or failure/halt log to complete, who to call, what steps are needed before startup such as string dumps, memory dumps, and what type of recovery is needed. They should now the difference between Warmstart, Coolstart and Coldload and the importance of each. What is a Reload and when and why should it be done? How should it be done? What can be done to save data when a system is down and there is no current backup available? What is a sysdump and why is it done? What types of backup are there?

These are all good topics to cover and periodically these types of questions can be put together in a test format and given to the operators. I have several of these and give them to my new operators to use as a worksheet or a learning tool. They have the manuals to use for answers and it forces them to find the answers and know where to look for them. It can also show you weak areas that may need to be covered in future training sessions. Some of the questions that I have for them are:

1. What are the four parameters that can be used with the ALTSPOOLFILE command?

2. What command would you use to log off only session 211?

3. What command would you use to abort job 21?

4. What is a UDC?

5. What are two ways that spoolfiles can be deleted (purged)?

6. How can you raise the priority of a JOB in the WAIT state?

7. The OUTFENCE is set at 14. How do you print spoolfile #0412?

8. How do you find the filenames for all files that begin with the letter S, that are in the SYS account and the PUB group?

9. Can the system's configuration be changed from a:
   SYSDUMP _____ YES _____ NO
   COOLSTART _____ YES _____ NO
   WARMSTART _____ YES _____ NO
   COLDSTART _____ YES _____ NO

10. Can you log on to a non-console terminal via HELLO OPERATOR.SYS?

11. What's the difference between a full SYSDUMP and a STORE @.@.@?

12. Explain what a file equation does and the parameters that can be used.

13. How do you correct the margins on a terminal if they are set wrong?

14. No reports have printed and in doing a SHOWOUT you notice that LP shows a report as 'ACTIVE' - what are four reasons the printer isn't printing?

15. The console is "beeping" - what should you look for?

SUMMARY

This is by far, not a complete list, but hopefully enough to get you started with your own in-house training, if necessary. By branching off at the key areas where you need to work in your own software and pertinent information to your organization, you should have a working outline for training.

All of this information can put fear into a new operator - but putting them at ease from the start and setting limitations as to what they can do on their own will help them gain a comfort level. They'll never know everything because I doubt that there are too many people who do fully understand what the system does and what it's capable of doing, but your operators will know what is necessary to keep your system up and running and what to do when it won't!

TITLE: Parity Pitfalls

AUTHOR: Karen Davis-Mackle

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 0062

Addressing the Problems of Program Documentation

Claire M. Perkins
Kaibab Industries
4602 E. Thomas Rd
Phoenix, AZ 85018

Documentation has long been a thorn in the side of many data processing departments. Tedious to prepare, it is often left until the last stage of system development and done sloppily, incompletely or not at all. Major maintenance problems occur when the only documentation for a system exists in some programmer's head -- and he is no longer around.

As programmers, we have all felt the lack of quality documentation at one time or another when we were expected to maintain a system with which we were totally unfamiliar. We probably did everything short of creating a voodoo doll of the programmer who left us this mess of spaghetti code with not so much as a data flow diagram for a roadmap. Unfortunately, someone somewhere out there was probably thinking much the same thing about the systems we left behind.

As programming managers, we have felt the lack of quality documentation each time we've watched minor maintenance requests evolve into major time consumers. We have sweated over deadlines and smoothed users' ruffled feathers as time ticked away while our staff struggled to find the right source code, the right compile job and all the relevant parts and pieces that make up the system. And we have sworn that we would put an end to the problem by creating the missing documentation, or updating and organizing what little documentation we had. But not this time, not this project, because we were already pressed for time.

As end-users, we have often thought we might just as well have asked for a new system when we asked for maintenance work from the DP department. We just couldn't understand why a simple request should take so long and cost so much. When the programmer in charge of the project suggested that we should allow even more time for the project so that documentation could be created, we were convinced that the DP department's motives were completely self-serving.

Everyone, it seems, has felt the effects of the problem, but like the weather: "Everyone complains, but no one does anything about it."

While the weather is something beyond our control, the documentation problem is not. Each of us; programmer, programming manager and end-user, has the power to improve the situation. Okay, so where do we start? Well, like all good analysts and programmers, we start with the basics: defining terms and defining the problem.

WHAT IS DOCUMENTATION ?

Documentation is a collection of documents. Getting out my good old Funk and Wagnalls I see that a document is "something written or printed that furnishes conclusive information or evidence." Conclusive means "putting an end to the question." I guess that means that those old binders full of paper which usually create more questions than they answer don't qualify as documentation.

There are many categories of documentation to be found in the typical DP shop. Some categories of documentation and the things they may contain are as follows:

> Policies and Procedures--used to define the guidelines under which the department should operate.
>
> Project Documentation--includes project plans, estimated time and cost for project completion, critical path diagrams, status memos and meeting minutes.
>
> System Documentation--includes high level HIPO charts, a narrative system overview, and a list of all the programs, job streams, data files and reports contained in the system.
>
> Program Documentation--includes a narrative description of the program, some kind of flowchart or other diagram illustrating logic flow, a maintenance history, and narrative descriptions of any complicated sections of code.
>
> Operations Documentation--includes instructions for streaming and/or monitoring jobs, instructions in case a job aborts, report distribution instructions, and a list of contact people for each of the systems maintained.
>
> User Documentation--includes instructions for data entry, pictures of screens used and reports generated by the system, and helpful hints for problem resolution.

Different DP shops will have different combinations of the kinds of documentation listed above. This is due, in part, to a different style of work and a different range of needs from shop to shop. It is also due to the fact that programming is a relatively high turnover field. As your programming staff changes and evolves, so does the prevelant style of documentation.

This trend seems to continue because documentation is often considered to be the sole responsibility of the programming department. No one has attempted to standardize the process, so the content, accuracy and format of the documentation is left to the discretion of the programmer.

WHOSE DOCUMENTATION IS IT, ANYWAY ?

Documentation belongs to everyone in the company. End-users, operators, managers and programmers all benefit when the documentation of DP systems and procedures is accurate, current and complete. Since most smaller companies cannot afford to staff a full-time technical writer, the actual creation of documentation may fall to the programmer(s). But the entire responsibility for defining documentation standards and absorbing the cost of creating and maintaining quality documentation should not rest with the DP department.

Everyone who benefits from good documentation has to commit to the idea of defining and enforcing documentation standards. Each part of the company needs to realize how they would benefit in the long run by taking the time and effort to confront the problem. If your DP department consistently supplied quality documentation for all DP systems, these are some of the ways that everyone would benefit:

> Policies and Procedures--would ensure that all necessary documentation was being created and maintained.
>
> Project Documentation--would provide ongoing communication about the resources required to carry out a request, as well as a historical record on which to base future time estimates.
>
> System Documentation--would provide an overview of each system, giving new programming staff a shorter learning curve toward efficiently maintaining those systems.
>
> Program Documentation--would make minor maintenance requests the short and simple things they should be, and make major modifications less complicated than they might otherwise become.

Operations Documentation--would allow the operations staff to do their job more quickly and efficiently.

User Documentation--would provide end-users with quick reference information, allowing them to solve some of the problems that come up without having to incur the cost of programmer or operator time.

Generally speaking, better documentation would lead to faster, better service from the entire DP staff. Whether you work in an environment where DP costs are billed directly to each department, or absorbed as overhead, time savings will always mean money savings.

Virtually every DP shop has some level of documentation available, yet often that documentation is not doing its job of "putting an end to the questions".

WHAT ARE THE PROBLEMS ?

I've run into all of the following problems to some degree:

Paper Overload-- This is when the documentation for the system you have just inherited consists of six three-inch binders, thirty-seven assorted manilla folders and a box or two of program listings. Every note that was ever taken during the life of the system has been saved...somewhere. Good luck finding anything that makes sense!

Inaccurate Information-- This includes misinformation, misleading information and obsolete information. The scary part is, this kind of documentation often looks very complete, very organized and very "official". It may have been top quality documentation at one time, but unfortunately it was so pretty that no one wanted to mess it up by updating it as the system changed.

Missing Pieces-- Somehow, the one piece of information that is vital to your understanding of the system is not available. Either it did not seem important at the time the documentation was created, or it seemed like the kind of thing that was just generally understood, or it used to be there but it disappeared at some time and was never replaced. By the time you figure it out you are so frustrated, you probably won't add your discovery to the existing documentation. Let the next guy struggle through it like you had to!

Unorganized Information-- Lack of organization can cause the same problems as missing documentation. The information may all be there, but if you can't find it, it still doesn't do you any good.

Unstandardized Documentation-- When generations of programmers and analysts have created documentation when and how they saw fit, you may end up with a wealth of perfectly valid documentation that is simply impossible to digest. The format, content and style of the documentation is upredictable from system to system. The lack of consistency prevents any intuitive familiarity with a system you've not worked with before. Every new system you are assigned to maintain is going to have its own set of problems. Does it have to have its own style of documentation too?

No, it doesn't. None of the problems listed above are unsolvable. Actually, they are all related. They are all surface problems, symptoms of a set of deeper, interrelated problems. They stem from the lack of defined documentation standards, which in turn stems from the unwillingness to invest the necessary time and money in first defining the standards and secondly living by them.

WHY STANDARDIZE ?

Many people fight standardization. They feel that it stifles their creativity, or they equate standardization with bureaucracy. This is especially true of programmers and analysts whose job is to creatively solve problems.

But let's face it. Documentation is supposed to be a factual representation of our systems, not an exercise in creativity!

The real reason most people resist standardization is because it is forced upon them. If you don't give people any say in the rules they have to live by, they are going to fight tooth and nail against them. They are going to fight whether they agree with the rules or not, simply because they feel imposed upon.

Involve the people who will create and use the documentation in defining the documentation standards. This approach fosters cooperation and commitment. Also, these are the people who can best define exactly what it is that is needed.

Build a team of qualified "experts", pulled from the programming staff, the operations staff, DP management and the end-user population. Assign them the task of defining documentation standards. The project should basically follow these steps:

I Define your needs
II Explore your options
III Detail your requirements
IV Select your tools
V Streamline
VI Write it down!
VII Put it to work
IX Revise and refine

Defining your needs is a critical first step. Appendix A lists the components we felt were important to include. This may give you a starting point, but it is essential to the success of the project that you define your own needs as clearly and completely as possible.

What is important from the user's viewpoint may be quite different from what is important to the programmer. The first time through, don't overlook anything. Keep track of all the ideas that come up. It may be helpful to group the ideas into categories like system documentation, program documentation and user documentation.

Exploring your options means looking at all kinds of available tools. Tools range from paper and pencil to sophisticated case tools and PC packages. Your shop may already have some forms that have been used successfully, but maybe they would be easier to use if you set them up as Vplus forms on the HP3000. Or maybe there is a PC package out there that produces the same kind of document, but provides some additional benefits like an integrated data dictionary system. Your final documentation system could be completely on paper, completely on-line on the HP3000, completely on floppy disk, or any combination of the above.

Detail your requirements. Now that you have a list of general needs and an idea of the tools available to you, you can become more specific about the form your documentation will take. There are many difficult choices involved, but the object is to match the right tool(s) to your requirements. At this stage, you want to define the essential ingredients for your shop's documentation. You want to define not only the general areas of documentation, but the actual details required in each of those areas. For instance, if you have decided that you need documentation on data files you may decide that that will include record layouts, blocking information, cross-referencing to all programs which update the file, or many other possibilities.

Selecting your tools now becomes much easier, because you can eliminate any that will not support your requirements. Having narrowed your range of choices, you must now find a balance between such considerations as ease of use, cost, training requirements and effectiveness. These choices apply whether you are comparing one PC package to another, an automated tool to paper and pencil methods, or anything in between. Pick the single tool or combination of tools that best meets your requirements.

Streamline. Review the list of requirements and the selected tool(s). Now is the time to take a hard look at your decisions. It would be nice to have every one of your documentation needs met, but is it practical? The reality is, if your standards are too cumbersome they will be worked around. Make some compromises, if necessary, so that you don't doom the standards to failure even before implementation.

Write it down! Your team has done a lot of work to define these standards. Don't leave their implementation to chance. Prepare a document that explicitly defines the new standards. Provide samples where possible. Provide instructions for using all the tools you have selected. Publish the standards and distribute them to everyone who is expected to follow them.

Put it to work. You need to plan for the implementation of the new standards. If they represent a major change from the old style, you may need to ease into the changes. People will have questions, and you will run into situations that have not been planned for and don't quite fit into your guidelines. It's not a bad idea to hold some status and review meetings to keep things on track.

Revise and refine. Keep in mind that just because you have defined some standards, you have not set anything in concrete. You'll need to work with the new standards on a day-to-day basis for quite some time before you can really see what's working and what's not. Don't paralyze yourself waiting for the perfect tool or the perfect solution. Give it your best shot, then work with it for a while. Over a period of time, the standards will be revised and refined until they really are meeting your needs.

Once you have a really good, workable set of documentation standards and guidelines, you will begin to see real gains made in the area of quality documentation. That first step leads to increased programmer and operator productivity, increased program and system quality, and better user support from the entire DP staff. Appendices A and B have been designed to give you some ideas and to help you get started. Keep in mind that the documentation effort needs continuing support from everyone involved: DP staff, management, and end-users. Given that support, you can break out of the cycle of:

```
             poor documentation
                     |
                     |
                     V
  high-cost, low-quality system support
                     |
                     |
                     V
             poor documentation
```

and into the more desirable cycle of:

```
             good documentation
                     |
                     |
                     V
  high-quality, low-cost system support
                     |
                     |
                     V
             good documentation
```

APPENDIX A
Documentation Requirements

Policies & Procedures

* Documentation Standards — Defining the required documents

* Coding Standards — Mainly used for COBOL, defining standards like paragraph numbering, format of IF constructs, use of intrinsics, copylibs and macros

* Production Account Standards — Defining security conventions, naming standards, account structure

* Project Management Guidelines — Guidelines on making time estimates, recording progress, making status reports

* Administrative Policies — Policies regarding things like work hours, vacation, sick leave

Project Documentation

* Service Request — Every request for service from the programming, operations or tecnical services staff is recorded on a numbered service request. The request is used as a vehicle to record estimated and actual time spent on a project, and to record the project sign-offs.

* Statement of Scope — A narrative on the scope of the project, used mainly on larger projects

* Project Plan — Includes plan of action, time and cost estimates for the project. Can be free form, or follow a preprinted outline of the typical project phases (analysis, code, test, etc...)

APPENDIX A
Documentation Requirements

* Project Log	Details action taken by date and amount of time spent
* Status Memos	Required on a regular basis for any substantial project
* Meeting Minutes	Minutes for any meeting involving the user and/or the project team

System Documentation

* System Flow Diagram	High level input/process/output
* Cross-reference of Jobs, Programs, Files, Reports	Defining the interdependence of processes and files
* System Problem Log	Includes date and time of problem, who reported it, who worked on it and how it was resolved
* Contact List	List of contacts for the system including programmer and/or analyst, main user contact, controller

Program Documentation

* Flow Chart, Warnier or HIPO	High level illustration of main logic flow
* Program Narrative	Narrative description of purpose
* Maintenance History	A running report of changes made including date, version number programmer and one-line description. Should include reference to the service request number.
* Imbedded Narrative	Imbedded comments anywhere they are needed to clarify the code

APPENDIX A
Documentation Requirements

* Compile Instructions	Which production compile job should be used and/or related copylibs, macro files or special prep requirements
* Current compiled listings	Stored in hanging folders, sorted by system

Operations Documentation

* Job Descriptions	Narrative of job purpose
* Job Run Instructions	Operator instructions for streaming and/or monitoring the job
* Job Restart Instructions	Operator instructions in case of job abort or system failure
* Report Distribution List	List of who is to receive report(s)
* System Contacts	List of contacts for the system including programmer and/or analyst, main user contact, controller

User Documentation

* User Manuals	Include system overview, data entry instructions, sample screens and reports, help section
* On-line Documentation	Where possible use self-explanatory input screens and/or on-line help facilities
* Data Services Handbook	Defines who's who in the DP what services are offered, and how to get help

APPENDIX B
Pitfalls and How to Avoid Them

Programmer Resistance--First, be sure to involve the programmers in defining the standards. Secondly, select the tools carefully to ensure ease of use. Avoid defining standards which appear to be no more than red tape to the people who must create and maintain the documentation. Keep the programmers involved in fine-tuning the documentation process.

Management Resistance--Sometimes it's the programmers who feel the problems of unstandardized documentation the most. Management may not be terribly cooperative, however, if they are approached only with complaints about the current state of things. Do some of the ground work before approaching management. Define the problem(s) in detail, look into some of the available tools and solutions. Go to management with a plan for change, well outlined and thought through, not simply another complaint.

End-User Resistance--In our shop, the users pay cold, hard cash for our services. They resist the idea of paying for more programmer time to create documentation. You need to involve the users in the definition of the standards so that they understand what you are asking them to pay for. You need to foster the idea that they are investing time and money now in order to save time and money later. Perhaps you can compromise by billing for only half the time spent on documentation.

Overkill--Once you have decided to define standards where there have been none, it is easy to get carried away. Everyone involved in the process of defining standards must keep in mind that in order for the standards to work, they must be easy to follow. Having end-users on the team will help in this respect because they will require that you explain exactly why each document is needed. They will not want to pay for busy work.

Reinventing the Wheel--The definition of standards is important, and for the most part your needs will be unique. It is possible, however, to spend far too much time defining your standards and developing processes to support them. One thing to keep in mind is that the goal is not perfection, or even excellence. The real goal is to make steps in that direction.

APPENDIX B
Pitfalls and How to Avoid Them

Anything can be designed to death, including standards. Set a time limit at the outset of the project. Set a date by which you will have the Documentation Standards document complete and stick to it.

It is also possible to spend more time than you anticipated in the implementation stage if you decide to design your own forms or, as we did, your own on-line system. Be sure you take enough time up front to investigate the tools that are already available before you decide to design your own. The Data Processing field is notorious for reinventing the wheel. If that leads to a new and improved model, that's great! But it's awful disappointing to spend months in development and then discover there was a good solution already available.

Asynchronous and Synchronous
Auto Dialing Equipment on the HP 3000
Why, When, and How

Benedict Bruno
STR Software Company
PO Box 12506
Arlington, Virginia 22209

Most HP 3000 computer systems have asynchronous modems connected to CPU ports to support remote terminal users. If connectivity to other HP 3000 or IBM mainframe (or compatible) computers is desired, then synchronous modems may also be connected to the Intelligent Network Processor (INP) accessed by subsystems such as Remote Job Entry (RJE), Multi-leaving Remote Job Entry (MRJE), and Distributed Systems (DS).

Very few people truly understand the fundamentals of data communications. Even fewer understand how the task of dialing the requested telephone number is performed. Believe me, it is not peformed by "magic", but rather by several different well defined methods.

In this presentation, I will describe how automatic dialing equipment performs in both asynchronous and synchronous environments. This will include the RS-366 interface as defined by the Bell 801C automatic call unit specification guide, the Hayes modem (and compatible) automatic call facility, and other dialing systems. In particular, proper MPE I/O configuration, troubleshooting suggestions, and sample programs to support automatic calling from an asynchronous terminal port will also be discussed.

Asynchronous Background Information

The HP3000 is not a full duplex machine with respect to asynchronous communications. This should not be a surprise to any of you now that you have been using your HP3000 for quite some time. Let us review how the HP3000 actually communicates with an HP (or compatible) terminal.

Since the HP3000 is not a full duplex machine, the designers created a flow control and character echo scheme that is quite unique in the computer industry. The scenario is the following: the terminal may only send when the HP3000 says it is ready. Furthermore, the HP3000 may only send data in a maximum of 80 character transmissions whenever the terminal acknowledges that he is ready. This strategy is known as the HP inquiry and acknowledge protocol or better known as ENQ-ACK. This strategy is found in Figure 1 below:

```
         HP3000                                Terminal

 FWRITE of data where
 data <= 80 characters ---------->
 inquiry (ENQ) ------------------>

                                     <----------- OK, send more (ACK)

     (Repeats until all data sent from CPU to terminal)


  FREAD informs the terminal
  that the HP is ready for data

  Read trigger (DC1) ------------>

                                                  Terminal sends data
                                     <------------ terminated with CR
```

Figure 1: HP3000 ENQ/ACK protocol

Notice how this protocols works: The terminal connects to the CPU (either modem or hardwired) and issues a carriage return. The HP3000 senses the correct speed and parity and echoes the received carriage return and adds a line feed. The HP3000 is then ready to receive an MPE command (the :HELLO command) by issuing the read trigger, i.e., the DC1 character. This read trigger is actually sent to the terminal when the FREAD (or READ or READX) intrinsic is executed. This DC1 character inhibits or unlocks the keyboard in order for you the terminal user to actually begin keying your command. Until this DC1 character arrives, you cannot begin keying any data.

The HP3000 will echo each character that you type at the keyboard. Depending upon the asynchronous terminal driver that you are using, this echo is implemented differently. The Asynchronous Terminal Controller (ATC) on the Series II and Series III machines would pass every received character directly to the CPU in order for the character to be echoed. Thus each character keyed at the keyboard would interrupt the CPU. This same technique is employed on the Asynchronous Data Communications Controller (ADCC) on the Series 30,33,40,44,48. With the creation of the Asynchronous Terminal Processor (ATP) the CPU is no longer interrupted for each character. Notice the term "processor"! The ATP provides the character echo directly to the terminal user. When the data is completed with the carriage return, the ATP then interrupts the CPU by passing the data buffer directly. You can see the obvious advantages of this method.

Now you are all experts on the ENQ/ACK protocol. But how do we attach a modem to an asynchronous port on the HP3000? More specifically, to an ATC, ADCC, or ATP port? Well, this does not seem that hard but we need some more background information.

All devices in the world of computers fall into two categories: either a DTE or DCE. Loosely defined, a device is classified as data terminal equipment (DTE) if it provides computer processing support (a CPU) and user input/output (a CRT). A device is classified as data communications equipment (DCE) if it provides transmission capability (modems, PBX, and so on). So what is the HP3000? A DTE of course. Wrong! Why? The HP3000 actually is a DTE in function, but it is cabled as a DCE device. This causes alot of people to incorrectly construct their cables. Why did HP do it this way? Because an HP terminal is a DTE device and a modem is a DCE device. If the HP3000 is cabled as a DCE device, then the same cable could be used between the terminal and the HP3000 or between the terminal and the modem!

Figure 2 below shows the data flow relationship between a DTE and DCE. The 25-pin DB25 connector usually associated with RS-232-C devices specifies that pin 2 is used for transmit data by the DTE device. Furthermore, pin 3 is used for receive data by the DTE device. A DCE device is exactly the reverse. Notice that the direction of the arrow points to the DCE for pin 2 and to the DTE for pin 3. This is fundamental in constructing our cable between the HP3000 and the modem.

Pin #	DTE	DCE
2	------->	------>
3	<-------	<------

Figure 2: DTE vs DCE data flow

But how do I connect a real DCE device such as a Hayes modem to the HP3000, that is, how does one connect two similar devices? Using Figure 1 and the state fact that a DTE transmits on pin 2, two like devices (DTE to DTE or DCE to DCE) must cross the transmit and receive pins. This crossover is simply constructed with pin 2 on the CPU end wired to pin 3 on the modem end. Next pin 3 on the CPU end is wirted to pin 2 on the modem end. The remaining signals necessary for modem controls must also be crossed as well. This includes pins 20 (data terminal ready or DTR) and pin 6 (data set ready or DSR), pin 4 (request to send or RTS) and pin

8 (carrier detect or CD), while pin 7 (signal ground or GND) is wired straight through. This cable is simply known as the HP3000 dataset cable with part number HP30062B. It is diagrammed in Figure 3 below. Notice that the additional signal of ring indicator (RI) on pin 22 is crossed with clear to send (CTS) to provide uniformity with products overseas and in the United States.

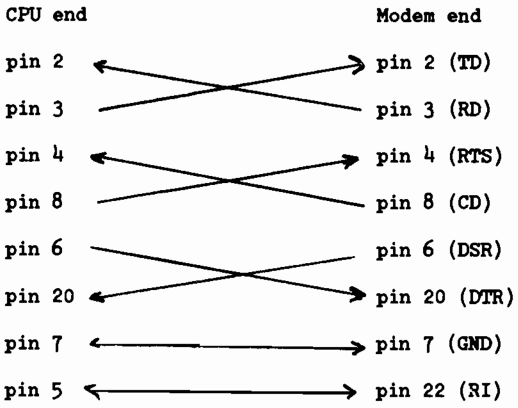


Figure 3: Connection of a modem to an HP3000 CPU port

MPE I/O Configuration

Now that the HP3000 is connected with the data set cable to the Hayes modem, the next step is to configure the modem within the MPE I/O configuration. The MPE :SYSDUMP program is used to modify the I/O configuration. The System Operation and Resource Management reference manual documents the :SYSDUMP program and its options. The items of importance here are the terminal type, subtype, and speed.

The terminal type of 10 specifies the standard supported HP terminal protocol of ENQ/ACK. The terminal type of 18 specifies any non-HP type of terminal and does not use ENQ/ACK. Specify the terminal type based upon the device that this modem will be calling. Since the incoming :HELLO command can specify the terminal type (;TERM=nn) and the outgoing program can specify the terminal type (FCONTROL 10), the terminal type should be for the device that will mostly use this port.

The subtype specifies whether the device is hardwired connection or a modem connection. Subtype of zero (0) specifies a hardwired connection and the HP3000 requires only three computer signals of transmit data (pin 2), receive data (pin 3) and signal ground (pin 7). Thus only three wires are necessary to connect an HP (or compatible) terminal to the HP3000 CPU using RS-232-C. This fact reduced the connector size for what HP classifies as "direct connect" ports on the ATP. Direct connect ports on the ATP use a special three pin connector rather than the standard DB25 connector associated with RS-232-C. Furthermore, with only three signals, the additional modem signals of data terminal ready, data set ready, carrier detect, etc. are not provided.

Could we attach a modem to either the three pin direct connect or the 25 pin "modem port" on the ATP (or ADCC or ATC ports) and configure the MPE I/O configuration for this port as hardwired? Certainly, but remember that the HP3000 CPU will not provide data terminal ready to the modem. Thus the Hayes modem would have to be strapped for forced data terminal ready. This means that the Hayes modem will always answer an incoming call regardless if the HP3000 CPU is really up. Not really a good idea.

The proper way to attach a modem to the HP3000 is with the subtype of 1 specifying modem support. All modem signals are supplied with subtype 1. Remember from our earlier discussion that all the signals are reversed at the CPU end! The CPU monitors data terminal ready raising the signal when the HP3000 is running. The signal is momentarily dropped when a disconnect occurs, when the remote user issues the :BYE command, or when the :ABORTJOB command is used on the session number of the remote user active on this port. This sounds more of what we want, control of our session in the case of disconnects and logoffs.

The last item, speed, is really of no consequence because MPE will "sense" the speed from that specified with the character string received at the CPU. A Hayes modem supports speeds of 300, 1200, and 2400 baud. Depending upon the speed of the remote terminal user, the local modem at the CPU will synchronize with the remote modem at the terminal. The HP3000 CPU will then sense the correct speed and echo the carriage return with a line feed and the colon prompt.

Asynchronous Auto Dialing

Our objective is to have a program open the logical device number with our Hayes modem attached. With our configuration parameters above, can we do it? Almost. The Hayes modems provide a simple command mode interface that can easily be written by any DTE device. We can specify any speed of 300, 1200, and 2400 baud with the FCONTROL intrinsic. But we cannot open the port and issue an FWRITE (with the Hayes "AT" commands) without the HP3000 knowing that the "unit (modem) is ready". By this we mean that the HP3000 must see carrier detect (CD) and data set ready (DSR) high from the modem in order for the FWRITE to complete.

The Hayes modem has dip switches specifying a number of options. One of the options specifies the signals of carrier detect and data set ready as either normal or forced on. We need this to be forced on. While we are at it, specify the dip switches as in Figure 4 below:

DTR from interface
Terse command mode responses
Enable command mode responses
Disable echo of command characters
Auto answer enabled
CD and DSR forced high
Enable command mode

CTS forced on
Dial-up operation
Blind dialing method
Asynchronous operation

Figure 4: Hayes modem strap settings

Notice that DTR is specified from the interface, i.e., from the HP3000. With subtype 1 the HP3000 provides this signal; otherwise we would have to force it high for subtype 0. Commands executed by the Hayes modem always return a response. This response can be a character string such as "OK", "CONNECT", etc. or can be a single digit also known as terse mode. In terse mode, the Hayes modem transmits a single ASCII character terminated by a carriage return. Just what we want, data terminated with a carriage return. Obviously we want these responses to our command and we do not want the command characters echoed back to the CPU. The remaining signals are self explanatory. One should notice that the only signal we MUST have is that of CD and DSR forced high. All of the remaining signals can be specified with an appropriate AT command that we may send programmatically.

Program Pseudo Code

Now that we have the MPE I/O configuration correct, the Hayes modem strapped, and the cable connecting the modem and the CPU, we are now ready to develop our program to use the autodialer and control the modem. will describe this as a series of steps that can be programmed in any language on the HP3000.

Step 1: Open the port

We must first open the logical device number assigned in our I/O configuration to the attached modem. The FOPEN intrinsic may be used with a file reference to specify which logical device number the modem is on as follows:

```
  :FILE PORT;DEV=ldevnumber
  filenum = FOPEN (*PORT, %400, %4, -80)
```

Notice that the FOPEN intrinsic references the formal designator of "*PORT" requiring that a file equation for PORT be present. The file options (FOPTIONS) of octal 400 (%400) specify carriage control and a new file. More on carriage control later. The access options (AOPTIONS) value of %4 allow us to read and write to this device. The record size of 80 bytes is for a standard terminal. You may need to increase this to talk to your device.

Step 2: Disable echo

Since the modem will respond to our AT commands with a response, we do not want the HP3000 to echo back this response to the modem. Recall, the HP3000 always echoes incoming data back to the sending device. This will obviously confuse the modem if this data is echoed back! Use the FCONTROL intrinsic with the returned file number from step 1 as follows below:

```
  FCONTROL (filenum, 13, ldummy)
```

Notice that parameter 13 disables the echo feature. No parameter value is required to disable echo. Hence the value of the "ldummy" variable is meaningless and unused.

Step 3: Specify modem speed

Recall that the modem supports a variety of speeds. We must specify the speed of the modem to match that of the device that we are trying to call. The FCONTROL intrinsic allows for the specification of the input and output speed. As it turns out, the HP documentation states that specification of both the input and output speed may cause problems. Therefore, we only need specify the output speed with option 11. The speed is actually specified in characters per second (not as the baud rate) in the ldummy parameter. Values of 30, 120, and 240 characters per second correspond to 300, 1200, and 2400 baud respectively. This can be done as follows:

```
  ldummy = 30
  FCONTROL (filenum, 11, ldummy)
```

Step 4: Disable MPE automatic CR/LF

For each record received by the HP3000, i.e., for each read completed with a carriage return from the device, the HP3000 will echo an automatic carriage return and line feed back to the device. Obviously, this will create havoc for the modem when we have just received the command response. This automatic carriage return and line feed can be disabled with the FSETMODE intrinsic as follows:

```
  FSETMODE (filenum, %4)
```

Step 5: Change DC1 read trigger to CR

Recall from the background discussion earlier, that the HP3000 will transmit the DC1 character as the read trigger. The remote device may only transmit data to the HP3000 when this read trigger character arrives. The Hayes modem has no idea what to do with the DC1 character. The solution is to define some character that the HP3000 may send as the read trigger that will in turn complete the information that the modem wants. How about a carriage return? By using a carriage return as the read trigger, whatever data was previously transmitted by the HP3000 in the FWRITE will now be complete from the FREAD! Not only that, but we are guaranteed at the HP3000 of being ready for the modem response.

The DC1 read trigger is changed to a carriage return with the FDEVICECONTROL intrinsic. This intrinsic accesses the Work Station Configurator in order to perform the change. The Work Station Configurator is a nice product supplied by HP to change many of the attibutes in communicating with an asynchronous device. These "changes" are placed into terminal type or TT files in PUB.SYS and can be referenced programmatically and within the I/O configuration.

The FDEVICECONTROL specifies that the read trigger character be placed in an integer variable (16 bit word). The actual character is placed in the second half of the word, i.e., bits 8 thru 15. The first half of the word is ignored and should be set to binary zero. The parameter of 192 specifies the Work Station Configurator. The parameter value of 32 specifies the read trigger should be accessed. The parameter value of 3 specifies first change the read trigger to that supplied in I and then display the value in I. This validates the change. If an error occurs, the value is returned in the error variable which can be used in a call to the FERRMSG intrinsic to locate the message from the error catalog.

```
  I=%000015
  FDEVICECONTROL (filenum,I,1,192,32,3,error)
```

Step 6: Enable terse messages; disable command echo

As discussed earlier, using terse messages from the Hayes modem yields a single ASCII digit response to our AT commands. This digit response is completed with a carriage return. The modem must not echo the command responses because the HP3000 will think it is valid input from the remote device. The echo of command data should be disabled. Terse messages and disabling command echo can be requested with a single command to the modem. Notice that the data is not terminated with a carriage return (nor a line feed) as specified with the FWRITE parameter of octal 320. Why? Because we will transmit the carriage return as the read trigger when we execute the FREAD intrinsic.

```
  FWRITE ("ATV0E0", -6, %320)
```

Step 7: Read digit response terminated with CR

Once the modem completes our command in step 6, a single ASCII digit is returned terminated with a carriage return. This can be requested by the HP3000 using a two character terminal read. The received data can be checked against the valid responses supplied by the modem manufacturer. The Hayes specification states that zero (0) indicates successful completion of the previous command.

```
  FREAD (answer,-2)
```

Step 8: Dial the number

Now that the modem is ready, we can simply send the dialing sequence. This dialing sequence will include the digits of the telephone number, delay characters, wiqait for second dial tone, etc. Place the dialing sequence in a buffer and send it to the modem with the FWRITE intrinsic as in step 6 above. Note that the FWRITE specifies no carriage return.

```
  FWRITE ("ATDT 1-703-689-2525", -length, %320)
```

Step 9: Read the modem response

As before, the previous command is completed by issuing the FREAD intrinsic. The FREAD transmits the read trigger character (carriage return) and awaits a two character response from the modem. The modem places the call and determines status. Locate the list of valid responses for your particular modem. You may also wish to enable a timed read for the modem response. Use the FCONTROL with a parameter of 4 and an integer value for the number of seconds that the HP3000 must wait for a response.

```
  FREAD (answer,-2)
```

Step 10: Continue programming!

Assuming a connected response from the modem, you are now able to continue with your program. Don't forget now that you are connected, the carriage return may be inappropriate as the read trigger. This depends on the device to which you are communicating.

Should you wish to disconnect abnormally, the Hayes modems may be interrupted while connected and return to command mode. The default string of three plus characters (+++) when received by the modem invokes command mode. How might we do this? Simply issue a two character FWRITE of "++", change the read trigger with FDEVICECONTROL to "+", and then issue the FREAD. You will then be in command mode. Remember to then change the read trigger back to a carriage return in order to continue with the AT disconnect command.

Synchronous Auto Dialing

The Intelligent Network Processor (INP) provides all synchronous communications capabilities for the HP3000. One of the most important features of the INP is its ability to establish the telephone call of switched (dialup) telephone lines automatically, eliminating manual intervention. Autodialing with the INP is completely different than that described above with the Hayes autodialing method. As before, let's describe how to configure and cable the INP for autodialing. Then continue with a discussion of the auto call unit itself.

Configuring the INP

Depending upon the HP3000 series computer, different hardware and software capabilities are available on the INP. The INP for the Series III requires two boards which unfortunately do not support the automatic call feature. The INP for the Series 30, 33, 4X, 6X and 70 computers has been available as the original non-auto dial feature (part number 30020A or 'A' board) and as the current auto dial feature (part number 30020B or 'B' board). The INP board for the Series 37 also provides auto dial support but is completely different from the 'B' board and cannot be interchanged. The reader should note that HP offers the INP within its 'Link Services' product offerings making it somewhat easier to bundle the INP, cable, and download software within one product.

The :SYSDUMP command invokes the MPE I/O and system configurator program. Many parts of this dialogue are also supplied with any of the system startup procedures, i.e., WARMSTART, COOLSTART, UPDATE, COLDSTART, and RELOAD. I am sure that most of you have created or updated the system I/O configuration of your own HP3000. However, should you wish to add the automatic calling feature to your INP, you may notice that it is readily documented within any of the HP3000 manuals. This includes the INP Installation Reference Manual. So what is the trick? How does one do it?

The three prompts of 'DIAL FACILITY', 'ANSWER FACILITY', and 'AUTOMATIC ANSWER' are provided by the SYSDUMP dialogue in order to properly configure the dial and answer capabilities of the equipment connected to this INP. Obviously the 'DIAL FACILITY' prompt appears related to specifying that this INP is to utilize the automatic calling feature but the System Operation and Resource Management Reference Manual and the Data Communications Handbook specify that values of 'YES', 'NO', or [RETURN] are the only valid responses. The YES response specifies that a telephone handset is attached in order for the operator to manually dial the phone; the NO response is identical to the [RETURN] response in that no telephone handset is attached. The reader should note that MPE issues a system console request with the specific phone number that must be dialed. When the connection is made, the operator simply replies to this console request.

However, this still does not answer the question of enabling the automatic dialing feature. Since HP did not want to add another prompt to the dialogue, the developers decided to allow another value. By supplying the logical device number of the INP to the 'DIAL FACILITY' prompt (which is obviously the value supplied to the first prompt of 'LOGICAL DEVICE #'), we enable the INP to support automatic dialing.

Furthermore, how can you determine from a SYSDUMP (or for that matter SYSINFO) listing of the I/O configuration and CS devices which INPs have the automatic calling feature enabled and which do not? Consider the output from the SYSDUMP program in Figure 5. Two identical INPs are displayed utilizing 4800 bps modems. One INP provides automatic dial support and the other provides manual dial support. Which is which? You guessed it, there is no way to determine it! When a customer explained having difficulty in establishing the automatic dialing feature on the INP, the first thing I did was to configure the entry once again. This way I was certain that the I/O configuration was correct.

Figure 5: List I/O and CS Devices

```
DEV  #   N  H Y TYPE  TERMINAL   WIDTH  DEV            NAME    CLASSES
 #       I  A P       TYPE SPEED
         T  N E
11  181  0  0 17 0                0      0             IOINPO  INP
12  181  0  0 17 0                0      0             IOINPO  INP

LDN PM PRT LCL TC  RCV   LCL   CON   MODE   TRANSMIT  TM BUFFER D DRIVER
           MOD     TMOUT TMOUT TMOUT         SPEED        SIZE  C OPTIONS
11  0  X    X  X  20    60    900   OIA    600        1  1024   N  0
12  0  X    X  X  20    60    900   OIA    600        1  1024   N  0
```

Notice that most of the columns are self explanatory. I was happy to see that with the Update #1 (January 1985) release of the System Operator and Resource Management Reference Manual the values of the 'MODE' field are now documented. The reader is referred to page 7-4 for a description of each of the heading and entry values. The mode of 'OIA' specifies dial out, manual answer, and automatic answer. We do not know if the auto dial feature is enabled or disabled. Maybe an additional mode value of say 'U' (from aUtomatic dial) can be added to the list and displayed if the automatic dial feature is enabled.

The reader should also notice that both logical device numbers of 11 and 12 specify the same Device Reference Table (DRT) entry. Thus an INP may have multiple configurations in order to support several devices and capabilities. Therefore, you may enable auto dial support on ldev 11 and disable auto dial support on ldev 12. I strongly suggest that you add meaningful device class names of 'AUTODIAL' and 'MANDIAL' respectively in order to avoid further confusion!

The reader should also beware that if the auto dial feature is enabled on ldev 11, that MPE will expect to find an automatic calling unit attached. If not, then nothing will happen. No console request will be issued for the operator to dial the remote number should you attach a telephone handset to place the call manually. The same is true if the INP is configured without auto dial support and you connect an automatic calling unit to the INP.

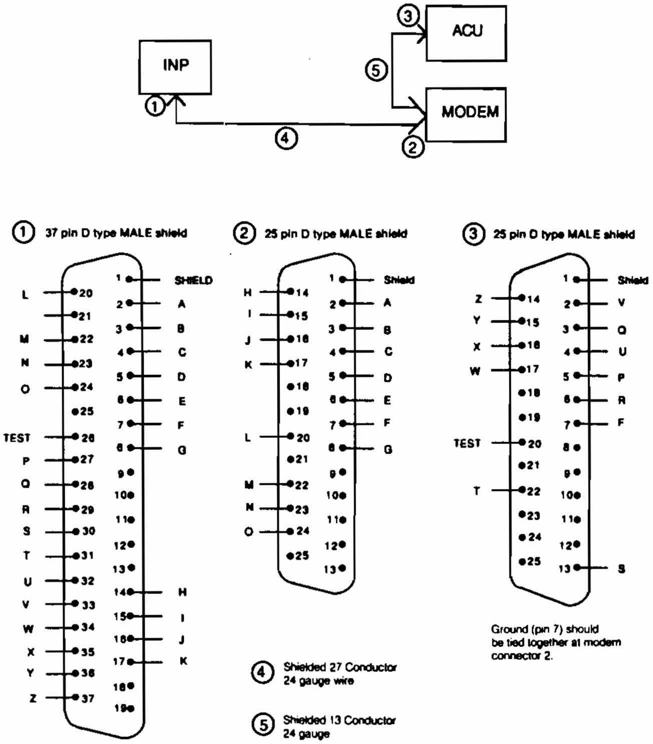
HP30221G Auto Dial Cable

The automatic dial facility requires a special cable between the INP junction panel connector and the necessary external equipment of the modem (2400bps, 4800bps, etc) and the automatic calling unit. This is the first mention of the automatic calling unit. Synchronous modems were developed requiring an external device place and control the dialing of the telephone number. This is far different than many of the asychronous modems of today that are 'Hayes compatible' in that the modem and the dialer are all in one nice package.

The defacto standard for the automatic calling unit is the Bell 801C auto call unit (ACU). Connection to the Bell 801C ACU requires the RS-366 interface. This interface is well documented in the Bell 801C-LI/2 Data Auxiliary Set Interface Specification, PUB 41203.

Although the RS-366 interface utilizes the same DB-25 connector as the RS-232-C interface, the two are vastly different. Attachment of the ACU and modem to the INP requires the HP30221G cable (or 'Y' cable as it is often called) as diagrammed in Figure 6. Notice that the connectors differ at each end of the cable. The connection to the INP requires a 37 pin connector in order for all 27 signals be passed to the INP. The connection to the ACU and the modem utilize the standard 25 pin male connector. However, notice that the pins used by the ACU appear similar to the modem signal usage. This appearance is totally misleading.

Figure 6: HP30221G cable schematic



The pin designation of the RS-232-C standard is diagrammed in Figure 7. Notice that the direction of the signal clarifies whether it is an inbound or outbound signal. For example, the HP3000 raises data terminal ready (DTR) on pin 20. The modem then reacts by raising data set ready (DSR) on pin 6 when the link is established. The remaining signals will not be discussed.

Figure 7: RS-232-C Pin Designations

		To INP (<---)	
Pin	Name	To modem (->)	Description
1	FG		Frame ground
2	TD	------------->	Transmit data
3	RD	<-------------	Receive data
4	RTS	------------->	Request to send
5	CTS	<-------------	Clear to send
6	DSR	<-------------	Data set ready
7	SG		Signal ground
8	DCD	<-------------	Data carrier detect
9		<-------------	Positive DC test voltage
10		<-------------	Negative DC test voltage
11			Unassigned (not used)
12	SDCD	<-------------	Secondary data carrier detect
13	SCTS	<-------------	Secondary clear to send
14	STD	------------->	Secondary transmit data
15	TC	<-------------	Transmitter clock
16	SRD	<-------------	Secondary receive data
17	RC	<-------------	Receiver clock
18		------------->	Receiver dibit clock
19	SRTS	------------->	Secondary request to send
20	DTR	------------->	Data terminal ready
21	SQ	<-------------	Signal quality detect
22	RI	<-------------	Ring indicator
23		------------->	Data rate select
24	TC	------------->	External transmitter clock
25		------------->	Busy

The pin designation of the RS-366 standard is diagrammed in Figure 8. Notice that the signals do not resemble any of those specified in the RS-232-C standard in Figure 7.

Figure 8: RS-366 Pin Designations

Pin	Name	To INP (<---) To ACU (--->)	Description
1	FG		Frame ground
2	DPR	------------->	Digit present
3	ACR	<-------------	Abandon call, retry
4	CRQ	------------->	Call request
5	PND	<-------------	Present next digit
6	PWI	<-------------	Power indicator
7	SG		Signal ground
8			Unassigned, not used
9			Positive DC test voltage
10			Negative DC test voltage
11			Unassigned (not used)
12			Unassigned (not used)
13	COS	<-------------	Call origination status
14	NB1	------------->	Value 1 of digit
15	NB2	------------->	Value 2 of digit
16	NB4	------------->	Value 4 of digit
17	NB8	------------->	Value 8 of digit
18			Unassigned (not used)
19			Unassigned (not used)
20			Unassigned (not used)
21			Unassigned (not used)
22	DLO	<-------------	Data line occupied
23			Unassigned (not used)
24			Unassigned (not used)
25			Unassigned (not used)

The pin assignments in Figure 8 require some explanation. The digit present (DPR) signal on pin 2 is set high by the INP whenever the digit is sent by the INP on pins 14, 15, 16, and 17; otherwise it is set low.

The abandon call retry (ACR) signal on pin 3 is set high by the ACU to indicate the probability of an unsuccessful completion of the call attempt. The INP alerts the user with a CS error 59 suggesting another call attempt.

The call request (CRQ) signal on pin 4 is set high by the INP to request the ACU to originate a call. This signal remains high througout the entire data communications transfer and is set low when the INP wishes to disconnect the telephone.

The present next digit (PND) signal on pin 5 is set high by the ACU to control the presentation of digits on the digit signal circuits. When set on, the ACU is ready to accept the next digit indicated on pins 14, 15, 16, and 17 as set by the INP. The digit is read when the INP sets DPR high. When set low, the INP may set low DPR and again present the next digit on pins 14, 15, 16, and 17.

The power indication (PWI) signal on pin 6 is set high when the ACU detects available power.

The call origination status (COS) signal on pin 13 is set high when the ACU has completed the call request function. Previously, this signal was named data set status (DSS) and later renamed to the call origination status. Once set high by the ACU, the INP begins its data transfer depending upon the specific protocol being emulated, i.e., 3780 on RJE, HASP on MRJE, etc.

The digit signal circuits on pins 14 (NB1), 15 (NB2), 16 (NB4), and 17 (NB8) present the binary coded value of the digits 0 through 9. Obviously, this is the reason why NONE of the HP data communications subsystems (RJE, MRJE, DS, IMF, and MTS) allow for any characters in the phone number other than the digits separated by the dash character. Thus, the Hayes modem phone number values such as 'k' or ',' for pause cannot be issued. This misconception was always a common question asked by customers when I was at HP.

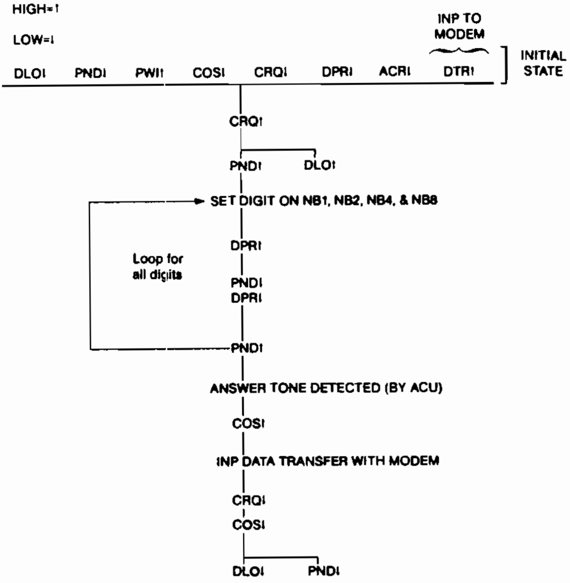
The last signal of data line occupied (DLO) on pin 22 is set high indicating that the data communications channel is in use for automatic calling, data transfer, voice, or testing. When set low and the power indication signal is set high, i.e., the unit is powered on, then the INP may initiate a call request.

Now that the signals are discussed, let me simply explain the automatic calling process between the INP and the ACU. The process is diagrammed in Figure 9. The initial signals of the ACU are set low when the power is off. When the ACU detects available power when the unit is turned on, then the PWI signal on pin 6 is set high by the ACU. This will be the true 'idle state' of the ACU when not in use by any of the HP3000 data communications subsystem software. The INP initiates the call request by setting the call request signal on pin 4 high. The ACU responds by setting both the present next digit signal on pin 5 and data line occupied signal on pin 22 high. The data line occupied signal remains high throughout the entire call until the INP terminates the line. The ACU requests the value of the telephone digit when the present next digit signal is high. At this point the INP places the digit in the digit signal circuits on pins 14, 15, 16, and 17. Once complete, the INP incates the digits are present by setting pin 2 high. Once detected by the ACU, the present next digit and digit present signals are set low by the ACU and INP respectively. This process repeats for each of the digits of the telephone number to be dialed, i.e., ACU sets high PND, INP sets digit signal circuits and high DPR, ACU sets low PND, and INP sets low DPR. Once the phone number is complete, the ACU attempts the call.

The ACU must indicate successful completion or failure to the INP. The call origination status (COS) signal on pin 6 is set high if successful or low for a failure. In addition, the abandon call retry signal on pin 3 is set high for a failure.

If successful, then the ACU is released from the data transmission circuit in that the INP now sends and receives signals from the modem. When the INP is ready to terminate the data communications activity, the DTR signal is set low which releases the modem and the CRQ signal is set low which releases the ACU. The ACU responds by setting low the call origination status, data line occupied, and present next digit. This leaves all equipment in the 'idle state' ready for the next call request.

Figure 9: INP-ACU Interface



Supported External Modems and ACUs

Hewlett-Packard tested and certified specific modems and auto call units when the auto dial support was added to the INP in late 1981. Vendor equipment from both Bell and Racal-Vadic were used. Specifically, the Bell 801C and Racal-Vadic VA811 auto call units were certified in connection to the INP. Lab engineers connected the Bell 201C and Bell 208B synchronous 2400 bps and 4800 bps modems to the INP.

HP concluded that any Bell 801C compatible ACU and Bell 201C or Bell 208B compatible modem are supported by the INP. However, customers were warned that other vendor equipment must be tested and certified. This process could be performed with HP assistance on a time and materials basis.

Case History Example

Over the years, I have been involved with several customers attaching equipment other than Bell and Racal-Vadic to the INP for automatic dialing of the HP data communications products. Each of the HP sales offices used either the Bell 801C or the Racal-Vadic VA811 with no installation or operational difficulties. However, I have experienced some problems with testing and certifying other vendor equipment to the INP. Most of these were resolved, but by far the most interesting and troublesome case deserves mention here.

MCI Telecommunications, Inc. had already been using nearly every data communications product to implement the MCI Mail Electronic Mail Network (see "Integrating HP Data Comm for Electronic Mail", INTERACT, October 1986). However, MCI wanted to automatically dial remote computers using DSN/RJE with the 3780 protocol.

Rich Oxford, MCI Technical Support Manager, contacted HP for assistance when the Penril 8208 modem (Bell 208B compatible at 4800 bps) already installed and working on the HP3000 Series 64 could not function with the Penril 8801C ACU (Bell 801C compatible) with the INP configured for auto dial. Tom Benedict, HP Network Consultant and I (HP Senior Applications Engineer at that time) responded.

We first determined that the I/O configuration was indeed correct. When RJE attempted the dial sequence from the #RJIN command, the ACU and modem performed the correct signals as in Figure 9. The INP appeared hung, experiencing a brief period of idle activity terminating in CS error 59. This included the call request by the INP, the present next digit by the ACU, the digit sequence, and so on. The ACU just never released control of the telephone line to the modem in order for the INP to continue.

We suspected either the ACU or the MCI constructed cable. The ACU and modem were sent to Penril for repair while we checked the cable on the HP sales office machine. The cable tested fine with the HP3000 Series 68 in the Rockville Sales Office. The ACU and modem were verified from the Penril engineers. So what was the problem?

After no success on site, we connected the MCI equipment on the HP3000 Series 68 at the Rockville Sales Office. Again with no success. At this point, Tom and I reached the time committment of the support contract and suggested that all continuing work would be on a time and materials basis. (I offered to assist Rich at night on my own time to develop a solution. I was intrigued by this one!)

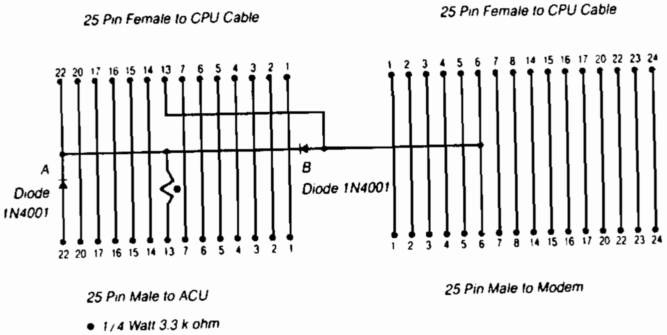
Returning to MCI, Rich and I studied the 801C specification directly from the Bell technical publication given to us from Tom. Rich decided to place breakout boxes and data scopes between both the ACU and the modem. This permitted us to not only monitor but bridge any signals from the equipment.

We solved the problem with some clever observations by Rich. The call origination status (COS, pin 13) is always low on the Penril ACU. Data line occupied (DLO, pin 22) is set high by the ACU when the INP initiates the call request (CRQ, pin 4). In the previous discussion of Figure 9, the 801C specification clearly states that DLO must remain high by the ACU during the entire call. Using the breakout box between the ACU and the INP, Rich noticed that when the ACU detected answer tone thereby releasing the phone line to the modem, the DLO signal was set low! Which is totally wrong based upon the Bell specification, i.e., not truly Bell 801C compatible! How could this ever work? Both vendors (HP and Penril) specify their equipment worked satisfactorily and yet we noticed that they did not worked together!

Therefore the Penril failed on two Bell 801C requirements. DLO must remain high during the entire call and secondly COS follows the DSR signal. The trick is to make the DLO signal appear high to the INP when the Penril equipment mistakenly sets it low. This must happen once the ACU detects answer tone thereby releasing the line to the modem. The modems synchronize which sets the DSR signal high. At this point, the COS and DLO signals were jumpered with DSR (pin 6) of the modem cable and IT WORKED! The COS, DLO, and DSR signals all remained high, the INP detected connection, and RJE continued with the user commands. The line terminated correctly when the RJEND command was encountered.

The final conclusion involved the construction of an adaptor cable between both the ACU and modem to the INP cable as diagrammed in Figure 10. Notice that two 1N4001 diodes and a 3.3k ohm resistor are required in this cable. Using layman terms, let me explain how the cable works.

Figure 10: Penril adapter cable



The resistor and pair of diodes are connected as a hardware logical OR gate. A diode permits positive voltage to flow through the direction of the arrow. When DLO (pin 22) goes high by the ACU, diode A permits this voltage to flow through to the pin 22 at the CPU end. Furthermore, this positive voltage cannot pass through to pin 6 of the modem nor pin 13 because of diode B (notice the direction of the arrow). When the call is answered, DLO goes low by the ACU, but the DSR signal is set high by the modem. This positive voltage passes to COS (pin 13) and through diode B in effect making DLO appear high to the INP. In fact, the INP never even notices that the signal drops at all!

The cable corrects the misinterpreted implementation of the Bell specification by the Penril ACU. The resistor is necessary in order to bias the diodes. The call origination status (COS) signal was previously named distanct station connected (DSC) which typically is set the same as data set ready (DSR). This explains why the COS lead on the CPU end is tied to the DSR lead in order to make the COS signal follow that of DSR. Incidently, total cost of the components from Radio Shack was only $2.20.

The Bell 801C ACU is implemented by a number of vendors. As you can see, its actual function is quite well defined.

Additional Features

Synchronous modems have been further enhanced during the past several years. Modems have become faster, cheaper, smaller, and more efficient at utilizing switched telephone lines. One of the most notable enhancements to the synchronous modems is the addition of a built in auto dialer which elminates purchasing a separate 801C type ACU. In fact, a 4800 bps (Bell 208B compatible) synchronous modem with built in 801C ACU is packaged even smaller than the older 801C ACU from the Bell System! (Of course, even less expensive.)

In addition to combining the auto call unit function within the modem itself, manufacturers are now offering other types of dialing interfaces. It is possible to utilize an asynchronous port to control the auto dialing function of a rack of synchronous modems (usually 16 modems). The synchronous data comm products on the HP3000 require the use of an INP and does not allow for this asynchronous type of auto dial control.

Another type of dialing control mechanism is implemented directly by the modem only. In this particular method, the host CPU initiates a synchronous dialogue to control the auto dial feature of the modem. This feature was pioneered by Racal-Vadic as the Synchronous Auto Dial Link or better known as SADL. The SADL method provides IBM bisynchronous and HDLC protocol support. Using SADL with bisynch, the host computer would control the modem as if it were another computer in that it would send to and receive from the modem auto dial information directly. Once the modem makes the remote connection, the auto dial function has been completed and control is no longer needed.

SADL is a very effective and interesting method of controlling the auto dial function for synchronous lines. However, in the current implementation of the HP3000 RJE product by HP, it cannot be used as a viable dialing option from the INP. We must resort to the standard 801C auto call unit.

Summary

In this paper I have discussed the types of auto dialing equipment in use in both asynchronous and synchronous environments on the HP3000. The pseudo code programming example provides the essentials for accessing an asynchronous auto dial modem from an HP3000 terminal port. The case history example disproves the fact that "oh, any compatible equipment will work". Good luck!

References

1. Bell 801C-LI/2 Data Auxiliary Set Interface Specification, November 1976, Bell System Data Communications Technical Reference.

2. System Operation and Resource Management Reference Manual, Update #1, January 1985, Hewlett-Packard Company.

3. Data Communications Handbook, Third Edition, April 1981, Hewlett-Packard Company.

4. Penril 8801C Automatic Calling Unit Operations Manual, Publication 8801C-OM-0381, Penril.

5. Penril Data Modem 8208A/B Instruction Manual, 60A047A01-01, Rev A, Penril.

Data Download
HP3000 to any Vectra Clone

Andre Cruz
Douglas L. Grossman
Merrill Lynch & Co., Inc.
May 18th, 1988

The purpose of this article is to show, by example, the concept of a Data Download. The body of this work is divided into four parts: 1) A definition of the topic; 2) an overview of the application detailed in this story; 3) traversal of the products and techniques involved in constructing this specific application; and, 4) a conclusion.

The Definition

Given the migration of defined, fixed end user functionality from the central host to one or more local processors, the Data Download is the scheduled data refresh, via electronic connectivity, providing local application processing input without continuous, direct online host interface during local application execution.

The term Data Download is not new. It should not be confused with Application Download. The former is a far more extensive topic. There, the central maintenance and downloading of common local application programs is involved, not only the refresh of the data. Its scope is beyond the confines of this small column.

The Application

The client organization processes compensation payments for salesmen located in branch offices. The product is insurance. The salesmen are known as Regional Insurance Specialist (RIS). An RIS requires monthly statement generation detailing compensable transactions. Statement processing is handled centrally from home office by the Compensation Unit. Paper statements are couriered to the RIS force in the branch offices.

Sale of product results in issuance of production credits. These are reflected in the monthly compensation sheet of the RIS. Production credits can be likened to green stamps, i.e., Sale of product accrues various amounts of production credits; US dollars are distributed to the RIS as a percentage of total production credits collected, etc. Product returns and cancellations cause chargeback against the compensation sheet of the RIS. In essence, payout must be calculated and reported to the branch sales force in an exact and timely manner.

Previously, Compensation Unit personnel transcribed individual transactions from home office generated host production reports. These figures became input to local processors for ultimate statement generation and distribution. As volumes grew, transcription time increased proportionately. The client requested a more expedient methodology.

The Technique

Prior to implementing the download, the existing system was composed of the following. Weekly production reports detailing RIS compensation were produced on the host HP3000 series 70. The output was recorded and summarized for entry into a local DBase III+ application, resident on a stand alone IBM PC/XT. The HP produced the Total RIS Compensation Report; the PC generated the Individual RIS Compensation Statement. A procedure was needed to remove the human element needed to reenter data into the PC.

The HP report was generated via a COBOL program. It accesses multiple Image sets to create the printer spool file. The first step was to modify the program enabling it to additionally write output to an MPE disk file. Because statements were produced on a monthly basis, some modifications were required to summarize the weekly data. These changes were not extensive. They were incorporated into weekly batch production.

Next, a package called DataExpress was used. The product is supplied by a firm known as Imacs Systems, Corp. in Marina del Rey, California. "DataExpress is a program that allows users to extract data from multiple files and/or data bases on the HP3000, to manipulate and reformat it for use in another application program. It allows users to make use of the extracted data in programs on their microcomputer, or to use the data in other HP3000 programs." [*1] A review of the DataExpress package is not possible. For now, let it suffice that pleasurable results have been attained from product usage.

DataExpress was used to convert the ·MPE file to DBase format. The output from DataExpress resides on the HP until downloaded. DataExpress has an option that allows downloading of the file when processing is completed, but this requires the user to run DataExpress online, from the PC. The example uses a combination of SuprTool, a product by Robelle, and DataExpress. SuprTool was used to extract the information quickly from an HP Image dataset. This is done because DataExpress is much slower doing "simple" extractions. The SuprTool output, an MPE file, is now read into the DataExpress procedure, where it is sorted, on three items, and summarized on each level. (Note: DataExpress will not allow summarizing unless it sorts the file first, therefore, it is of no use to sort the file prior to using DataExpress.)

Another problem arose in this SuprTool/DataExpress interface. Numeric output to SuprTool ASCII files is not padded with leading zeros. DataExpress generated a data exception when attempting to read a numeric field with leading spaces. A COBOL program was written to inspect the field and replace spaces with zeros.

DataExpress requires interface with another package to perform the actual file transfer. In this case, a product called Reflection-3, by Walker Richer & Quinn, stationed in Seattle, was used. "Reflection enables the IBM Personal Computer or workalike PC to emulate, or operate like, the terminal of a much larger computer system. Reflection emulates Hewlett-Packard ... You can transfer data between the PC and a host computer, and send data from the display memory of the PC to a printer or a disk file." [*2]

Reflection requires no added PC hardware. The program utilizes the standard serial ports soft labelled COM1 and COM2. Either modem or direct connections are acceptable.

Originally, usage of the Reflection product was ceded to operations. Because the generation of the compensation figures and the DBase file was a production job, operations controlled the actual download. Monthly, operations invoked Reflections on a PC earmarked for production usage. The operator would execute a canned Reflection procedure that would initiate an HP session, download the compensation file to a Drive A floppy disk on the PC and terminate the HP session. This floppy was handed to Compensation Unit personnel. Workers in that unit could then insert the floppy into Drive A prior to invocation of the DBase process that generated the compensation statements. No prompted input was necessary from the user.

The ultimate scenario is to design for a direct download into the PC of the end user, bypassing operations completely, as shown in the example on the following pages. In the example, the user invokes a Reflection Command File that will initiate an HP session, wait for user passwords, download the data to the correct hard disc directory on their PC, and terminate the HP session. If any errors occurred in the actual download, the user is given the opportunity to re-try the download.

Arguments can be made that good procedures would dictate that communications, specifically file transfers of a non ad hoc, fixed production nature, would best be handled and controlled by production operation. Additionally, stand-alone workstations do not have online host communication requirements other than for the download of limited scheduled production data. In such a case, a single sharable download workstation could serve sufficiently and align communications costs. The decision would vary based on the users requirements.

The Conclusion

The concept of a Data Download was new to this technical and end user community. The first installation has been a substantial success. The client has requested other downloads. Some financial reports are being considered. Overall, experience with the products mentioned and the concepts presented has been very favorable.

Footnotes

*1 Imacs Systems Corp., DataExpress Reference Manual, 11/87, p. 1-1.

*2 Walker Richer & Quinn, Inc., Reflection User Manual, 10/86, p. 3.

DataExpress Procedure and Explanation

Procedure Review Report

```
--------------------------------------------------------------------------------
ANNUDX(AJC.MTEST)            created on 87.09.25 by AJC          page   1 of 1
--------------------------------------------------------------------------------
OUTPUT FILE FORMAT: Report listing                          Type of output that
                                                               is to be created

OUTPUT FILE LAYOUT: Suppress detail records                 Display only total
                                                               lines when each
                                                               sort item changes
     ISSUE-DATE                          X(4)
     PROD-ID                             X(8)
     POLICY-STATUS                       X(2)
     subtotal(FACE)                     S9(11)
     subtotal(PC)                       S9(12)

SORTED BY:                                                  Items are to be
                                                               sorted in the
                                                               following order
     ISSUE-DATE                X(4)
     PROD-ID                   X(8)
     POLICY-STATUS             X(2)

SUMMARIZED BY:                                              What  fields  are
                                                               summarized.  As
                                                               each item changes
                                                               these fields will
                                                               contain totals

     ISSUE-DATE                X(4)  = subtotal(FACE)       Print the total FACE
                                                               for given date
                                       subtotal(PC)
                                       count(FACE)          Print the number of
                                                               records for given
                                                               date
                                       count(PC)
     PROD-ID                   X(8)  = subtotal(FACE)
                                       subtotal(PC)
                                       count(FACE)
                                       count(PC)
     POLICY-STATUS             X(2)  = count(FACE)
                                       count(PC)
```

```
SELECTED BY:                                                  To report only
                                                                 certain records,
                                                                 the user can use
                                                                 the following
     ISSUE-DATE                  X(4)                            choices for
 and PROD-ID                     X(8)                            selection. Enter
 and POLICY-STATUS               X(2)                            blanks to use
 and FACE                        9(10)                           all records
 and PC                          9(11)


FILE ACCESS PATH:

     ANNUOUT(type=MPE)                                        MPE input file
      1    filler                     X(4)                       and  layout.
      5    ISSUE-DATE                 X(4)
      9    filler                     X(3)
     12    PROD-ID                    X(8)
     20    filler                     X(1)
     21    POLICY-STATUS              X(2)
     23    FACE                       9(10)
     33    PC                         9(11)

--------------------------------------------------------------------------------
annuity summary
```

Reflection Command File Example and Explanation

```
;    THIS IS A COMMAND FILE USED TO DOWNLOAD DATA FROM
;    THE HP3000 TO THE IBM/PC.  THE HP FILE IS CALLED
;    M0078P06, WHILE THE IBM/PC FILE IS TISFILE.DB3
;    IN DIRECTORY DBASE\COMPnn, WHERE "nn" IS THE YEAR.
;
LET V1=0                                     Sets counter V1 to zero
:PROMPTER                                    Label name
TRANSMIT "^M"                                Transmit a Carraige Return
WAIT 0:0:2 FOR ":"                           Wait up to 2 seconds for the
                                                      colon prompt

IF NOT FOUND                                 Result determined by above WAIT
   LET V1=V1+1                               Increment V1 by 1
   IF V1>3                                   If V1 is 4 or above, then
      DISPLAY "^[&dA HOST NOT READY"             Display message on terminal,
                                                      in inverse video
      WAIT 0:0:5                                 Wait 5 seconds
      EXIT                                       Return to PC
   ENDIF
   GOTO PROMPTER                             Go to PROMPTER label
ENDIF
WAIT 0:0:2
TRANSMIT "HELLO DOUG.MTEST ^M"               Transmit MPE "HELLO" Command
HOLD 0:0:30 FOR "HP3000"                     Wait 30 seconds for system to
                                                      return Logon Banner
                                                  At this time the user goes
                                                      from Reflections Mode
                                                      to HP Mode, allowing
                                                      the user to enter the
                                                      password(s).
                                                  If the user has not entered
                                                      the password(s) within
                                                      the 30 seconds, HOLD
                                                      will automatically end
                                                      put the user back into
                                                      Reflections Mode, and
                                                      this command file will
                                                      continue.
IF NOT FOUND                                 If Logon Banner wasn't displayed
   TRANSMIT "^M"                                 Transmit 3 Carriage Returns
   WAIT 0:0:1                                         to exit HP Security,
   TRANSMIT "^M"                                      if it was active
   WAIT 0:0:1
   TRANSMIT "^M"
   WAIT 0:0:1
   DISPLAY "^[&A LOGON FAILED "                  Display message on terminal
   WAIT 0:0:10
   EXIT
ENDIF
WAIT 0:0:15 FOR "^Q"                         Wait for DC1 character to be
                                                 returned from host
```

```
:TRANSFER                                    Label name
CONTINUE                                     This will cause the command file
                                                 to continue whether an error
                                                 occurred or not

RECEIVE C:\DBASE\COMP88\TISFILE.DB3 FROM M0078P06.DATA ASCII DELETE

                                             Transfer file M0078P06.DATA from
                                                 the HP to the PC directory
                                                 DBASE\COMP88.  If the file
                                                 already exists, delete it.
IF ERROR                                     If there is an error in transfer

   DISPLAY " ERROR TRANSFERRING DATA TO PC.  TRY AGAIN (Y or N)? ^M^J"

                                             Display message on terminal,
                                                 with Carriage Return and
                                                 Line Feed
   ACCEPT V2 LIMIT 1                         Accept user response of 1 char.
   IF V2="Y"
      GOTO TRANSFER
   ENDIF
ELSE
   TRANSMIT "PURGE M0078O06.DATA^M"          Purge old M0078P06 file
   WAIT 0:0:5 FOR ":"
   TRANSMIT "RENAME M0078P06.DATA,M0078O06.DATA ^M"

                                             Rename datafile for storage
ENDIF
TRANSMIT "^M"
TRANSMIT "^M"
TRANSMIT "BYE ^M"
WAIT 0:0:5
EXIT
```

Integrated Information Engineering

Peter Ney
Richard Irwin Associates (RIA)
Postbus 15348
1001 MH Amsterdam, The Netherlands

Overview

Much industry attention is currently being focused on methods of creating production environments within Information Systems departments that effectively support corporate strategy. There is a powerful drive to improve the efficiency and quality of the IS function, as the growing cost of maintenance of systems that have been poorly designed, managed and constructed is appreciated. There is a pressing requirement for a comprehensive environment that supports the management, design, construction, documentation, change management and maintenance of informations systems. Clearly the data dictionary and its associated tools are seen as the core of such an environment.

The HP3000 community at the moment does not have access to truly integrated and user-friendly facilities that support such an environment. It is most likely that in the near future this area will be addressed with a release of a new range of products, but meanwhile there are techniques and methods that can be used to approach the ideal information engineering support environment reasonably close with existing tools and facilities.

This paper discusses what is currently a very dynamic area in the HP community, that of System Dictionary, its associated products, integration services and Computer Aided Software Engineering (CASE). Because of certain developments currently taking place in this area, I have decided not to submit the full paper for inclusion here, as some of its detail is very likely to be affected by these developments, thus effectively making the paper "out of date" by the time it is presented at the Conference.

The presentation will examine the strategy and methods leading to integrating data, functional and resource information on a corporate level using the System Dictionary on the HP3000.

The System Dictionary, when considered as the hub of the IS department, must be structured to effectively support the development and production functions. Currently System Dictionary has many critical problem areas. Some of these, such as the poor user interface, no access via 4GLs, lack of true version control or support for coexistance with Dictionary/V, can be "lived with" in the hope that these and other annoying shortfalls will be resolved by HP in the near future. There are some problem areas, however, that actively discourage the use of System Dictionary to successfully support the IS environment, and these must be overcome by the user in the short term if effective use of the product can be currently made.

One of the main problem areas is the current open-endedness of the System Dictionary coreset that does not encourage a sophisticated use of the product. There is, for example, no standard provision for data and functional analysis (eg. EAR models and Data Flow Diagrams), and users have to independently customise the coreset to support this area. Not surprisingly, many choose not to bother.

Partly linked to this is the current lack of a standard interface to CASE tools, the use of which is spreading in the HP3000 community.

In the presentation I hope to discuss ways of coping with the available tools and attempt to create a corporate integrated information engineering environment in order to give effective support to today's IS department.

Data Center Management and Efficiency

Betsy Leight
Operations Control Systems
560 San Antonio Road
Palo Alto, California

INTRODUCTION

The efficiency and security of data processing operations is a major concern to corporate data processing managers.

To assess their data center's performance, management often conducts a review of the following areas: standards and procedures, operational work flow and control, scheduling, data security and access control, equipment utilization, and environment.

If the data center management and staff understand the concerns of upper management and the information that is needed, the operations review can proceed more smoothly, and the results can be more beneficial to the entire organization.

Corporate direction for data center management focuses on control issues, automation procedures and the gray area in between. The data center manager reviews the operations and interprets corporate objectives to the operations staff. Because this article focuses on the concerns often overlooked by the data center manager, it can be a useful check list for improving daily operations as well as preparing for a data center operations review.

STANDARDS AND PROCEDURES

Data center managers should verify that standards and procedures exist and are enforced. These written rules are the controls; they should include:

- Ensuring proper timing in running programs and jobstreams.
- Inserting changes into production runs; entering run dates.
- Using correct data for programs; accesses the correct data files.
- Protecting data and programs from accidental or intentional destruction.
- Specifying methods of physically moving input and output.
- Scheduling work and getting work rerun in the event of an error.
- Keeping records of work performed and session logons.
- Determining and recording sufficient resources for the work.
- Performing maintenance and general housekeeping associated with the operation of the data center.

The data center manager should ensure that formal standards exist for systems development and maintenance, program and system testing, file conversion, program and system change control, library operations, computer operations and documentation.

For each aspect of standards and procedures, your installation can implement procedures using automation software as shown in the following chart.

Task Description	Controlled Operation	Automated Controls	Controlled Automation	Automated Operation
Run Programs	User Task	User with Scheduler	User who Schedules	Scheduling Software
Change Jobstreams	User Task	User with Editor	User who Edits Jobs	Job Change Software
Verify Data File	User Task	User with File Scan	User who Scans Files	File Scan Software
Monitor Jobs	User Task	User with Jobstream Monitor	User who Monitors Jobstreams	Jobstream Monitor Software
Control Master Files	User Task	User with File Copy	User who Copies files	File XFER Software

OPERATIONAL WORK FLOW AND CONTROLS

The data center manager should investigate specific items in this area, including whether:

* Input data from other departments is complete and entered on time.

* The data center keeps job accounting and session logon information.

* Job accounting information is evaluated and used by management.

Error control procedures should also be reviewed. Specific questions to ask include:

* Is anyone notified in case of a production processing error?

* Are batch processing errors and logon violations documented?

* Are error statistics accumulated or ignored?

* Are errors followed up on so that they do not recur?

The data center manager should also confirm that downtime is reported and statistics compiled. A log of late reports and jobs should be maintained.

There should be a formal communications channel between data center operations and other departments; operational tips and other advice should be passed to all operators.

All problems encountered at the computer, as well as any action taken to prevent their recurrence, must be documented. Operators must also receive feedback on reported problems.

The data center manager scrutinizes output report distribution and disposal and determines whether:

- All reports have been distributed to the proper user departments.

- Procedures have been established to control the distribution of sensitive output.

- Procedures exist for disposing of confidential reports when they are no longer required.

Finally, the data center manager should ensure that jobstream run instructions are kept up to date.

SCHEDULING

Efficient and effective scheduling is extremely important in providing a high level of reliability and predictability to data center operations. The data processing manager should determine whether:

- Daily processing activities are scheduled and a daily contingency schedule is maintained.

- Actual run times are recorded for batch programs and jobstreams.

- This data is used to calculate expected run times for a given day.

- Expected run times are compared with actual execution time to ensure that processes have not terminated abnormally.

- Unscheduled runs are supported by a work request or other written authorization. Schedule deviations should be documented and followed up on by a supervisor.

- User-submitted jobs are recorded to allow forecasting of future schedules, resource requirements, and special processing considerations.

Scheduling software enforces controls and totally automates the data center operations. Manpower reductions can result depending on implementation. A chart on scheduling appears below:

Task Description	Controlled Operations	Automated Controls	Controlled Automation	Automated Operation
Create Schedule	User Task	User with Streamer	User who Schedules	Scheduler Software
Monitor Run Times	User Task	User with Job Log	User who Logs Jobs	Monitor Software
Job History Report	User Task	User with Job Log Data Base	User who Compares Job Logs	Data Base Report Generator

DATA SECURITY AND ACCESS CONTROL

Data base and master file information should be protected from unauthorized access or loss. Employees must be instructed about their responsibilities concerning confidential information. Management should periodically review and update controls and security provisions relating to data.

Live production programs should be physically separated from development. The staff should be prohibited from running test programs against live files, and operations personnel should be denied access to sensitive data files.

Secured file management is not limited to source and object control. Data center managers should ensure that procedures have been established for:

- Accepting and transferring programs from development to production.

- Program library changes are formally approved.

- Acceptance testing of changed programs before transference to the production libraries.

- Updating production documentation after changes.

To maintain security, operators should be prohibited from renaming or transferring programs without supervisory approval. Internal labels must be used from all data and program files.

Passwords and lockwords should be used to protect accounts, users, data files and port access. Passwords, lockwords, dates and constants should be introduced at run time, eliminating the need to hard-code sensitive data in jobstreams.

Data security and access control software can bring automation to the data center; with automation you can expect a more efficient system operation as summarized in the chart below:

Task Description	Automated Controls	Controlled Automation	Automated Operation
Separate development and production areas	User who moves files	User with file mover	File Librarian Software
Restrict live file access	User who locks files	User with lockwords	Protected File sets to User Sets
Approval pre-step to Production Move	User who moves files	User with file mover	Automated Fila move after Approval
Project/memo notes Related to Changes	User who completes forms	User with text editor software	Online dialogue requesting memo text at save time

EQUIPMENT UTILIZATION AND EFFICIENCY

Once it has been determined that the entire data processing department is following a properly implemented set of standards and procedures, the data center manager should review equipment utilization.

The data center manager should collect raw data from the system log files in order to report the following information:

* How much machine time is spent on reruns?

* Whether reruns are analyzed?

* Whether certain jobs are especially susceptible to reruns?

With reported resource utilization information, the data center manager should check that the full multiprogramming capability of the system is being used. It then follows that multiple jobstreams should run concurrently, if there are no data file bottlenecks.

The data center manager then reviews whether many jobs can be restarted without rerunning the entire job. Jobstep tracking and restart software should be implemented for efficient data center operations.

ENVIRONMENT

The data center manager should review the work space to ensure that it is adequate for the number of employees. The environment should be neat, and supplies should be easy to locate.

Auxiliary items located outside the computer room, such as bursters and de-collators, should be accessible for the flow of work in the department. Tapes, discs and other storage media should be stored in a closed, fire-protected, limited-access area.

RECOMMENDED COURSE OF ACTION

The data center manager should make the organization aware that the following steps can enhance the operations review:

* Providing the data center management with as much information as possible.

* Implementing software systems that leave clearly defined audit trails.

* Keeping accurate records, log files and file history information.

* Maintaining formal written standards and procedures.

* Implementing an effective data security system and access control facility.

Following the data center manager's recommendations and procedures for operations can yield an efficient, secure and automated data center.

An HP 3000 Approach to IBM's LIBRARIAN Techniques

Betsy Leight
OPERATIONS CONTROL SYSTEMS
560 San Antonio Road
Palo Alto, CA

As HP data center become more sophisticated, users are attempting to introduce more standards and controls into their environments. During the early stages of growth, the need for controls resulted in automated batch processing, access restrictions, menu drivers, and non-user scheduling. In today's environment more sophisticated concepts such as file management and accountability are beginning to come under scrutiny. File management concepts are not new. In fact, IBM mainframe users have been controlling their development files for years with the aid of PANVALET and ADR/LIBRARIAN. These products separate test and production files, control source code libraries, archive modules, and perform audit functions. Many HP users now recognize the benefit to be derived from these techniques.

In this paper I will discuss the problems inherent in current control techniques and describe possible solution pathways, however, it is essential to establish unequivocally that the current approach to development tracking is inadequate. Whether it is recognized as such or not, one of the major functions of any data processing department is the creation and maintenance of software. This is not a process confined to development houses alone. Every day, any MIS department could receive requests to modify software and data. The efficiency and cost of development are instrumental to the corporation's success because computer applications have become an integral factor in the competitive struggle for market position. The software alone can represent corporate assets valued at hundreds of thousands or even millions of dollars.

Surprisingly, there are many environments that have no source access or change restrictions. Programmers can access production source directly. This approach should never be allowed, for two reasons: First, the original source code can be destroyed. If a production error results from programmatic changes, the original version is not easily restored. Therefore, production can be delayed for several hours, or in the worst case, for days. Second, there is no audit track. Programmers are not restricted from making unauthorized modifications directly to production files. Doesn't this possibility worry you?

Even in cases where some restrictions exist, other problems can arise.

Current development procedures often result in a discrepancy between source and object code. In other words, the master source file does not recompile into the production object file. Since it is extremely difficult to recreate source code from a load module, there is no way to ascertain that all the object code features exist in the source. Should the source require a change, there is no guarantee the resultant object will have the same functionality as the original load module. Production can and does often fail as a result.

Another common scenario involves multiple programmers who make change simultaneously to the same source module. In this case, the last programmer to update production wins because previous fixes are obliterated. The net result is wasted effort and invalid object code.

Although these and other problems are obvious to many, a dichotomy begins to appear when it is discovered that many HP 300 centers continue to operate in a reactive mode without introducing standards. Direct user support is considered the primary function of development and operations. This translates to a daily goal centered on processing user-requested jobs, completing batch production, and distributing reports. Although these functions are routine tasks of operations, one should not overlook the basic fact that the efficiency and accuracy of operations' output rely upon the cohesiveness of the software components. Just as a solid house cannot be built upon shaky foundations, reliable computer processing cannot be achieved unless the associated code shows internal integrity.

The component integrity problem is one HP 3000 centers are not managing effectively. Instead, the issue is currently relegated to the "wish list" and usually escapes notice. This policy is not a solution. Although operations can proceed error-free for months, inevitably a simple oversight snowballs into a catastrophe.

The important point is that software development and file maintenance procedures directly affect operations. Should programs fail, run out of order, or produce invalid data, the operations department will be required to rerun procedures. To avoid erratic production problems, file management issues need to graduate from the "wish list" to the "current projects list"

It would be inaccurate to give the impression that HP users are not addressing this problem at all. Some are, but the degree and depth of resolution vary widely. Let's examine some checkpoints along this range.

Various Approaches

The simplest procedure is no procedure. In other words, programmers merely log directly into the production account, modify the code, test it, and recompile. All three steps occur within the production location. As mentioned earlier, this method has no safeguards and is extremely risky.

A slightly more sophisticated approach requires programmers to FCOPY or CHECKOUT source code from the production account and move it into a development location. Unless stringent controls exist, there is no guarantee that only one individual has checked out a particular module. Furthermore, programmers are not restricted from accessing production accounts, nor is there any way to audit their access.

The production-to-development strategy just described can be envisioned at three levels. At the lowest level, the development area may be nothing more than an amalgamation of programmer groups. In this case, each programmer copies, develops, and tests in his own group. There is no standardized testing environment. A second level maintains a development account that duplicates the production account. Thus, each programmer can be confident that alpha testing is occurring in an environment that closely resembles production with accurate, up-to-date object and load modules.

Ideally, production and development accounts should exist on separate CPUs to eliminate completely the possibility of direct access to master files. Of course, this approach is not always possible and separate accounts on one CPU will produce adequate results.

Once the development effort has been completed, what happens to the code? My experience suggests the same developer simply overlays the original production files with the enhanced code. Such alpha testing does not represent adequate control because the programmers who test their own work can easily overlook bugs. Besides, they tend to test what works rather than attempting tests to "break the code". For this reason, it is strongly suggested that a Quality Assurance (QA) process be initiated.

There are several ways to initiate such a procedure, depending upon company size and resources. Smaller companies may have alternate programmers QA test their colleagues' development efforts. The retests are usually performed in the development account. Larger organizations may hire an individual or staff whose sole function is QA testing. When the process develops to this extent, there is generally a separate QA test account that reflects both the production and development environments.

Should a QA process exist, it is vital that the developer relinquish all claim to the code when it moves to the QA phase. Only one copy of the developing code should reside in either the development or QA area. If both accounts contain separate copies, undocumented changes to the development files may not be incorporated in the QA version. Thus, the final production version would not include all code changes. This safeguard is commonly overlooked. Similarly, if the QA analyst locates a discrepancy in the modules tested, the code should be returned to the original developer for revision. It is now QA's turn to relinquish all claim to the code until it is returned by the programmer.

At the conclusion of QA analysis, another step can exist. A higher level manager should perform a final approval on the development effort to ensure all checkpoints and tests have been satisfactorily completed.

Followed final approval, the enhanced code is ready to be moved into the production location. An FCOPY or move will overlay the original modules. The destruction of the earlier version could be detrimental if the revised code contained errors and no backup copy of the original files existed. Therefore, the original modules must be stored to tape prior to enhancement installation.

The update step can be both time-consuming and error prone, especially when large numbers of files are involved. To make matters worse, a compile step and JCL update must also be coordinated. Standards cannot be eliminated at this final stage. If they are, production may be under old JCL or inaccurate object code resulting in production that aborts in the middle of the night. Sound familiar?

When they exist, the aforementioned procedures are usually tracked on paper with a form. An originating service request often moves with the code from production to the programmer to the QA analyst. Each step along the process is signed off on the form. Such a tracking method is inadequate for several reasons. Most simply, the paper can be lost or misplaced, or any member of the chain can fail to record his involvement. Most importantly, there is no assurance that the form really reflects what has occurred.

Individuals have been known to misrepresent information for a variety of reasons -- often in the name of speed. A manual paper tracking method can be synonymous with no tracking method. An appropriate, complete, and accurate tracking procedure should be a primary concern to the development and audit staff.

Obviously, development solutions are very flexible. They can evolve through a number of steps and can encompass varying levels at each step.

An idealized strategy

It is possible at this point to extract an idealized development strategy based on the previous discussion. Needless to say each environment will require additions to or deletions from this ideal. However, the following does describe a general goal based on my own experience.

In this idealized scenario, three accounts exist: a production or master, a development, and a test account. Each account structure is a carbon copy of the others to the extent that files moved from location to location retain their original jobname and group designators. Only the account names change. In this way, it is easy to visualize the link between a developing program and its originating master.

When files move between the production location and the development area, a copy should be made. The original source should never be destroyed. However, movement between development and test should result in only one copy at either location.

Once QA has approved all changes, a project leader or manager should verify that adequate and accurate test procedures have been followed. Only at this point should code be moved into the production library. Such updates could occur once a day if desired and should be performed by operations or a production librarian. The latter is my recommendation as it restricts the responsibility, control, and audit functions to one individual.

Prior to update, the original production should be verified and stored. Without this step it is much more difficult to return to a prior version in the case of error.

Software tools that perform file tracking and auditing procedures automatically without information loss are available. The tools also force participants to conform to structured rules to ensure steps are performed in sequence along the development pathway.

In order to implement a structured development strategy with some components of the idealized route in your environment, it is vital to define goals. Possible goals include but are not limited to:

1. Establishing controlled access of production modules.

2. Creating a set of rules to minimize file transfer and maximize efficiency.

3. Ensuring a link between compatible object and source code.

4. Verifying all associated files, such as JCL and databases, are saved and moved to production concurrent with source and object updates.

5. Developing a methodology to track file movements accurately.

Development strategies, an idealized solution, and the goals to consider in achieving the ultimate solution have been described. To configure your site along the ideal path, four steps are necessary. First, identify and flowchart the specific attributes of your best solution to software development based on your needs. Second, assess the components of the development strategy that are currently employed. Third, compare the current structure to your idealized goal and prioritize change requirements. Fourth, develop a project plan from the priority list and implement the necessary changes.

Unfortunately, it is not possible to provide flowcharting assistance in this article. Each development effort is unique. However, the scenarios previously described should provide hints and suggestions for the first step.

Several areas of concern can be identified during the evaluation process of current strategies. Questions that should be asked during this analysis are included here.

After these data are collated, it is possible to perform a comparison between current methodologies and the site specific optimal development path. This third or comparative step is, once again, subjective. Each individual must determine for himself where the current procedure diverges from the idealized goals.

Prioritization of differences is dependent on site-defined needs. For example, the auditors may be clamoring for file movement control. Thus the implementation of tracking procedures would be the primary concern. On the other hand, QA testing can be the vital link. In either case, auditors can prove to be a wonderful resource in this evaluation process. The comparison step is often the easiest in the four-step strategy. It draws results from the analysis required to accomplish the previous steps.

analysis required to accomplish the previous steps.

Following prioritization, implement a project plan to attain the stated goals. Generalized solutions should have evolved through the process of current assessment, goal derivation, and priority setting.

To summarize, the need to control and track the software development cycle in becoming increasingly apparent. Without standardized controls it is virtually impossible to establish the validity of software modifications. This is most important in larger environments, especially those that manage enhancements for remote locations.

Therefore,it is important to move toward an idealized software development process. This goal can be accomplished by comparing the optimal solution to present controls and implementing plans to minimize strategy gaps. Manual tracking procedures can be used for this purpose, if necessary. Fortunately, the industry also offers software tools for an automatic solution.

Evaluating Software Development

1. FILE LOCATION: Where do the production files reside? Are all files contained in one account? Several accounts?
2. DEVELOPMENT ACCOUNTS: How are the development areas configured? Does each programmer maintain a unique development group or does all development occur within the same group?
3. ACCESS RULES: What rules are implemented to control file movement between production, development, and testing? Who has access to these locations? When does access occur? Are there any preconditions, such as management approval?
4. ACCOUNT STRUCTURE: What is the structure of the production, development, and test accounts? Are they duplicates of one another?
5. VISIBILITY: How much visibility of file movement exists? Can only one programmer gain access to a file at any time? Can all changes made by each programmer be verified in the content of the final code? Does a validation check for compatibility of source and object occur?
6. QA ANALYSIS: Is there an established need for separate Quality Assurance testing? Does the current development effort include QA testing? Are there plans to move in that direction? Where will QA testing occur? Does the account structure duplicate production and/or development accounts?
7. FINAL APPROVAL: Does tested code pass through a final checkpoint prior to re-entrance into the production account? Who is responsible for this final check?
8. UPDATE STEP: Who is responsible for moving tested code into the production account? At what time(s) does this occur? How much error exists within the current strategy? What can be done to reduce inaccuracies at this step?
9. VERSION CONTROL: Are original production files by newly modified code? Where do copies of old versions reside? Are all versions verified and tracked? What type of recovery procedure is available if new code fails? How does this occur? How complex is the recovery?
10. AUDIT TRACKING: How are file movements tracked? Is there visibility of when, why, how, and by whom files are moved? Can inadvertent purges be identified?

Foundation for HP Data Security

INTRODUCTION

Data security is an issue for any organization relying on HP3000 computers. Assets such as operating systems, applications and data files exist irrespective of an entity's size or purpose. Although the degree of any given file's sensitivity and recoverability varies, an expense is attached to any data that must be recreated.

These statements are as applicable to relatively small HP3000 users as they are in State Farm's case. However, the degree of applicability may be greater for State Farm, as it develops a network of hundreds of HP3000s to support and perform the sensitive task of insurance claims processing. The fact that application and network development were underway several years prior to any coordinated data security effort further complicates matters.

Consequently, substantial effort has been expended over the past three years of our HP data security program to "catch up and keep up". While I would not claim that State Farm has the ultimate HP security strategy for every other organization, our approach's effectiveness is due to a foundation of components that I feel should be considered by all other programs. Hopefully, the HP data security directions and experiences that follow will lend benefit to your company's computer security program.

BASIC GOAL IDENTIFICATION AND STRATEGY

Any project or task addressed by State Farm's HP data security program is undertaken to further at least one of the following two goals:

1. A user's system access and activity should be uniquely identified.

2. A user's computer, application and file access should be limited to only those resources necessary for satisfactory job performance.

Transforming these statements from philosophy to HP security practice in a large HP operation can be a time consuming, frustrating, misunderstood and political process. While developing an HP3000 data security program can sometimes be an unnerving experience, I have found that its negative possibilities can be greatly diminished through coordination of the following components:

1. Security administrator education
2. Exposure identification
3. Management backing
4. User awareness
5. System access control
6. File access control

The presented order of these six points is deliberate. As much as is practical, for example, I feel that security administrator education should precede all other aspects of the program. As a further illustration, I believe that implementation of any computer-based access control must follow management approval and user awareness. With the advance understanding that none of these six security building blocks can stand on its own, the following sections explore each of them in greater depth.

SECURITY ADMINISTRATOR EDUCATION

State Farm was almost exclusively an IBM shop for many years prior to the entrance of HP3000s onto its scene. Consequently, like many analysts "on the HP side" today, my background was that of an IBM application programmer. I had much to learn about MPE and State Farm's usage of HP3000 systems before I would be competent to lead any effort to improve my company's HP data security program.

MPE Education

My formal HP educational background consists of the "Programmer's Introduction" and "System Manager" classes. The former served as a satisfactory primer on topics such as system access commands, UDCs and file access security matrices. Roughly half of System Manager focuses directly or indirectly on security issues such as file access control and the power of PM and SM capabilities.

Informally, I have tried to tone my HP security awareness through such references as the Systems Operation and Resource Management Reference Manual. This resource notes the functions of the various MPE capabilities and details the purpose and usage of relevant MPE commands (like NEWACCT).

HP's Communicator manuals, published with new releases of the operating system, can also be an excellent source of data security-related information. One of the most helpful MPE enhancements in our data security effort was disclosed in the UB-Delta-1 Communicator. It stated that U-MIT would allow an SM-capability userid to perform all userid and group maintenance, discontinuing the security administrator's prior need to access the systems via hundreds of AM-capability userids assigned to the various accounts.

Data security-related articles in periodicals such as INTERACT and The HP Chronicle have also presented a wide range of facts and commentary. I'd like to close by noting one of the newest MPE educational tools: the MPE/XL Account Structure and Security Reference Manual. While identifiable with MPE/XL, the data security practitioner should find nearly all of its content applicable to MPE and nicely encapsulated in this functionally specific resource.

Organizational Education

Effective HP data security administration required that my operating system education be coupled with a familiarization of State Farm's HP3000 usage. This process began with orientations conducted by other HP-related areas, committee work and informal conversations. Through these experiences, I began to generally understand our HP program's strengths and weaknesses, and what area's were responsible for the various support functions (teleprocessing, system performance, application development, etc.).

The organizational understanding attained through these experiences has proven to be a key factor in reducing State Farm's HP system exposures. For example, several areas have agreed to relinquish their PM and SM capability assignments, alternatively allowing analysts in the system management area to perform sensitive tasks (e.g., file updates in the SYS account) for them.

What follows is a sampling of data security-related topics relative to State Farm's HP3000 usage. While the associated responses are unique to my environment, I invite you to consider what your answers might be. The point of this exercise is to develop a "big picture" of the challenges that your data security program faces:

What primary business purpose(s) does my company's HP3000s serve?

The automated processing of insurance claims.

What is my company's policy on PM/SM capability availability?

Availability should be as limited as practical. This policy also applies to PM-prepped program development. The negative potentiality of PM and SM on system security is too severe to tolerate passive assignment of these capabilities.

Have other areas been designated to assist in the HP security administrative effort?

1. Each of State Farm's twenty-five regions has at least one data security administrator. However, his/her responsibilities also span to the IBM systems. In addition, these administrators currently possess minimal HP security background and few software tools to effectively maintain a security program.

2. Users are responsible for the security of their userid(s). However, many users currently lack the HP security education and password assignments necessary to protect their userids to even a minimal degree.

Where are my company's HP3000s located?

State Farm's HP systems are located in restricted areas of its corporate headquarters, regional offices and larger claims service centers.

What are my company's most sensitive HP3000 files?

1. Claims-related databases

2. Operating systems

3. Teleprocessing systems

4. Electronic mail systems

What is my company's HP userid policy?

1. Userids should be unique in the session name- or user-level qualifier and be identifiable within a given individual or process.

2. Userids should be password protected.

3. A user is responsible for all system activity via his/her userid.

4. For security reasons (ironically), most users lack the AM and/or SM capability assignment necessary to maintain their own MPE password(s).

Data Security Education: A Closing Thought

A final point about HP data security education: the process never ends. I found that it is all too easy to formulate security policy based on an educational plateau, failing to invest time to additional research that may result in reevaluation of current practices. For example, PS (Programmatic Sessions) capability's availability was restricted until further investigation revealed that it did not grant its user carte blanche access to the systems through the userids of others. I feel that the data security function owes a duty to its organization to reexamine its policies in light of improved understanding and changing conditions.

MANAGEMENT BACKING

Just as many of State Farm's HP analysts have IBM "roots", so too does our data processing management structure. In many instances, DP management is responsible for pursuits in both the HP and IBM environments. Even in those areas totally committed to HP3000-related development, managers are often too busy with their areas' respective responsibilities to devote much if any time to data security issues.

Given this scenario, it became clear at an early stage that the success of State Farm's HP data security program was very dependent upon management's appreciation of the issues. I have delivered the message in various forms. For example, an article detailing the data security ramifications of PM and SM capabilities was addressed to all first-line DP managers with HP-related responsibilities. I have discussed the power and public nature of the MANAGER.SYS userid with the manager responsible for operating system integrity. (He now maintains his own list of authorized MANAGER.SYS users.) In varying degrees, several managers became involved as their areas relinquished their PM and/or SM capability assignments. A memo briefed a data processing vice president on the exposure to our regional office HP3000 exposure caused by Corporate analyst access to our X.25 network-connected Corporate gateway computer. Admittedly in varying degrees, presenting management with issues such as these as resulted in endorsement of our HP data security program.

Last mentioned but far from least important is the backing of my data security manager. Of all of State Farm's DP management, he has unquestionably been the most important individual for me to brief on HP security developments. This practice has not only helped him promote HP data security at his organizational level, but it has also enabled him to more effectively critique my ideas.

As I have stated, State Farm's HP data security program has had a lot of "catching up" to do. Sensitive capability assignments and obsolete userids have been removed, file access has become more restricted, etc. Management education and backing has greatly facilitated these sometimes delicate processes.

USER AWARENESS

I have seen examples of impressive data security awareness pamphlets, videos, etc. While I soon hope to pursue these more structured user awareness techniques (e.g., HP data security seminars for trainees and for on-board personnel), our area's priorities and staffing have dictated more informal approaches to date. Examples of these follow.

Committee Work

Committees can offer an excellent opportunity to express security-related opinions and suggestions, often at the assigned task's ground level. Committee charges in my environment have included dial-up procedures for non-State Farm users, userid implementation on new systems and HPDESK password procedures for regional office users. Analysts from a spectrum of other functions are exposed to security concepts in this manner. Committees are also an effective vehicle when the primary topic is HP3000 data security (e.g., security software evaluation, procurement/development and implementation). This latter case has rendered the added benefit of allowing others to participate in and understand State Farm's HP security direction.

Security Articles

This approach can make HP security a much less bitter pill to swallow. In a totally non-confrontational manner, users can learn more about their HP3000 environment and its security function. I have written articles on the security implications of PM/SM/OP/AM capabilities, group passwords and accounts without userids. A future topic is the comparison of our third-party system access security package with MPE. I would also like to explore the benefits and drawbacks of released files in an educational article.

Implementation Announcements

Third-party system access password implementations are in progress for State Farm's Corporate users. Rather than simply activating these passwords, I mail explanatory memos to the affected analysts a couple of weeks in advance. The notice details how the password will be implemented (e.g., on user-level qualifiers, (the "MANAGER" in MANAGER.SYS) or on specific session names of a user-level qualifier), illustrates what the new system access process will look like and explains how to change the password value. This vehicle has allowed users to become more aware of their HP system security responsibility without becoming confused and irritated with new procedures.

Informal Conversations

One-on-one telephone conversations or break area discussions can facilitate user awareness. This vehicle is more personal than memos, and it may be more appropriate than the committee setting for ad hoc HP security issues. In addition, sensitive issues can be dealt with in confidence. As an example, I have found this approach very useful when persuading users to relinquish their assignment of sensitive system capabilities. Rather than risking user embarrassment and/or resentment via the committee or memo approach, many sensitive capability assignment have quietly been eliminated.

Closing Comments

As much as practical, you want a supportive user base for your HP3000 data security program. Even with management's backing, your efforts will only result in a lukewarm level of effectiveness if your users are indifferent or opposed to them. Finally, it is humanly and programmatically impossible for me to notice every HP security shortcoming in a network as extensive and dynamic as State Farm's. I (and probably you) need to draw on users' expertise to flag exposures missed by standard security procedures.

SYSTEM ACCESS CONTROL

HP3000 system access control at State Farm rests on a developing foundation of security administrator experience, management backing and user awareness. These three factors coalesced in meetings of representatives from the various HP areas. In the early stages of discussion ,it became clear that State Farm's usage of HP3000s necessitated a software solution beyond the access security capabilities of MPE.

I'll begin by examining MPE's system access security shortcomings relative to State Farm's needs. My company supports a system access security policy of centralized creation, modification and deletion of userids, but decentralized password maintenance responsibility. Unfortunately, MPE requires that the userid and password functions either both be centralized or decentralized. In other words, AM and SM capability assignments may be severely restricted, with the controlling area responsible for userids and passwords. Alternatively, these capabilities may be widely available, with the various areas able to attend to their own userids and passwords. (In the latter case, however, HP3000 access security is based on the honor system at best.) Additionally, State Farm promotes a standard of user identifiability for all HP3000 access. For userids like MANAGER.SYS, system access uniqueness must be derived from the session name-level qualifier (the "KELLY" in KELLY,MANAGER.SYS). Unfortunately, MPE cannot require usage of the session name qualifier. Finally, in the absence of HP's Security Monitor product, MPE passwords are unencrypted. AM and SM capabilities can be abused to disclose the MPE system access passwords of others. Once again, this MPE feature's effectiveness is reliant on the very restricted availability of AM and SM capabilities.

To address these system access security drawbacks, State Farm recommended a third-party vendor product to enhance MPE. It disallows system access to unauthorized userids. While centralizing userid administration, the package decentralizes password maintenance. Users may be allowed to change their own vendor and MPE passwords, regardless of AM/SM capability assignment. The software also may enforce the supply of specific session name qualifiers at system access. Vendor passwords are encrypted and may be assigned to both the "user.account" and "session name,user.account" formats. Finally, security administration is not conducted through the MANAGER.SYS userid. Therefore, the userid upon which a system's access security is based need not be shared with areas responsible for other system management functions.

Simply implementing unique userids and encrypted passwords upon an HP3000 network like State Farm's is a sizeable task (well over 10,000 userid profiles are administered at Corporate Headquarters alone). However, the chosen access security product also provides for future "fine tuning" with features such as time-of-day restrictions, port restrictions and userid deactivation. These options, plus expanded usage of the product's reporting capability, will continue to strengthen the system access control component of State Farm's HP3000 data security foundation.

FILE ACCESS CONTROL

At State Farm, the HP data security administrator's job doesn't stop at the system gate. Whether uniquely identified or not, no HP3000 user is authorized to access all MPE files in all modes. System managers are not supposed to be reading the electronic mail of others. Programmers have no authority to recompile vendor code. No State Farm user is to be using TELESUP files from the TELESUP account. (We have established a separate account loaded with authorized TELESUP files for analyst use.)

Security administrator education in the area of MPE file access control is very important at this point. File access is determined by a matrix of account-, group- and file-level rules. Examining a single array of the matrix is most often misleading. For example, a particular file's file- and group-level access arrays may specify that any system user can take any action with that file, but the account-level array may limit some or all modes of the file's access to users logged into the file's account.

Security administrator education in the area of file access control must also extend to MPE's limitations. Matrix research via MPE is limited to the LISTSEC command in MPE's LISTDIR5.PUB.SYS (or "LISTF filename,4" in MPE/XL). On HP3000s like State Farm's, possessing millions of MPE files, an effective matrix evaluation via operating system tools would be impossible. Even a regular file examination of key accounts like SUPPORT, SYS and TELESUP would require a prohibitive time investment. Yet, with every system user able to identify every file via LISTF @.@.@, such regular, thorough access evaluations should be conducted.

Once again, State Farm has chosen a third-party vendor product to enhance its HP3000 file access security program. The software is used, for example, to flag released files. Another application may be identification of those PM-prepped program files with system-wide WRITE access. Scheduling these reports to be generated on a regular basis further strengthens this component of State Farm's HP data security program.

But what about identification of "mysterious" file creation, modification or deletion? We have a program that uses the system log to remedy these situations. It summarizes userid file activity or file usage regardless of userid for a given day or week. Hopefully, the aforementioned efforts to appropriately limit system and file access will diminish the need to invoke this utility.

CONCLUSION

From a very humble beginning, State Farm's HP3000 data security program has, by necessity, progressed rapidly to uniquely identify system access and practically limit file access. My hope for our program is that it is recognized within our organization as a partner in State Farm's insurance claims processing effort. To be an effective participant, however, our function must be based on a sound and growing program of administrative competence, exposure identification, management support, user awareness, system access control and file access control. While a sizeable HP operation like State Farm must address these components with large user solutions, I feel that the same data security foundation can be developed by organizations of all sizes to foster an effective and respected program.

Where's the Space

Joe Berry
Pekin Memorial Hospital
Court and Fourteenth Streets
Pekin, IL 61554-5098

INTRODUCTION

The management of space is crucial to any system. Most of us have had to learn about space management on the run and when we were short of disc space.

It is not difficult to manage free space. You can even delegate if your procedures provide good written instructions. A few minutes daily is all it takes.

I am not planning to unveil great new ideas. What I am writing about is methods that I have used that eliminate the "surprises". The key is to plan ahead and have your policies in writing.

This paper brings together many ideas learned from a wide array of sources. Hopefully it will help everyone eliminate hearing "where's the space?"

GLOBAL SPACE MANAGEMENT

Space management includes three equally important components that must be monitored on a regular schedule. These three components are:

1. The management of free space on all disc drives.
2. The management of image data set capacities.
3. The management of KSAM and MPE file capacities.

Failure to monitor these components on a regular schedule can lead to a variety of system problems. These "surprises" can vary in severity from annoying to devastating. Some examples of the problems caused by the failure to monitor system space regularly are:

1. Slow down of response time.
2. Programs abort due to insufficient sort space.
3. Your month end update program aborts because a crucial data set filled up.
4. MPE shuts the spooling system down.

There are many ways to manage each component. The only incorrect way is failure to monitor each component regularly.

The single most important step is to plan ahead and develop policies and procedures. It is much easier to evaluate all of the ways to manage space when you have both time and space available.

A good space management program contains written policies that are distributed to all users. The policies should be backed by upper management. They should be reviewed and revised as necessary on at least an annual basis. This will bolster user relations by preventing misunderstandings and by enforcing the management of space consistently across the entire user base.

Internal step by step procedures that detail all aspects of space management should include the "how to" to enforce the policies. The procedures will guide your staff on the proper steps of effective space management. These are procedures that cannot be left on a shelf to gather dust. No matter how much free space you have now it will mysteriously disappear in a relatively short time. Then everyone will yell "Where's the Space?" and point fingers at the system manager. Be prepared by planning ahead and following all steps of your space management program regularly.

It would be ideal to have pre-determined levels of free space that can are considered "urgent" and "critical". This would make everyone aware of the levels to watch for in numbers of sectors. It can even go deeper and identify levels by device. This would help point out the need to move files from one device to another.

One of the building blocks to easier space management is to implement and enforce a policy of naming conventions. All accounts, groups, and files should be named according to a master plan. Be sure that you can identify "test" data and "one time" programs and files. INTEREX has simple but effective naming conventions for the CSL file.

An archiving policy needs to be developed. Each group of files needs to have a plan for archiving that includes the type of files, the time frame, and the medium used. An internal step by step procedure also needs to be developed and followed to enforce the policy.

MANAGING DISC FREE SPACE

MPE allows the system manager to limit the amount of space that each user and each account can use. The NEWACCT, NEWUSER, ALTACCT, and ALTUSER commands have a parameter that limits the number of sectors allowed for each user and account. This is the most effective way of planning for future needs as well as managing current space. It is very simple to implement. It works well in a stable environment; however, in a fast paced environment it could cause more problems than it solves. If used, the MPE security must be enforced so that only the proper people have access to the NEWACCT, NEWUSER, ALTACCT, and ALTUSER commands.

There are many tools available for managing disc free space. The contributed library has several, third party software vendors have several and MPE commands and utilities round out the vast array of tools available.

There are simple and painless tasks using contributed library programs that conserve free space (and speed backup). You can SQUISH source code files (or any other files) using the program SQUISHER from the TECH account of the contributed library. This will save about fifty percent of the space. Large MPE or KSAM files can be re-blocked to a more space efficient blocking factor. There are several programs available in the contributed library to calculate blocking factors. These are just two from the contributed library - there are many more.

Third party software vendors have a wide array of utilities. MPEX by Vesoft is one of the most well known. MPEX can re-block files to the most efficient blocking factor and squeeze them to eliminate wasted space past the EOF. There are other third party programs that will compress data and many other space saving manipulations.

MPE offers FREE5, STORE, and REPORT as space management tools. Each of these have their place in effective management of disc free space. FREE5 is the essential tool - a report of all disc free space. REPORT can help pinpoint what group/account has grown rapidly. This is particularly useful if you have historic REPORT listings to use for comparison. STORE is effective in archiving files to tape and when used with the PURGE parameter can delete the files from the system after archiving to tape.

An MPE utility that is essential for effective space management is VINIT. Using the COND command, all disc drives should be CONDensed on a regular basis. CONDensing re-packs free space and eliminates fragmentation. This will enhance system response time and help keep the free space in the best format possible - large chunks of contiguous space.

After system failures a Coolstart and a Recover Lost Disc Space should be done. This recovers space that the system no longer recognizes due to the system failure. If a recover lost disc space is not done regularly after system failures thousands of sectors can be lost. LOSTDISC from the contributed library is a program that analyzes how much "lost" space will be recovered when a recover lost disc space is executed. This is a valuable program because it can help you decide whether it will be worth doing a recover lost disc space.

The one tool that system managers hesitate to use for space management is a reload. Granted this is a last resort; however, you should plan to do reloads regularly. I schedule reloads every six to nine months to eliminate fragmentation that VINIT CONDenses cannot re-pack. A reload packs the data and leaves the free space in large contiguous chunks.

Management of Data Set Capacities

The management of data set capacities will prevent many problems and enhance overall system free space. This component of space management must be monitored daily to eliminate "surprises" ranging from inconvenient to critical.

Failure to maintain data sets at their proper capacities can reduce overall free space if the capacities are too high. If the capacities are set too low then performance suffers and there is an increased risk of filling up a data set. Filling up data sets usually happen at a bad time during the most critical jobs and wreak havoc.

Monitoring on a daily basis will catch those data sets that are filling up fast. These can sneak up on you and be full in a few days.

The single most important tool to monitor data sets is a QUERY FORM SETS listing for each active data base. This listing shows each data set, the current capacity, and the current entry count. There are contributed library programs that enhance this listing; however, the FORM SETS listing is all that is needed to monitor data sets. It is a very simple task and takes only a few minutes. Write a procedure for you operators, and they can do this task daily for you.

There are third party programs available that will create a data base from the data in the FORMS SETS listing. This allows for trends and comparisons. This type of program would be helpful for an environment with many data bases.

There are third party programs that will help manage data set capacities. Many have features that allow background processing for the actual setting of the capacities. These programs make the management a little easier and offer other capabilities as well.

Management of MPE and KSAM File Capacities

The management of MPE and KSAM files is the most time consuming of the three space management components. This is due both directly and indirectly to the volume of MPE and KSAM files on the system.

MPE and KSAM files must be built with the proper blocking factor, the proper number of extents, the correct number of extents initially allocated, and a reasonable number of records. A written policy has to be developed and enforced. Staff who build MPE files and/or KSAM files must follow a plan for building files; otherwise, human nature will take over and files will be built very large and with poor blocking factors.

A review of these factors for both MPE and KSAM files should be done on a regular basis. There are programs in the contributed library that will help you with this task. Vast numbers of sectors can be saved when files have been built without regard to space considerations. Special attention should be made to files with a blocking factor of 1 and to files that have all extents initially allocated. These are two potential space hogs that can be identified easily from a LISTF,2 listing.

KSAM files need to be re-organized on a regular basis to remove the records marked for deletion. KSAM files need to be evaluated for activity to determine how often they need to be re-organized. Not only will the re-organization potentially save disc space it will also enhance performance for the file.

Both KSAM and MPE files need to be evaluated for correct capacities to prevent files from filling up during update processing. Naming conventions will help here to determine which files actually are processed for updates. With a LISTF,2 listing for that fileset a quick review will determine if capacities need to be revised.

Conclusion

Space management is not difficult nor is it time consuming. A few minutes daily is all it takes.

The most elemental part of a space management program and the most difficult is developing policies and procedures. They both provide the framework for the remainder of your program. Without both written policies and written proceedures your space management program will be alot more difficult, if not impossible, to manage.

The policies, proceedures, and daily monitoring will eliminate surprises. The system manager will keep abreast of all the space management components and no one will ask "where's the space?"

****
* DATA PROCESSING PROCEDURE MANUAL *
****

S A M P L E

Subject: CONDensing DISC DRIVES

Policy: To enhance response time and pack freespace all disc drives will be CONDensed weekly. The CONDenses will be done on weekends. Check the CONDense log to determine which drives need to be CONDensed.

Prodedure:

I. Do a full back-up per usual procedure.

II. After back-up set the limit to 0,0.

III. Print a FREE5 listing.

1. Label printout "BEFORE COND" and list the drives that are being CONDensed.
2. Log off all other terminals except console.

IV. Type "VINIT" and at the ">" prompt type "COND n".

*** NOTE: n = LOGICAL DEVICE NUMBER FOR DISC BEING CONDensed

V. Warning messages will display on console.

A. At the beginning of each CONDense "HH:MM/#Snnn/nn/WARNING*System logging disabled while CONDensing (PVWARN 179)".
B. At the end of the CONDense "HH:MM/#Snnn/nn/WARNING* system logging enabled by CONDense (PVWARN 180)".

VI. After the appropriate drives have been CONDensed successfully:

A. Repeat step IV-E if time permits more drives to be CONDensed.
B. Exit the VINIT UTILITY.
C. Record CONDenses in the CONDENSE LOG.
D. Run a FREE5 listing.
   1. Label printout "AFTER COND", and list the drives that were CONDensed.
   2. Send FREE5 listings to DATA Processing Supervisor.

Application Software as a Long Term Investment

Arthur J. King
SOTAS, Inc.
192 Merrimack Street
Haverhill, MA 01830

Introduction

The purchase cycle of financial software has evolved considerably over the last decade. It has become a complex and sometimes confusing period for the purchasing company. More and more, people are realizing that the software they are purchasing should be viewed from the long term as opposed to the short term, "How do I solve today's problems?", perspective. This session will examine this evolution as well as the importance of looking at the *"big picture"* when reviewing your software needs. The four steps necessary to ensure a solid, long term investment: 1) Investment Planning, 2) Deciding on the Right Investment, 3) Protecting Your Investment, and 4) Reaping the Rewards of Your Investment will be discussed in detail. This session will also cover the importance of having a close *working* relationship between the software purchaser and the software vendor to ensure an investment that will meet their current needs and create a solid "Foundation for the Future".

I. Evolution of the Financial Software Industry

In the late 1970s, the financial software industry had just begun to get its feet wet and was experiencing its first growing pains. Many software companies were begun as one system shops. If the user wanted to purchase Accounts Payable and General Ledger, for example, the user would probably buy two different products from two different companies. This was because certain companies had established a reputation for having a good Ledger, yet they would have either no Accounts Payable or a weak AP system at best to offer. The only way to come up with a satisfactory solution for their organization was to purchase the applications separately. This meant that there were two different communications channels, two different support organizations to call, two differently designed systems and worst of all there were two separate learning curves. Unfortunately, just because the user knew how to logon and run jobs on the General Ledger system, the user would still be a novice as far as the Accounts Payable system was concerned. This became an even more frustrating situation to the user when they realized that there were many custom changes made to their system that were not documented or supported by the vendor. These changes occurred because there were usually some features or functions that were missing from the "standard" system. As the vendor had to survive on the basis of one application, they would generally try to add a quick fix to that client's system. This patch might address the client's initial problem, but as future releases of the system were sent, these patches would become outdated. The vendor might have a record of these fixes, but even if they did, it was often a lost cause trying to patch them into the new system. Usually, the user was left with two choices: 1) reinvent the wheel by rewriting a new patch, or 2) give up on the new release and struggle with the old one. Despite this gloomy situation, many users were satisfied with their situation mainly because there was no alternative. The other problem came in the area of bridging information from one application to another.

Many users wanted the information from their sub-ledger products such as Accounts Payable and Accounts Receivable to be passed automatically to the General Ledger system. With two separate applications, the only way to pass information to the General Ledger system was to either key the information in manually from an Accounts Payable report or to write their own bridge program. If the user chose the latter option, they had to carefully monitor any new releases to either system and make changes to their bridge program accordingly.

As the software industry evolved, the vendors realized that they could no longer survive as a one product company. The survivors began to expand their product breadth, in one of two ways. The first method was to establish a separate task force that was out of the mainstream so that they could develop a system faster and without interruptions. This procedure was good enough to develop their first module, so they felt that it should work for any other module. The problem with this method was that it was slow and although the people could develop a Fixed Assets system, it didn't guarantee that they could develop a reasonable General Ledger system. The second method was to acquire the software from some other vendor with the promise to make it different and to not compete with that vendor with the same product. The problem with this approach is that the vendor had very little knowledge of the application that they were supposed to be selling and supporting. In either case, the second and subsequent systems produced by the vendor seldom bore any resemblance to the system on which they had built their reputation. This was because styles are different and the separate development group had their own style just as the company that originally developed the acquired software had theirs.

Many vendors ended up choosing the acquisition route because it was faster, and to some extent, the software that they were purchasing had already proven itself in the real world. This meant that the product line offered by a vendor was stronger in that they now had more competitive products to offer. In fact, to this day, the reason that some applications look alike from vendor to vendor is because of the rampant interbreeding of some applications.

Product Line in Late 1970s

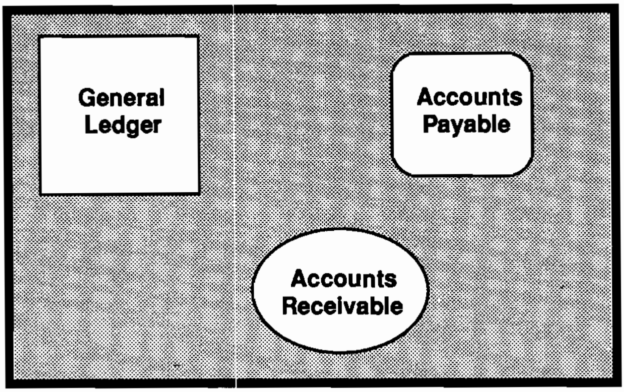


Figure I-1

With the situation changed at the vendor level, the user had some extra choices to make when they decided to purchase software. Although Vendor A's General Ledger was stronger than Vendor B's, Vendor B's Accounts Payable system might have significantly more bells and whistles than Vendor A's. The user would then have to decide whether the advantages of having the "best" systems (buying one from each vendor), were outweighed by the advantages of purchasing both systems from one vendor. There would be some cost savings through purchase price discounts, multiple product support licenses and the ability to contact one organization for all their software needs. However, there was still the problem with the learning curve, because even if both systems were from the same vendor, they probably looked different. The vendor might possibly build the bridge between the sub-ledger systems and the General Ledger system for the user, and if the user was persistent enough, they might even maintain these bridges as new releases came out. To some extent, there was still the problem with customization as many users pressured the vendor to add features to the system so that they would not lose as much by purchasing from one vendor instead of purchasing the two best systems. Although more and more users were switching to a one vendor solution, there were still many users who preferred dealing with multiple vendors to get the best solution.

As the 1980s progressed, the software industry matured even further. Many vendors realized that the key to increased sales was through two avenues. One was to improve user friendliness and the other was to make the entire product line more similar. This increased the likelihood that the user would purchase more than one software product from the vendor. As the features and functions became more and more similar between vendors, the buzzword became "user friendly". In other words, systems were finally being designed so that they could be used by the end user without heavy involvement from the data processing staff. This meant that documentation, screens and procedures had to be redesigned in such a way that it was understandable at a nontechnical level. Another change to improve user friendliness was the increased use of common tools that could be used by more than one application to make the data entry and reporting options easier. The vendors also established some design standards so that systems would look and function somewhat similar. This helped the user reduce their learning curves as the knowledge they had on one system was somewhat transportable to the other systems. Since the core applications were very similar between vendors, they had to search for new ways to differentiate their products. Software vendors began to realize that there were many areas outside the *core* of the application that they could focus on. They began developing Query/Reporting facilities, Security facilities, Links to PCs and hooks into other applications.

Product Line In Mid - 1980s

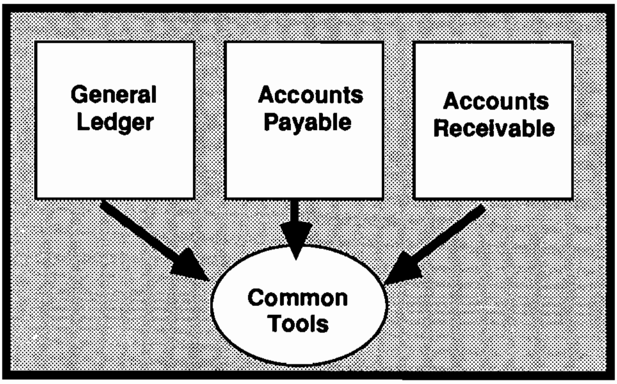


Figure I-2

This latest step in the evolution cycle was a welcome change to the end user. Firstly, they no longer had to sacrifice major feature/functionality when choosing between vendors. When evaluating software, the user could spend more time focusing on how the software fit in their environment from a usability standpoint. This change in focus put a much higher priority on the user friendliness of input screens, the flexibility of reporting and general ease of use of the applications. Also, the user did not have to be as concerned with software customization because, with the wide range of choices available, they no longer need a quick fix to meet their needs. Most of the changes could be made external to the application or through the use of the hooks that the vendor had established. It was becoming more and more advantageous to purchase all of the financial systems from one vendor because of the reduced learning curves that resulted from the similarity of products as well as the ease of having one contact for all the user's systems. It was also a real time saver if the user was interfacing subledger products to the ledger. The interfaces were already written and supported by the software vendor. There were no longer any worries that when a new release came out the systems would not interface with each other. In summary, there was a much wider and more complex choice of solutions for the user. The advantages of going with one vendor were very clear; the concern was in choosing the right one. One way to help clarify the decision is to take a look at what the future of financial software applications.

In the 1990s, the evolution of the financial software industry will focus even further away from the individual core applications and it will focus on integration and other external factors. The user of financial software in the 1990s will also be focusing on many nonapplication specific areas. These changes will be in three areas: 1) User Tailored Drivers, 2) Integrated/Shared Databases and 3) Comprehensive Services.

Product Line in 1990s

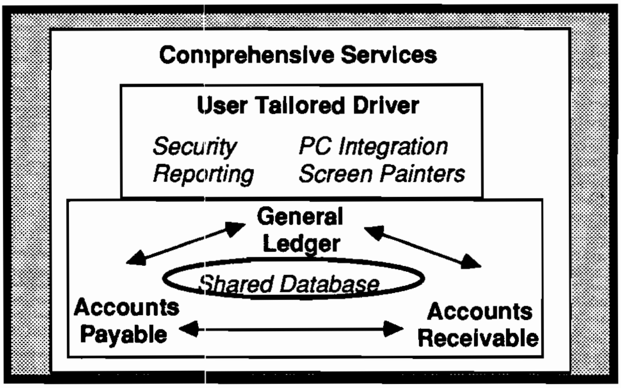


Figure I-3

In the area of user tailored drivers, the user community will demand and the software vendor will supply more PC oriented applications that work as an integral part of the main application. This will allow the user who has invested in PCs and LANs to be even more productive with the tools that they are already using. These PC modules may serve as an entire subsystem or they may serve as a tool to enhance an existing application process. The end result is that it will make the applications more accessible to more users. Another area that has already received attention is the area of security. As the software application has been opened up to a wider range of users, there has been a need for tighter control over who can access what information. As the applications become integrated even further, there will be a need for user defined security down to the level of restricting certain users from accessing certain fields while performing certain tasks. These security modules will reside over all applications and will allow the user to have complete control over the level of security that they are looking for. As always, there will also be a need for completely flexible reporting and inquiry tools. Software vendors will enhance existing canned reports to allow for greater flexibility from both a format and a data selection standpoint. There will be a change in philosophy in regards to report writers with a greater dependence on tools that already exist that can access any application. Many software vendors will turn away from writing their own report generators and instead integrate their software with the report and inquiry tools that the users are already using at their site to access their existing applications. Another important piece of the User Tailored Driver facility will be the screen painters. These screen painters will allow the user to tailor all of their screens, whether they are input or inquiry screens, to their environment. The software vendor will supply skeletons of prepainted screens with prelabelled fields. If the user wants to change these labels so that they are more meaningful, it would be a simple matter of entering the new name into the system through the screen painter facility; and that new field label would be displayed throughout the system. These User Tailored Drivers allow the user to custom tailor the system to their specific needs without modifying the system. This drastically improves the maintainability of the system for both the user and the vendor.

In the case of integrated databases, this would mean a fundamental change in the way that both the user and the vendor view financial applications. Currently, a financial application is considered a separate module that performs a specific function; such as, General Ledger, Fixed Assets, or Payroll. As the applications and sophistication of both the user and the vendor has evolved during the last decade, the borders between these applications are starting to be broken down. As you have already seen, there has been a trend towards making applications look similar, integrating applications and the sharing of application tools. This has allowed the user to easily navigate between two separate applications. However, if you step back and take a look at the whole picture, you will see that each of these so called modules, you will see that each of them is really a function of the entire financial process. If, for example, your company was to purchase a machine that manufactured widgets, and you produced those widgets for wholesale widget distributors, you might step thru five separate functions which would encompass five separate financial systems. First, you would write a purchase order for the widget machine using the purchase order system. Next, you would enter the invoice for the machine on the Accounts Payable system. You would then have to set up the machine on the Fixed Assets system for depreciation purposes. After the machine is up and running, you must keep track of the manufactured widgets thru the Inventory Control system. Orders for the new widgets are entered thru the Order Entry system and finally all billing for these orders would be processed thru the Accounts Receivable system. There are six different but related accounting functions which would require journal entries to the General Ledger system. In the financial system of the future this would be handled by six functions within *one* umbrella financial application sharing one database. Some of these functions would be automatically generated for the user instead of

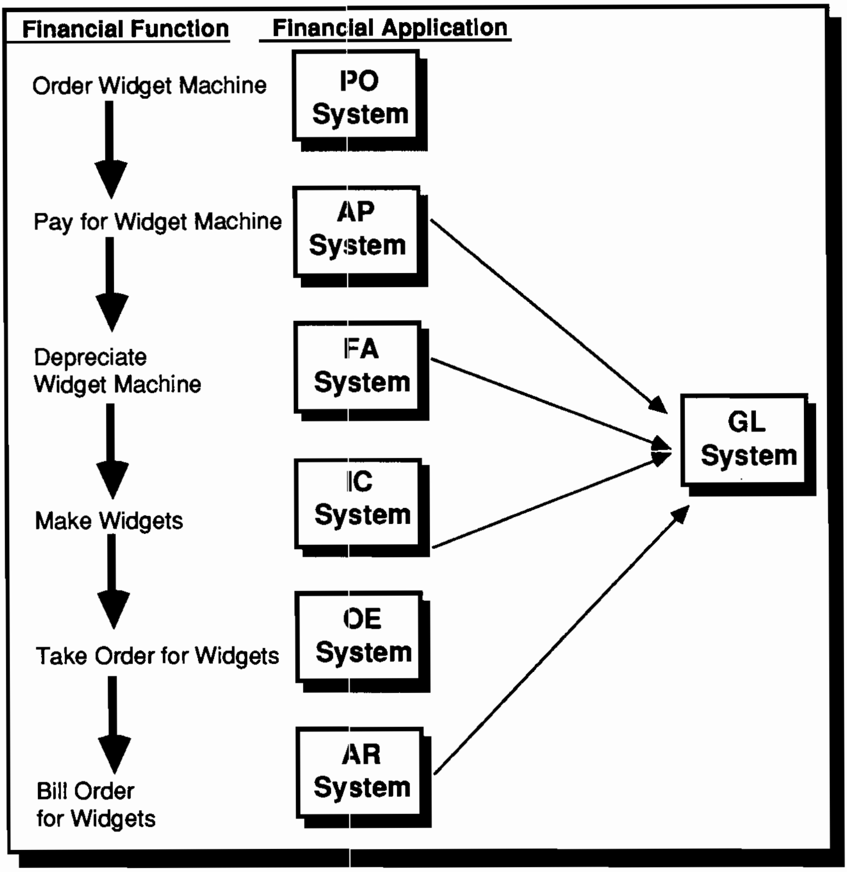


Figure I-4

being manually entered thru six separate systems. The redundancy of data would be significantly reduced thru the integration of the financial software database. In short, the user would have increased control and simplified input of information. It would help streamline the entire financial function.

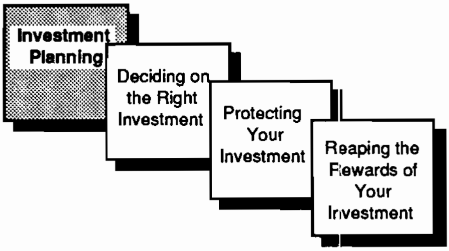
The third change that will take place in the product line of the 1990s is the increase of built-in and add-on services from the vendor. The one area of financial software that has not yet evolved is the area of software services. In many cases, the only support that a client may receive from the vendor is telephone support. Any major new enhancements may be at an additional charge. The software vendor of the future will see this as an area in which they can offer a major advantage over the competition. These new services are already starting to take hold as users are demanding a better value for their maintenance dollar. These services could include *Comprehensive* support which covers the user from installation and planning thru custom enhancements and upgrades. Phone support will be augmented by dial-in support and increased use of electronic mail. This will speed up the problem solving process. Systems may be shipped with self-diagnostic modules that will assist the user in pinpointing and solving the problem. There will also be a noticeable change in the relationship between the user and the vendor. There will be an increased sharing of ideas. Vendors will work closely with their user base to plan and develop new ideas. Users will share their ideas and modifications that they have made with the vendor. It will no longer be an *us against them* relationship as both the user and the vendor will realize the benefits of working together. The user will be able to get the enhancements that they want into the system, resulting in a happy customer reference base which will subsequently enhance their sales. It will truly be a joint relationship for the good of all.

Now that we have examined the evolution of the financial software industry, let's take a look at how this impacts the software purchasing cycle.

II. Ensuring a Solid Long Term Investment

The evolution of the financial software industry has drastically changed the way in which users purchase software. In the 1970s many users when through a limited evaluation. This was because of the reputations that had been established by the software vendors. Unless there was an extraordinary need, there were certain vendors from whom users knew they should buy certain software. As the industry evolved, the purchase cycle became more complex and more involved, users began to do more research into which product or products would best fit their needs. They began talking to other users, reading documentation and having demos. As the product lines evolved even further, it became even more critical to review in detail all of the factors prior to making the purchase decision. As users began to view the purchase of software as a long term investment, it became important to take a calculated step-by-step approach to making and nurturing that investment..

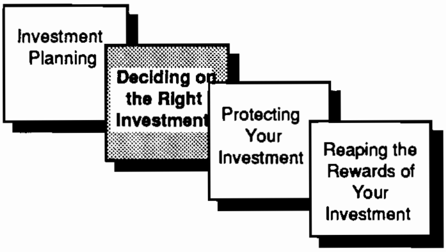
The rest of this discussion will focus on a four step approach: 1) Investment Planning, 2) Deciding on the Right Investment, 3) Protecting Your Investment and 4) Reaping the Rewards of Your Investment.


Investment Planning
Deciding on the Right Investment
Protecting Your Investment
Reaping the Rewards of Your Investment

III. Investment Planning

The first step is Investment Planning. At some point prior to making the purchase decision, you should go through a cost analysis to determine how much it would cost to develop and support the application in house versus purchasing the software and the support from a software vendor. Generally speaking, you will find that there are significant savings, especially in the area of maintenance, by purchasing the application. This will give you a cost bench mark to use for comparison purposes. It is very important to decide what your needs and priorities are *before* getting seriously involved in the purchase decision. For example, before shopping at a car dealership, you would first determine your basic type of car preference and which options were high priorities.

To do this you must assemble a team made up of users, accountants, data processing staff and other appropriate decision makers. This will help assure that everyone that is affected will be represented in the decision process. It is also important to spread the responsibility of the research amongst several members of the team so that not just one or two people bear the total responsibility of the decision, especially in the early stages, when you should get as many different opinions as possible. Next, a list of your needs from hardware to detailed features and functions and from service, to future plans should be compiled. Each of these detailed items should be prioritized, and then you should determine the vendors that could possibly meet those needs. The avenues that can be pursued to come up with a long list of potential vendors include scanning directories, talking to other companies that you know have financial software, getting information from your hardware vendor and talking to other users at a hardware user conference such as this one. Ask for brochures and as much information as you can from the vendors and have the team review the materials. After you feel comfortable that you have reviewed enough materials from enough vendors, you should narrow your focus to the best three to five vendors; this will become your short list. One more step to deal with before going forward with your short list is to project where your company will be heading in the next five to ten years. Try to forecast what your accounting needs will be during that period so that you can select a solution that meets your needs now and in the future. This would include examining your hardware plans, usage of PCs and any other needs that you can envision. After you have completed this, youwill be ready to proceed to the next step of the decision process.


Investment Planning
Deciding on the Right Investment
Protecting Your Investment
Reaping the Rewards of Your Investment

IV. Deciding on the Right Investment

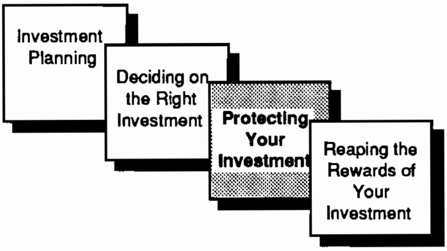
Now that you have narrowed your potential solutions down to a short list, it is time to get down to the serious business of deciding on the right investment closely examining each of the options. The first step to take with your short list is to assemble the list of your needs and to produce a document generally referred to as a Request for Product (RFP). Next you should meet with the vendor's sales representative and have a discussion that covers such topics as an overview of the vendor's company, the vendor's goals and philosophy as well as a discussion on the user's goals and objectives. It is also the point in time where the RFP should be given to the software vendor's sales representative. You should give the vendor a reasonable amount of time to review the RFP and return a written response to you for your review and you should also schedule a follow up meeting for a detailed presentation. After sitting down and evaluating the responses and comparing them to the priorities that you have determined for each issue, you should determine what follow-up questions must be asked in the detailed presentations. It is important that the detailed presentations be conducted in two parts; the first part being a detailed overview of the system with the second being a demo. The detailed presentation should be scheduled at a time when the entire team can attend and it should be uninterrupted time so that all team members can attend all aspects of the presentation. Questions from the RFP that needed further clarification should be asked at this time.

This are also two critical pieces of information that should be addressed at the detailed review.

First, obtain a list of some of the vendor's users that you can talk to and find out their impressions of the software, services and the performance of the software vendor. It is usually a good idea to talk to the chairperson of the vendor's user group. This will tell you whether there is a strong relationship between the vendor and the user community and if there is a user group. A good sign of an investment that will grow with you is if there is already a mechanism in place for the users to work with the vendor on ideas. The chairperson is also a good sounding board for the mood of the entire user base as opposed to the narrower focus you might get from one user. You also will usually get a less biased picture from the chairperson than you would get from a vendor selected reference.

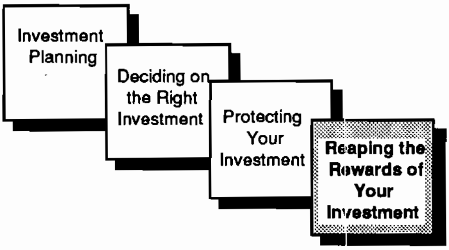
Secondly, review the future plans of the company for its product line. This will help you determine if the vendor has taken the time to sit down to plan for the long term or whether they are still focussing in on the short term. When I say long term, I am talking three to five years down the road, not just plans for the next release. This will give you a chance to see if the vendor is a good *partner* for the long term and whether the vendor's plans and your long term plans are along the same line.

After completing the detailed review it is time to make the final decision. Using the information from the RFP, the detailed review and the users' references, it is time to come to a decision and negotiate a final contract. If you have followed the above process, it should become quite evident who the final choice should be. If there is any question at all and you are at a toss-up between two vendors, you should go with the vendor that appears most willing to work with all of its users and who appears to be positioned with the best future plan. This is not to say that you should buy "vaporware". In fact, you should be able to predict which vendor will have the least likelihood for "vaporware" selling by who has spent the most time determining where the future of the industry lies. Usually, the vendor who has enough confidence in what they are doing that they does not have to sell future products because they have committed to a long term goal. That goal would be damaged if they began promising pieces of it before they could deliver them. Also beware the vendor that promises to complete all of the development projects that you are interested in this year and is willing to shift their development plans just for you. This vendor probably makes the same promises to all of its prospective users and never delivers once the contract is signed. Look for the vendor that is willing to give an honest answer even if it might not be what you want to hear because that vendor is more likely to deliver in a timely manner. The bottom line is that you are making a long term investment and it is important to take the time to make sure that you leave no stone unturned and that you have all the information that you need to make the right decision. The main difference between a short term solution and a long term solution is in what you choose to do after the purchase.


Investment Planning
Deciding on the Right Investment
Protecting Your Investment
Reaping the Rewards of Your Investment

V. Protecting Your Investment

The investment cycle does not end with the purchase of the software. If you were buying a car, you would take the time to tune it up regularly, change the oil and performed other maintenance tasks designed to make your car last longer. This is the same idea that you should follow for your financial software. It is especially important to get started off on the right foot. This means taking the time to sit down with the vendor to plan an implementation schedule that includes training on the systems for all the individuals who will be involved with the system. This would include accountants, data entry clerks and even the management staff that will be requesting certain financial reports. The more everyone understands how the system works, the easier it will be to get things done. The implementation schedule should be spread out in such a way that different systems are being installed at different times because chances are that many of the same people will be involved in each system. As I said before, each application is merely one step in the financial function, not a totally different function that would be handled by totally different people. It is also helpful if you set up a procedure for calling the vendor's support. It is generally a good idea to have the same one or two people making the phone calls so that both parties get a chance to build a strong working rapport. Once your applications have been implemented smoothly and you have established a procedure for calling the vendor , it is tempting to just sit back and take the changes as they come. You must become an *active* rather than *passive* participant in the process if you want to protect your investment. You should subscribe to vendor or user newsletters to keep on top of what is happening. You should immediately become an active participant in the user group of that vendor's users just as you have taken the time to come to this conference. Many valuable pieces of information are learned and shared at conferences such as this. First and foremost, you get to meet other users from other companies and learn how they use their system. It is a great opportunity for discussing "what if?" ideas with your peers. It is also an opportunity to influence the future direction of the vendor's product line. Software vendors should welcome the chance to work *with* their users. It is to everyone's benefit, especially the vendor's, if everyone is pulling together in the same direction. These user groups generally have some form of committee that acts as a planning staff with the vendor as the vendor begins planning new releases. If you want to protect your investment, then you must be a participant in this process. As I stated earlier, the software industry is evolving to the point where there is an intense sharing between the vendor and the user. If you do not make the effort to share ideas, then you are taking a risk that your investment may become outdated because your needs were not addressed.


Investment Planning
Deciding on the Right Investment
Protecting Your Investment
Reaping the Rewards of Your Investment

VI. Reaping the Rewards of Your Investment

As is the case with any investment, the more you put into it, the greater the return. If you take the time to sit down and analyze both your current and future needs, review all available options prior to making the purchase, and become an active participant after the purchase, the rewards can be great. You will reap a system that will meet the demands of your company and will build a solid relationship with your vendor. More importantly, you will make an investment that can grow with you and change to meet your needs as you change. You will end the dreaded cycle of having to trade in your software every few years because it no longer meets your needs. Granted, it takes a commitment of time and people up front, and it has to involve many resources. However, if you take the time to do a thorough job up front, there is no reason why you will have to trade in the software again.

VII. Conclusion

In summary, it is important that you look at the long term ramifications of your investment decision. Take a look at where the industry and your company are headed and determine if the vendor is prepared to handle any new changes. . Verify the fact that the vendor is willing to work *together* with their client base. Explore every option prior to the purchase, but don't let it end there. Like any other worthwhile relationship, you must constantly work at it to keep it strong. If you sit back and let the events happen without providing any direction or input, you will only have yourself to blame if the software does not meet your future needs. Take the time to make an investment of your time in the software investment process and you will have built a solid "Foundation for the Future".

TITLE: Twisted Pair: A Thing of the Past and The Wave of the Future

AUTHOR: Mark Indermill

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 0073

PLAYING THE WRONG GAME: MEASURING PROGRAMMER PRODUCTIVITY IN A 4GL ENVIRONMENT

Gene Harmon
AH Computer Services, Inc.
8210 Terrace Drive
El Cerrito Ca. 94530

There are many answers which were developed in response to some extremely difficult questions posed as part of the development of the period we now term the third generation of computer technology. The question considered in this paper will be the one which asks: "How does one measure programmer productivity"...to which we will add the phrase : "in a 4GL environment." This question is being heard with increasing frequency lately, and it, in turn implies a series of other questions which can be thought of as part of an agenda which has been left to us as part of our heritage from the third generation of programmers, their managers and their users. I see these other implied questions as the following:

-Is there such a thing as a programmer in today's 4GL environment?

-What is the definition of work for today's knowledge worker? And who does what?

-What is the end-product desired in such an environment?

-And. . . what is the best way to ensure a successful delivery of that product?

-What are the impediments to excellence?

These are all questions which come up and are covered with endless regularity at this and every other conference in the world, and my experience is that when I attend these types of lectures, I am both amused and enlightened; and when I look around, (expecially if it is a good speaker,) I can see a sea of nodding heads. So, it would be a mistake to say that yet another utterance by me will solve these seemingly perennial problems . . . yet, it does seem that, over the years, the environment provides enough change to prompt for a serious re-evaluation of our tasks in the workplace. Some of the new by-words in the market-place are :

- Competition	[the Japanese]
- Responsiveness to the Market	[developement cycle]
- Lean and Hungry	[fire middle management]
- Innovation	[RISC architecture]
- Quality	[Chips, Cameras, and VCR's]

Although all of the words have an effect on each and every one of us, no matter what our job in the marketplace may be, it is the last word "Quality" which would seem to exert the most leverage in our topic for today. No, this is not going to be a talk on the merits of quality control; so please don't have the reaction of the American who was caught in the wrong place in the jungle with his Japanese and French co-entrepreneurs . . . immediately prior to execution they were allowed to say one last thing: the French person asked to be allowed to sing his national anthem; the Japanese wished to deliver his lecture on quality control one last time; and the American requested that he be shot before the Japanese so that he would not be forced to listen to the lecture one more time. So, the only stave in the quality control lecture I shall use to beat you over the head with is the following quote from David Halberstam's book, "The Reckoning" :

". . . quality was not some minor function that could be accomplished by having some of the workers at the lowest levels attend a class or two, or by appointing a certain number of inspectors to keep an eye on things. True quality demanded a totality of commitment that began at the very top: if top management was committed to the idea of quality and if executive promotions were tied to quality, then the priority would seep down into the middle and lower levels of management, and thus inevitably to the workers. It could not, as so many American companies seemed to expect, be imposed at the bottom. American companies could not appoint some medium-level executive, usually one whom no division of the company particularly wanted, and for lack of something better to do with him, put him in charge of something called quality. The first thing that an executive like that would do, Deming said, and quite possible the only thing, was to come up slogans and display them on banners. If the company treated quality as a gimmick or an afterthought, then true quality would never result. Above all he was saying, quality had to be central to the purpose of a company."

This book is well worth reading, and one of the most telling incidents in it concerns an American visitor to the Nissan automobile factory who politely asks where the holding area is for cars which did not pass inspection and must be repaired. There, of course, is no need for such an areas the the Nissan factory, although each American factory has a repair bay at least the size of a football field to hold its rejects.

Now, the lesson here is not how to make better cars. The point is that anyone who makes a committment to quality and who positively refuses to compromise or surrender that committment, can expect superior results. Another lesson this book makes, in a very excellent point, is that there is no one operational improvement that one make which suddenly transforms your world; rather, it is a series of reasoned, methodical changes made by a team striving toward a goal of excellence.

Considering the plight which really does affect us all, I think it behooves us to consider the questions posed earlier in the talk, and to do all that we can possibly do to provide answers to them--which just might mean that our jobs ten years from now will still be in the computer business rather than in the fast food business. If this statement seems at all bizarre or laughable to you, as you are enjoying the ambience of this resort atmosphere on tax deductible money, remember that just 15 years ago, people with similar messages were being laughed out of the boardrooms of the big three auto makers who are now existing at the sufferance of voluntary quotas by Japan and Korea, and by the mercy of government bailouts; to paraphrase Sinclair Lewis: "It Can Happen Here!"

Question #1: Is there such a thing as a programmer in today's 4GL environment?

-Old 3GL role Vs. New 4GL role.

I speak today as a refugee from the 3GL environment where I was first a trainee, then a programmer, then a programmer/analyst, than a senior programmer/analyst, then a systems engineer, than a technical consultant, then a contract programmer, and now a senior consultant who is back to typing in his own code (even a programmer got to hand his coding sheets over to a key-punch operator back in the old days). In some ways, it seems I have come full circle while theoretically having advanced to the 4th generation of programming languages. . . while I can enter three statements which will cause a processing screen to be built which can add, update, and delete a record with many fields in it, I find myself pondering escape sequences to send commands to laser printers. What has changed? Primarily; the combination of new hardware and software has released business organizations from the total dependance upon a hierarchical data processing staff. Although there may be large DP staffs associated with 4GL envionments, they are typically populated by individuals with a wide range of skills. They are no longer limited in their activities by archaic job titles with corresponding specialized narrow ranges of action. They consider it a part of their normal activities to consult with the user, design databases, write and test programs, and consider new hardware/software options as solutions to user requirements. In a word, the old definition has "vanished".

-Old Specialized Skill Sets Vs. New Omnibus Skill Sets

Repeating my theme of initiation into the world of data processing via the third generation, I can recall taking the IBM programming test which was to measure my aptitude for programming. High emphasis was placed on that test, and I imagine that the sales representatives for that organization were required to take an entirely different test. The litmus test for those who would succeed in today's world would seem to me to be a hard one to construct. The requirements for success are so varied and so demanding that it seems unfair to ask any one person to fulfill them all. It is not unusual nowadays for a two or three person staff to be responsible for the complete data processing requirements for companies doing millions of dollars of business a year. These circumstances require that from the resources and talents of these few people; all user interface, hardware and system requirements, software and programming needs, vendor interface, internal management, and planning for the future- must be satisfied. Such high demands cannot be met by the static requirements of job descriptions dredged up from old 3GL references, they require a data processing staff willing to learn a wide range of technical, business, and interpersonal skills; whatever the cost, and wherever they can find them.

-Old Vendor Knowledge Pipeline Vs. New Market Pipeline

The 3GL workplace was one in which information about new products, whether hardware or software, arrived via the vendor salesperson, from some comment a manager might hear at DPMA dinner, or perhaps in the pages of ComputerWorld. Information of this sort might or might not filter down to the person actually dealing with the problems, and users typically had to make do with DP solutions which have been developed five to ten years prior, and hope that his latest requests might be handled by some anonymous maintenance programmer. Much of that scenario has now changed. It is not uncommon to see the entire staff of a 4GL installation in attendance at popular user group meetings, even those which might require traveling hundreds of miles. At these meetings each staff member has the opportunity to sample the wares of dozens of vendors and attend technical talks where he might encounter an entirely new solution to his problems entirely by chance! This new market pipeline of access to vendors and peers represents both a challenge to select from the marketbasket of offerings and a requirement to maintain current knowledge of the latest hardware and software solutions.

Question #2. What is the definition of work for today's knowledge worker, and who does what?

- Re-evaluation of on-site and off-site time
- Redefinition of roles

Much of what has to be dealt with here for this question is a natural outgrowth of the first question; the new responsibilities foster a new definition of tasks for today's knowledge worker. In order to provide the user a satisfactory solution to his problems, the person providing that solution can no longer exist in a cubicle laying out lines of code; the systems analyst can no longer rely on tried and true methods of batch processing to present monthly and annual reports from which the accounting department must extract the information needed to generate financial information. Time well spent by the worker of today consists of such varied tasks as reading the latest technical magazines, experimenting with the latest software package, and attending every users group meeting possible. These activities are no longer the domain of just a privileged few in the DP organization, they are an obligation of all who would maintain functionality in a 4GL environment. It is necessary that all staff members who have the task of implementing user requirements also develop the expertise to extract those requirements from the user; those who have no direct feeling for the user's needs will not be able to focus their full implementation talents on a refinement and enhancement of those needs. There is a book written by Nevil Shute about a religion which has sprung up amongst a fraternity of aircraft engineers in Asia which contains the lines:

"Aeroplanes come to grief because of wrong cravings and wrong hatreds and illusions in men's hearts. One of you may say, "I have not got the key to the filler of the oil tank. I cannot find it. I looked yesterday and there was plenty of oil. It is probably all right today." So accidents are born, and pain and suffering and grief come to mankind because of the sloth of men. . .' . . . It was the same message that he had preached so often in the hangar at Bahrein, that the maintenance of aeroplanes demanded men of a pure and holy life, men who would turn from the temptations of the flesh to serve their calling first."

The analogies to a DP worker are easy to see here; though one might reasonably draw the line at certain sacrifices hinted at, the spirit of dedication to the task at hand is surely admirable.

Question #3. What is the end-product desired in such an environment?
. . . and what is the best way to ensure a successful delivery of that product?

Questions such as this one surely lurked in <u>few</u> programmers' minds of the third generation; usually any programm that ran bug-free to end-of-job was considered a roaring success...if one could do it quickly and often enough it was a passport to the next job-step or to the next job for more money. Data processing empires were built and turfs were staked out in fights over personnel and budgets; although many of these fiefdoms still exist, they are slowly being undercut by users getting their hands on PC's and sometimes using their own budgets for small mini-computers which can be tailored specifically to their needs. Thus, the end-product has really been redefined by the new technology and the ability of the users to insist that their needs be directly satisfied. This new definition of the end-product means that the 4GL DP person must learn how it is that the business is run, and then make sure that the system of programs used accurately reflects that flow; this is quite different from capturing information from after-the-fact forms and then churning out reports. The main difference seems to be that while a 3GL allowed one to report upon the state of the business, the new 4GL allows day-to-day operation and control of all business operations. These new abilities mean that the end-product of DP endeavors is now very closely tied with the goals of the business which it serves. An effort must be made to make sure that these business goals are fully understood, lest the new technology hinder rather than help them.

Question #4. What are the impediments to excellence?

Everyone, I"m quite sure has personal experience with these impediments, whether they be political blockades or short-sighted economies. Many times it is impossible to exert the leverage necessary to overcome these impediments; however, it is possible not to construct them yourself. In many environments, it possible to get users to sign off on systems which fall short of doing the job that could be done, and then allow yourself to become the victim of inertia and merely maintain your position instead of going forward... do not let this happen. Allow yourself to be open to new methodologies, but do not surrender yourself and all your existing systems to them.

CONCLUSIONS

THE CYCLE OF IMPROVEMENT

- Commitment to Quality
- Statement of Goals
- An Agenda

It seems to me that once you are determined to do the best you can possibly do in your particular environment (or possibly change the environment itself), a committment to quality which emanates from management throughout your organization is an excellent first step in the cycle of improvement. Remember that each step of this cycle, as well as the total aspect of quality consists of many smaller increments, and of many choices. Remember also that some of these choices may be difficult ones; ones which may not be accepted by everyone, and with which you may not be in love yourself. Lastly, always remember what Walter Bagehot, a 19th century British essayist said:

"One of the greatest pains to human nature is the pain of a new idea"

It is also necessary to state what the goals of your particular organization are . . . note that is easy to develop beautiful systems while, at the same time, failing to meet the needs of the user of those systems. The goals developed must be in alignment with the reason for your organization's existence. Once you have made you committment to quality and stated your goals, you must then develop an agenda, or a plan for action. You should not fall into the trap of running your organization as the Harvard Business School would run it:

". . . too much information, too many options, too little feeling about the product."

Just say "NO" and try to do a few things well before going on to the next group of a few things that need doing.

My own feeling is that the cycle of improvement is very much an iterative process which very much demands constant re-examination and tinkering. Before it becomes necessary to include the measurement of programmer productivity into this cycle (for the want of something to do with your time) I hope that all of us here have graduated to the 5th GL and are totally occupied with a brand new set of challenges.

TITLE: Integrating Paperless Systems in a Fortune 100 Company

AUTHOR: R. L. Pringle

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 0075

HOW TO TRAIN A TERMINAL USER
TO BE AN EFFECTIVE PC USER

Jack K. Marshall
Solano County Management Information Systems
601 Kentucky Street
Fairfield, California 94533

SCOPE OF THIS PAPER

This paper addresses the transition from a terminal to microcomputer-based workstation environment for the user community in a county government. The paper will focus on the training needs of these users and how these needs have been addressed by our training department.

BACKGROUND/ENVIRONMENT

When I joined Solano County MIS in March of 1984, the bulk of the processing work was performed on Hewlett-Packard 26XX series terminals connected to our then single HP 3000 series 68.

Today the County has over 400 terminals connected to three HP 3000 series 70's. In addition, we now have 75 HP 150 Touchscreen and 30 HP Vectra microcomputers.

Along with a growth in the sheer volume of devices attached to the 3000's, there has been a shift away from terminals to PC workstations.

When new management arrived in 1983, there came the addition of the first series Hewlett-Packard Series 68 and a dramatic growth in the number of on-line systems written and purchased for use on the 68. Financial systems, case tracking systems, client services systems. And all were to be accessed using terminals.

GROWTH OF THE TERMINAL INSTALLED BASE

As applications systems were completed and installed on the 3000/68, terminals were purchased and installed in the user departments. The user was given a terminal operation manual and told to now perform the job they had done manually for years using their new terminal.

You can imagine how smoothly the transition went. Department management often used the "big stick" approach to force the use of these terminals and the users learned only enough to perform their immediate tasks and nothing more. These expensive computer resources were vastly under-utilized.

In fairness, the customer support staff at that time consisted of a single staffer transferred from a data entry position. She was responsible for unpacking, delivering and installing all terminals and printers. She was also responsible for help desk coverage and what little training that could be provided to users on a one-on-one basis.

The position of Customer Services Manager was created and I was hired to fill the slot in 1985. I have since built the Customer Services Department to a staff of four. The installation, help desk and training functions are each now full-time equivalent jobs. For over two years, Customer Services has conducted a regularly scheduled program of classes in all areas of information processing. However, four years after systems and terminals started proliferating, we are still playing "catch-up", trying to locate and train all those first terminal users who never received adequate training and continue to use their terminals in a sub-optimal manner.

When our MIS Department outgrew our single facility, we had to acquire an additional building. I was able to negotiate for a separate Training Center in the new building. We installed four 2392A, four 2628A terminals, a letter quality and a line printer.

Creation of the Training Center was central to establishing a quality training program for the County. It enabled me to train a large group of students at one time in a controlled atmosphere with no distractions. A dedicated training facility is a must if you can pry the resources loose!

TRAINING ISSUES: TERMINALS

In order to assist our users in becoming more proficient in the use of their terminals, I developed and our department offered, at no charge, a regular schedule of classes in what I refer to in Figure 1 (page A1) as Level 1 courses:

Beginning HP DeskManager, 4 hours
Advanced HP Deskmanager, 4 hours

covering the following major topics:

keyboard operation

logon/security procedures

basic troubleshooting

how to find information in hardware and software manuals

using HP Deskmanager

Additionally, we offered training courses in specific HP 3000-based applications including:

Beginning HP Word/3000, 18 hours in 3 sessions

Advanced HP Word/3000, 6 hours

HP Listkeeper, 4 hours

Data Entry, 18 hours in 3 sessions

Opticalc, 12 hours in 2 sessions

GROWING NEED FOR MICROCOMPUTERS

My background before joining the County was in the private sector as a PC and multiuser PC systems integrator. One of my first lobbying efforts at Solano County was to show the need for microcomputers. I was surprised that with the sizeable investment in computing resources, there were only a handful of PC's scattered around the County and no plan to incorporate microcomputers in to the overall office automation plan.

My efforts concentrated on the performance improvements which could be made by distributing some of the processing out to microcomputers and off of the HP 3000. Several existing performance and environmental factors supported my claims:

The response time on our single HP 3000/68 was increasing daily as we approached 200 users, many of them using HP Word 3000.

Disc storage space was getting scarce as HP Word 3000 and HP Listkeeper users created, but never purged documents and files.

The number of transaction-intensive 3000-based applications being written for and installed on the 3000 was increasing and further degrading response time.

Users were becoming more sophisticated as to the availability of office automation applications available on the 3000 (I'd like to think that this was a result of our increased training effort) and were using these applications with greater frequency.

Users were becoming more sophisticated in their use of the office automation applications and created "custom" applications with these products which strained processing and storage resources. Some examples:

The Sheriff's Department developed a fingerprint identification system using HP Listkeeper.

The District Attorney's office used HP Listkeeper as a case tracking system.

The Customer Services department developed a case calendaring system for Superior Court using HP Word/3000 to create the documents and HP Deskmanager to distribute them.

The increasing use of office automation applications: HP Word/3000, Opticalc, HP Draw and EasyChart put additional strain on the overworked 3000 CPU and response time for these products became almost unacceptable.

Additional response time and downtime problems were increasing as remote users attempted to run HP Word 3000 over point-to-point data communications equipment.

I realized many of these problems could be resolved or ameliorated by moving many office automation applications off of the 3000 and on to microcomputer.

TRANSITION PATH: TERMINAL TO HP 150 TOUCHSCREEN MICROCOMPUTER

We decided to make our MIS managers the "test market" for the PC experiment and purchased the first HP 150 Touchscreen microcomputers to replace existing HP 2628A terminals for the 5 MIS Managers. The transition was not without its rough points but, basically, I was surprised that the managers all rapidly became PC "converts". Even our MIS Director, a card-carrying mainframe/minicomputer "techie" since the days of the Series III and before, became a champion of the Touchscreen PC.

A primary reason that we selected the HP 150 Touchscreen over other true IBM compatibles was the Touchscreen's HP Word 3000 terminal emulation capability. At Solano County, we have many departments with heavy dedicated word processing requirements. County offices are also geographically disperse and there are extensive requirements to share documents between departments with remote offices. HP Word/3000 solved the logistics problem but failed to perform efficiently and reliably over the data communications network. Microwave channels drifted, multiplexers overflowed, modems failed, data circuits died and users complained.

In fairness, HP never intended HP Word/3000 to operate over data communications equipment so we were the ones "stretching" the technology to serve our own requirements.

In addition to continuing to run HP Word 3000 on the Touchscreens, we installed HP Word/150, Advancelink, Lotus 1-2-3 and RBase 5000 PC software and worked increasingly in the stand-alone PC mode.

TRAINING ISSUES: TERMINAL TO TOUCHSCREEN PC

As a test case, the MIS manager group was a complete success. We all adapted readily to the working on the Touchscreens with our PC applications. It seemed that every day, someone would come into my office with another Lotus macro to share.

No formal training classes were given to the MIS managers as we were initially the only staff with Touchscreens and were still in the learning phase. As training manager, I was encouraged by this example and assumed that our enthusiasm for the new technology and willingness to "self-teach" would be shared by our users as they, too, received new Touchscreen PC's.

The motivation was certainly there. Many users had publicly voiced the desire to have independence from MIS, to control their own computing and be free from MIS CPU charges and inconvenient batch processing schedules. They would surely "take the micro and run" and have no need of our training classes.

As the 1985-86 budgets were approved, Touchscreen microcomputers were purchased in volume and installed in departments where on-line performance was the worst and the need to work in stand-alone PC mode the greatest.

Unfortunately, the average non-MIS department user did not take to the Touchscreens as readily and enthusiastically as the MIS managers had. Many continued to use only the terminal portion of the Touchscreen to run HP Word and the other 3000-based terminal applications they had been running on their old 26XX terminals. The 10MB 9153 disc drives served only as a convenient base to place the Touchscreen terminals on. Even users who had used terminals extensively and had received formal terminal training were confused. Calls for assistance with the new PC's increased.

One of the primary functions of my Customer Services Department is to staff the help desk, a phone line for user assistance with computer-related problems. We monitor the types of calls for assistance received by application type and we could see that the calls relating to the new Touchscreen´s were representing an increasing proportion of the calls received. These calls were enlightening. Some examples:

> User; "I´m working in HP Word." Help desk staffer: "HP Word on the 3000 or on the microcomputer?" Caller; "I don´t know."
>
> "I lost some Lotus 1-2-3 files." "Can you have Operations restore them from tape for me?"
>
> "The manual says I have two disc drives but I only see one!"

Obviously, my assumption that non-MIS users would self-train on the Touchscreen and operate more efficiently (and autonomously) was incorrect. The Customer Services staff met to assess this new need and develop new training classes to meet the need. I refer to these classes as Level 2 (see Figure 1, page A1). These courses included:

HP 150 Microcomputer, 4 hours

covering the following topics/areas:

online (to the HP 3000) versus stand alone operation of the microcomputer

information transfer between HP 150 and HP 3000

working with diskettes

installing software

information retrieval

use of the P.A.M. (Personal Applications Manager) facility

Beginning Lotus 1-2-3, 6 hours

Advanced Lotus 1-2-3, 6 hours

Beginning HP Word/PC, 18 hours in 3 sessions

Advanced HP Word/PC, 6 hours

We also negotiated for the resources to upgrade the Training Center equipment. We replaced the four 2392A terminals with Touchscreen microcomputers (on-line to the HP 3000) equipped with 2225A Thinkjet printers.

Note, also that we make attendance at the HP 150 microcomputer class a prerequisite to attending either of the applications classes.

TRANSITION PATH: HP 150 TOUCHSCREEN MICROCOMPUTER TO HP VECTRA MICROCOMPUTER

As our HP 150 Touchscreen microcomputer base grew, we, in Customer Services grew to love many of the features unique to this microcomputer.

As a CP/M, then DOS "purist", I initially regarded the Touchscreen/PAM features as novelties which would remain largely unused by most users. As a trainer, I grew to love the feature's simplicity and ease of use for the first-time microcomputer user and, yes, my DOS skills did become a bit rusty during the period when a Touchscreen graced my desk but I, too used and loved them!

Our users complained loud and frequently about the Touchscreen's lack of compatibility with the IBM PC and its clones; but, as the one responsible for installing, distributing and controlling the whereabouts of the County's microcomputer software, I was thankful for the non-standard MS-DOS format on the indestructible 3.5" diskettes. Also, in that Touchscreen environment, we did not spend hours on the phone troubleshooting a problem only to find that the root cause of the problem was a user-installed bulletin board provided utility!

Needless to say, standardizing on microcomputer software packages and providing regular training classes for this software was much easier given the limitations in software initially available for the Touchscreen.

I was able to standardize on HP Word/150 (then PC), Lotus 1-2-3 and RBase 5000 as the basic packages for microcomputer applications within the County. Even now, many departments still use their Touchscreens only for word processing and to access HP Desk and other terminal-based applications.

But technology advanced and as the price for the HP Vectra "AT" dropped below that of a similarly configured Touchscreen in the Fall of 1987, we made a decision to fill all new requests for microcomputers with a one of two standard configuration Vectra's:

the ES model 21 with monochrome monitor for "standard" use to replace the 640KB, Touchscreen/9153

the ES-12 model 42 with 2MB extended memory, 80287 co-processor, color monitor and mouse for users requiring large spreadsheets, graphics and as a future bridge to the New Wave or OS/2 environments

Both Vectra models were ordered with a 3.5" disc drive in addition to the standard 5.25" flexible drive. This was done to facilitate transfer of files created on the 3.5" media used by the Touchscreen microcomputer.

TRAINING ISSUES: TOUCHSCREEN MICROCOMPUTER TO VECTRA MICROCOMPUTER

Once again, we began the transition by replacing the MIS manager´s Touchscreens with Vectra ES-12´s. Once again, the managers were enthusiastic as spreadsheets could expand in size and recalculate faster, as graphics loaded and displayed faster and as print buffers carved out of the extra 2MB of memory allowed a 10 page document to print to the Quietjet while another program was running. But there were problems.

Where was the familiar Touchscreen? (learn to use the mouse). Where is a full-function PAM screen? "You mean I have to type out a DOS command to do copy and backup?" (learn to install these commands on the Vectra´s PAM program).

Even though the Touchscreen and Vectra were both microcomputers manufactured by the same company, many issues unique to the Vectra started to surface:

How to we perform the daily tasks like copy, backup and format? We used to just touch a box on the Touchscreen PAM screen.

How do we transfer files created on the Touchscreen to the Vectra?

How do we access HP Word 3000 documents?

Which word processing package should we use?

How do we move documents created on the Vectra using WordPerfect to HP Desk and mail or store them in HP Library?

We, in the Customer Services department could see that this would be an even more difficult transition for our users than the move from terminal to Touchscreen and we broke the project down into major tasks:

Standardize the Vectra's config and autoexec files and PAM screens to "automate" many of the common utility functions such as copy, backup and format. Yes, we were recreating the Touchscreen environment!

Identify the training needs unique to the Vectra, plan and schedule classes based on these needs.

Obtain four ES model 21 microcomputers to upgrade the Training Center.

We realized that we would need a training course on the Vectra, separate from the training course on the Touchscreen. We also recognized the need to provide a separate course on using DOS.

As of the current time, we offer the following courses for Vectra users. These are referred to as Level 4 courses on Figure 1, page A1.

Vectra Microcomputer, 4 hours

Beginning DOS, 4 hours

Intermediate DOS, 4 hours

Advanced DOS, 4 hours

These courses cover the following major topics:

Operation of computer hardware and interfaces

Working with diskettes

Software installation

Setup of applications in PAM

File management: storage, retrieval copy and backup

Hard disc directory management

MS-DOS commands

Creating and modifying autoexec, config and batch files using EDLIN

Working as a terminal using the HP terminal program

Additionally, those receiving a Vectra as their first workstation will generally also attend the HP Deskmanager course.
We plan to combine the Vectra Microcomputer and Beginning DOS classes into a single one-day course. We have found that there is generally a lag between the time users take the Vectra and DOS classes and without a basic review of DOS, users have had difficulty working with the Vectra.

The following application software courses are currently offered on the Vectra:

Beginning Lotus 1-2-3, 6 hours

Intermediate Lotus 1-2-3, 4 hours

Advanced Lotus 1-2-3, 4 hours

Beginning HP Word/PC, 18 hours/3 sessions

Advanced HP Word/PC, 6 hours/1 session

As the demand increases, we plan to offer courses in WordPerfect and RBase 5000.

TRANSITION PATH: MICROCOMPUTERS FROM STAND-ALONE AND POINT-TO-POINT CONNECTIVITY TO LOCAL AREA NETWORKING (LAN)

As part of our planning for FY 88-89, we have requested funding for a 12 node HP Starlan 10 network. As before, our plan is to install this LAN at MIS and research the training requirements by learning the system and developing a training plan based upon our experience.

Currently, the County Counsel department is the only user department to have requested a LAN (an 18 node Starlan with a bridge to the HP 3000) for this budget year but I know that the requirement for LANS will increase as the number of installed microcomputers grows.

Already, we have identified some potential areas for concern for the transition to LAN:

Will primary support for the management of the LAN be at the user department or MIS Customer Services level? If the support comes from Customer Services, how will we charge the user? If the support is at the department level, how will this person be selected, trained and compensated? In speaking with my colleagues at other California counties, it appears that support of a LAN of this size warrants close to a full-time position.

Which applications will be run "shared" or networked versus stand-alone?

How will data backup be accomplished? Who will be responsible?

How will peripherals be shared? Will a microcomputer server or HP 3000 using Resource Share be used?

Some of these training issues have been identified as Level 4 requirements on Figure 1, page A1.

As this topic of transition to LANS is really outside the area of concentration of my paper, I will not spend further time on it now. The topic is critical, however, and we will be devoting considerable resources to it in the coming year.

APPENDIX FIGURES

TRAINING REQUIREMENTS: TERMINAL TO PC (Figure 1)

Figure 1 on Page A1 graphically represents the training requirements necessary to support the current and anticipated workstation configurations at Solano County.

The training topic areas on this graph are cumulative; all prior level topics must be incorporated into training for the highest level of training required. For example, the user receiving a terminal as his or her first workstation, need only take Level 1 courses. The user receiving a Vectra PC as a first workstation must take all level 1,2 and 3 courses.

Level 1 courses include:

- **Beginning HP DeskManager**, 4 hours
- **Advanced HP DeskManager**, 4 hours
- **HP Listkeeper**, 4 hours
- **Data Entry**, 18 hours in 3 sessions
- **Opticalc**, 6 hours

Level 2 courses include:

- **HP 150 Touchscreen Microcomputer**, 4 hours
- **Beginning Lotus 1-2-3**, 6 hours
- **Intermediate Lotus 1-2-3**, 4 hours
- **Advanced Lotus 1-2-3**, 4 hours
- **Beginning HP Word/PC**, 18 hours in 3 sessions
- **Advanced HP Word/PC**, 6 hours

Level 3 courses include:

- **HP Vectra Microcomputer**, 4 hours
- **Beginning DOS**, 4 hours
- **Intermediate DOS**, 4 hours
- **Advanced DOS**, 4 hours

Level 4 courses include (proposed):

- **LAN Management**
- **Specific courses in working with networked versions of microcomputer software such as RBase System 5 multiuser**

WORKSTATION TRANSITION PATH (Figure 2)

Figure 2 on Page A2 shows the transition path of workstations at Solano County.

This path is chronologic, showing our workstation transition over time from terminal to LAN. As workstation hardware has evolved and become more powerful (and less expensive from a price/performance standpoint), we have purchased the newer model and incorporated it into our installed base.

This path is also applicable on a task-complexity and user sophistication basis. As the user's tasks to be performed become more complex, and as that user becomes more sophisticated in his or her knowledge of workstation capability, the workstation needed to support those tasks becomes more sophisticated.

From a training standpoint, there are several implications:

> We must maintain an ongoing training program for all levels of courses because all types of workstations are in use within the County.
>
> County staff move within departments and must be trained on the workstations in use within their current departments.
>
> Departments are constantly upgrading to new workstation technology and their staffs must be trained in the use of the new workstations.

Since our MIS Department is a chargeback agency, purchasing and then leasing workstations and peripherals to County departments, we keep these devices circulating through the departments on a "task/technology" matching basis. When a department outgrows a terminal and requires a microcomputer, we select the appropriate microcomputer and move the terminal to a department requiring its capabilities. In this way, we can maximize the "field time" of these devices, control inventory, minimize new purchases and lower the overall computer hardware costs to the County.

There are many other economies of scale we derive from centralized computer hardware management but those are outside the scope of this paper.

SUMMARY

Training the terminal user to be an effective PC user requires several actions:

Workstation hardware must be standardized to leverage the training effort over large groups of students. Standardization will also provide for a lower overall training bill for the organization as staff can move between departments and likely find the same (or similar) workstations and software in use there.

A Training Center must be established and maintained to provide for the ongoing education of the organization´s staff.

The concept of an ongoing training program must be "sold" (and continually resold) to senior management. The training manager is the only one in the position to know and sell the cost benefit (to the organization) of maintaining a quality training program.

As new workstation technology is brought into the organization, before it is widely distributed to the non-technical users, it must first be adopted by those responsible for training. The training staff must assess training requirements for the new technology and develop and offer applicable courses to the users at the same time as the new workstations are distributed to the users.

The Training Program must provide standardized courses on a regular basis to accommodate the schedules of staff to be trained. These courses must be offered at no or low cost in order to encourage attendance. We have found that it is wise, however, to charge a "no-show" fee in order to ensure full classes and maximize training resources.

Calls for assistance from the help desk or help line must be periodically reviewed in order to determine if existing training courses should be modified or combined or if new courses need to be developed.

A student/course database should be established in order to provide statistics on the training effort. These statistics can be used to show such trends as:

Period to period changes: How many sections of a particular course have been given over the previous year. How many students have been trained?

Which training courses are no longer desired/required?

Which courses need to be offered more frequently?

Which students frequently repeat the same course (is the student a slow learner or is the course ineffective?).

Which departments have never received formal training?

Which students are broadly trained and might be candidates for department level support contacts or future Customer Services staff.

Training the terminal user to be an effective PC user is an ongoing effort. The resources devoted to this training effort must increase as the technical complexity and capability of PC´s increases. The training staff must be "detectives" and actively seek out user training needs. Finally, the training manager must maintain a quality training program and continually "sell" the program to users and management.

TRAINING REQUIREMENTS - TERMINAL TO PC

MAJOR TOPICS COVERED:

LEVEL 4 - LAN
- LAN MANAGEMENT
- SERVER BACKUP
- PERIPHERAL SHARING

LEVEL 3 - VECTRA PC
- SOFTWARE TRANSFER
- P.A.M.
- DIRECTORIES
- MS-DOS

LEVEL 2 - HP150 PC
- P.A.M
- STORAGE/BACKUP
- RETRIEVAL
- INSTALL S/W
- DISKETTES

LEVEL 1 - TERMINAL
- HPDESK BEG/ADV
- MANUALS
- BASIC TROUBLESHOOTING
- LOGON/SECURITY
- KEYBOARD OPERATION

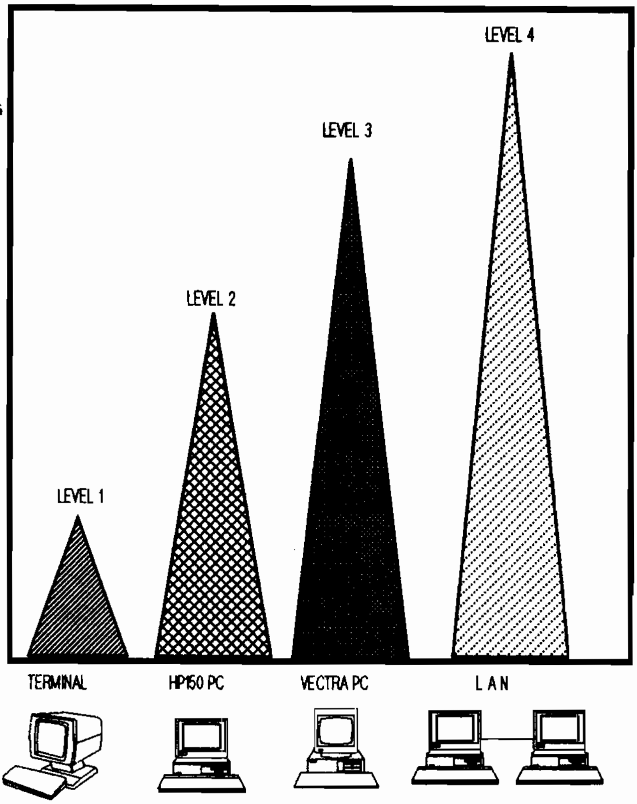
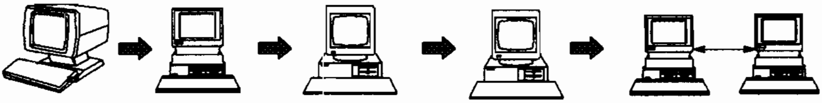


FIGURE 1

WORKSTATION TRANSITION PATH



TERMINAL

HP150 PC

HP VECTRA ES/MOD 21

HP VECTRA ES-12/MOD 42
2 MB EXTENDED MEMORY

L A N

TASKS:

DATA ENTRY

CLETS

ON LINE INQUIRY
HP DESK

WORD PROCESSING
(HPWORD 3000)

HP3000
APPLICATION
PROGRAMS

TASKS:

TERMINAL TASKS
PLUS:

STAND ALONE:

WORD PROCESSING

W/CONVERSION TO/FROM
HPWORD 3000

LOTUS 1-2-3

RBASE 5000

HP150 SOFTWARE

TASKS:

ALL TERMINAL TASKS
EXCEPT HPWORD 3000

STAND ALONE:

WORD PROCESSING
HPWORD PC
WORD PERFECT

LOTUS 1-2-3

RBASE 5000

APPLICATION SOFTWARE
AVAILABLE IN STANDARD
PC-DOS FORMAT

TASKS:

ALL TERMINAL TASKS
EXCEPT HPWORD 3000

ALL STAND ALONE TASKS
FOR VECTRA ES/MOD 21

LARGE LOTUS 1-2-3
SPREADSHEETS

DESK TOP PUBLISHING

HP NEW WAVE OR
OS/2 OPERATING
ENVIRONMENT

CD ROM

TASKS:

ALL TERMINAL TASKS
EXCEPT HPWORD 3000

ALL STAND ALONE
MICRO TASKS

SHARE DEPARTMENTAL
BASED INFORMATION

SHARE LOCAL
PERIPHERALS:
PRINTERS
PLOTTERS

CD ROM

FIGURE 2

Documentation: The Necessary Evil

Robert M. Gignac
Motorola Information Systems
9445 Airport Road
Brampton, Ontario (Canada)
L6S 4J3

Abstract

The systems we are developing today will become our 'foundations for the future'. The cornerstone of our foundations should be an array of clear, concise, and well written documentation. Do not misinterpret the title, documentation is far from 'evil', the keyword is 'necessary'. The 'evil' designation is in the eyes of the analysts and programmers in charge of developing our systems. Of the many steps involved in a project, documentation is the one we all agree is required, but is the one nobody wants the responsibility of writing, let alone maintaining. To further undermine our foundation, documentation is the area most likely cut from a project as costs rise and deadlines draw near.

1.0 Introduction

A tremendous amount of time, resources and knowledge must be expended in the development, programming and testing of computer based systems. The results of these efforts must be carefully organized so the myriad of details related to the programs within these systems can be recorded in a clear and orderly fashion. This means significant information about the programs and the systems must (not should!) be written and stored in a clear and concise fashion. The creation and maintenance of this information is called DOCUMENTATION. Documentation is a vital part of every system, and in order to build 'foundations for the future', documentation must be one of the cornerstones of that foundation.

Unfortunately, as vital as documentation is, it remains possibly the weakest link in the computer systems field. If you need confirmation of why this is so, perform the following informal survey of your programmers and analysts: ask them to list their 15 favorite tasks on a sheet of paper and when they are done, sit down and review them. More often than not, the words 'documenting systems/programs' will fail to appear on the paper. Why? Documentation isn't trendy, machine oriented, technically stimulating, and most programmers and analysts consider it beneath them to document. When your staff approaches the task of documentation with this attitude it is easy to see why documentation has been neglected, completely ignored, or written in a haphazard manner in many DP shops.

There are two hazards encountered when writing about the subject of documentation. The first hazard is the fact the term encompasses a variety of different forms, all having different meanings to different people. G. Prentice Hastings and Kathryn King in their book <u>Creating Effective Documentation for Computer Programs</u> make reference to the following levels of documentation: Reference Charts, Operator's Guide, User's Training Document, Student Workbooks, Reference Manuals, Management Guide, System Administrators Guide, Logic Diagrams, Microcode List, Technical Manuals, User's Manual, and Installation Guide. I am not about to cast judgement on how much of this documentation is really required or even practical to create. A decision on this matter would depend on the size of the system you are creating and the needs of your individual company. The second hazard arises from the subjective nature of documentation. Like anything subjective, changes in documenting procedures may make things better, worse, or leave them much the same - and often there are no easy answers about what to do. Every change or implementation of new ideas involves trade-offs - What type of documentation? Structure? Amount? Who will write it? - which are significant in determining what needs to be done. This paper is, therefore, by no means an attempt to provide definitive 'yes/no' answers. However, if this paper proceeds to equip you with the knowledge required to reach your own conclusions about why documentation is not evil, it will have done its job. The scope of this paper will be two areas of the documentation field that I feel are key, yet are often left incomplete, incorrect or nonexistent: Program Documentation and Application Software Documentation.

2.0 Program Documentation

It remains a fact that even today many programs are delivered for which there is little or no documentation to accompany them, or the documentation that does exist bears no resemblance to the source code. On the surface you could ask "So what?", because the program/systems will run whether or not the documentation exists. As long as the program runs, this remains true. Unfortunately, I have seen very few programs that ran forever without 1) going crash in the night, or 2) requiring some form of maintenance. Addressing the first issue, we all know that it will be late one Friday night that the XYZ analysis program which has run flawlessly for 27 months will abort with some cryptic Image error, requiring someone to dig out the program documentation. If we can find it, we are a step ahead of the game. If it really matches what the program is doing, your maintenance staff will love you. More likely, you will have documentation that is either: a) incomplete, or b) lacking the revisions Peter Programmer made last June before he left the company. Addressing the second issue, I have yet to see in my relatively short experience in the field (6 years) any program that has run for 2 years or more without ever requiring some form of modification. Fixes to old bugs, fixes to new bugs caused by fixes to old bugs, report format changes, company policy changes, etc. The reasons why you might change code are endless, but you can be sure that if your program is important it will need to be changed at some point. There is also a third possibility, which I have seen only once, that may be valid in your shop -- extremely accurate documentation of very poorly written programs. While not as severe a problem as the first two, it deserves some recognition as well.

Given all these potential problems, don't despair, you won't be the first to encounter them (or the last). However, to ensure that they don't happen more than once, we should discuss some solutions. The most obvious (and since it is obvious, it is probably the least effective) is to write standards and directives for what you expect to find in your shops program documentation and make compliance with these directives a requirement for continuing employment. We will assume you are doing this already, and if it is working, fine -- but if it isn't, we require some additional tactics. More humane and perhaps more sensible might be to seek out programming methodologies with built-in documenting enhancements. But don't be mislead here: structured programming by itself (regardless of language) won't solve the documentation problem (despite what your programmers say...). Choose whatever documentation method you feel is best for your shop, then it is up to management to insist that the guidelines are followed. It still appears that after all this time there is no mechanical substitute for old-fashioned management control.

2.1 Program Documentation - Implementing Change

In order to get your DP staff to follow the new management guidelines, you will have to get them to change the standard and widely held view of documentation: "Those who can, do, those who can't, document". Do not expect this change to happen easily, as people resist change in any number of ways, and for many different reasons. Key among them are lethargy and fear. Phillip Metzger in his book

Managing Programming People outlines the following options as possibilities:

1) Serious Threat: "Do it or I'll kill you"

2) Appeal to Self-esteem: "You don't want the people in Linda's department to make us look bad, do you"

3) Opportunity: "We finally have a chance to look into this new opportunity to improve ourselves"

4) More Opportunity: "Here's a chance to blaze a trail for the rest of the department"

5) Bribery: "Do this and I'll remember it when salary review time rolls around"

On the surface, these options appear quite humorous, but they are some of the methods you may have to use if you are to change the opinions of your staff towards documentation. Believe it or not, people will actually resist the opportunity to increase their skill levels or improve their credentials in this area (ask my former supervisor). In part, this may be due to the mistaken belief that good documentation skills are not seen as a marketable asset, as are courses in structured design, database methodology, 'C' programming, etc. As an analyst or manager, you may face an additional problem. You may have programmers reporting to you (whom you can convert), but you in turn report to someone who may hold the same beliefs as your programmers. It will probably be easier to get your technical people to try new things than it will to get management to join. Management's reasons may be as follows:

1) If the group spends too much time on this, other projects may fall behind (yet time for this should have been planned in advance...)

2) How do we know it will do any good?

The best way to answer managements concerns would be by analogy. You have to look at program documentation as insurance. If nothing ever goes wrong at your site then the effort may appear to be wasted. On the other hand, when things do go wrong (and they will...) your first line of problem solving will be a referral to the program code and the accompanying documentation in order to: a) find the cause of the problem, b) fix it, and c) get the system rolling again. It is because of this fact that documentation is often viewed in nebulous terms, you can't tell management how much value documentation has until a crisis arises, and by then it is too late to start documenting.

2.2 Program Documentation - Assembling the Material

Hopefully, we have determined that program documentation is actually required. We will assume it doesn't exist, and that attitudes on the part of programmers and management can be changed. Just what kind of information should we create? The following chart summarizes one of many possible alternatives for creating a program documentation manual.

1) Title Page

The title page should contain the program name, system of which the program is a part, original programmers name, and the date released to production.

2) Revision Page

The revision page is required to document the history of the program. Contents should include the name of the original programmer, date released to production, estimated time to complete programming. On this page in chart form, provisions should be made to document all subsequent revisions, descriptions of them, names of the parties responsible for them, and the date the revised program was released to production. In order to ensure this page is always up to date, do not allow programs to be moved into the production environment until it has been verified that the revision has been documented.

3) Abstract

The program abstract should contain a general purpose and description of the program, frequency of use, input and output files, subprograms called, and a list of programs that are prerequisite for this one to run.

4) System Flowchart

The system flowchart documents the flow of data through the system, providing a visual means of identifying input and output files used in the required steps of the total processing cycle. (There will be those who feel this process is becoming obsolete.)

5) Logic Description — This section should provide a detailed description of the program including special editing performed, sequence checking, reasonableness checks, tables used in the program, special forms required, and operator instructions. The detailed program logic must be illustrated using program flowcharts, decision tables, or pseudo code.

6) Test Data — A listing of the test data used in testing the program, and a sample of the program output should be provided in the documentation manual. Test data should be sufficient to test all routines within the program.

At Motorola Information Systems we are using a combination of the items mentioned in the above list as shown in Fig. #1, #2 and #3 (see appendix). These documents must be completed for every program released to production, and no program will be run in the production environment before these pages are verified to be complete. These pages, as good as they may be, are only half of the battle. The remaining program documentation must be carried out in the program code itself.

2.3 Program Documentation - Using your code

As mentioned earlier, some programming languages lend themselves to documenting. COBOL for example can be somewhat self documenting if certain standards for documentation are enforced. Require COBOL programs to contain a purpose, description and brief history in the REMARKS section of the code. Require that sections or paragraphs be commented if the paragraph performs complex routines or calculations that would not be obvious to someone unfamiliar with the code. Require that all COPYLIBS and SUBPROGRAMS be identified as in Fig. #4 (see appendix).

In case you feel that I am the only person 'crazy' enough to feel this way, I offer the following quote: "In my opinion, there is nothing in the programming field more despicable than an uncommented program. A programmer can be forgiven many sins and flights of fancy; however, no programmer, no matter how pressed for time, no matter how well intentioned, should be forgiven for an uncommented and undocumented program". This quote comes directly from Edward Yourdon, author of Techniques of Program Structure and

Design. Do keep in mind however, that commented code is not an end to itself, as good comments are not a substitute for bad code, nor is good code a substitute for lack of comments. Program code obviously tells us what the program is doing, but it cannot tell us why it was done in a certain fashion. Before you decide that code documentation will be the key to solving your problems, be prepared to hear the following excuses from your staff:

- I don't have enough time

- My program is self-documenting

- Any competent programmer can understand my code

- This is a one shot program, its not worth it

- The program will change dramatically during the testing and debug phase, so any documentation will be useless by the time the program is finished (if this is the case, perhaps you should question their design skills before they start to code?)

- I understand the code, I'll be here to fix it

- My programs will take too long to compile

- Who will read the stuff anyway?

We have all heard these arguments before, and perhaps we have even used one or two of them on occasion. In order to combat them we may choose any of the techniques for change outlined earlier by Metzger, or we may choose to implement documentation as a philosophy across a department. Actually putting program documentation methods into practise is not that difficult if it is kept in mind at all times (perhaps a system welcome message that reads 'Have you documented your code today?'). Good documentation habits are generally best exemplified by personnel who work for consulting firms. Reassignment to another task is a common occurrence, and for the success of the project (and perhaps the firm), it is imperative that the next person be able to pick up the system where the last one left it. Arguments will be raised here as well, because people feel they work in a relatively stable environment, so this precaution is not necessary. One only has to look at DP turnover rates to see why it is necessary. The average length of employment with one firm is less than three years, and the easiest way to turn programs or systems over to new people is with decent accompanying documentation.

2.4 Program Documentation - Reducing Maintenance Costs

Programs spend most of their life being maintained. Often, considerably more time and money is put into extending and changing programs than was spent at the initial development. If this surprises you, it shouldn't. New systems and their associated programs change the environments in which they are used. In turn this changes the way they work, and when work habits change, changes in the system are a natural result. Barry Boehm in his book Software Engineering Economics reports that DP shops are currently spending over 50% of their budgets on maintaining their existing systems (see Fig. #5 in appendix). Over the past 10 years this figure has increased by about 25%, and will probably continue to increase in the future. If you wish to reduce your costs (and who doesn't?), you can use reducing your maintenance costs as a selling point for program documentation. Below is a list of why maintenance costs are so high, and it is easy to see how program documentation may reduce these costs.

- Often programs are released to production that still have a significant number of bugs. Due to this, what is often called maintenance is really just an extension of the testing phase.

- When maintenance is required, the original programmer has often left the company, or has been reassigned to a different project.

- Programmers do not often view maintenance as glamourous work.

- Most people have difficulty understanding other peoples code.

- Documentation that accompanies most programs is just short of awful. Some testing in university settings has indicated that maintenance programmers would be better off removing all of the comments accompanying a program and then trying to find bugs or implement improvements. Because of this, many firms are now paying the price for poor documentation standards of the past, as their maintenance times and costs increase.

2.5 Program Documentation - A Dissenting Opinion

As with most concepts in the systems field, there are people who are 'for' the concept and those who are 'against' the concept. I feel I would be remiss if I didn't at least address the viewpoint of the 'against' delegation. John

Boddie in his book Crunch Mode, approaches program documentation as follows, "On some projects there is a rush at the end to produce 'program documentation' -descriptions of the code in the system. This is done in the name of maintenance. What it is, really, is stupidity.". On this issue I must disagree. If the project was properly planned and the documentation completed at each step in that plan, they wouldn't be running around at the end of the project trying to complete program documentation. Mr. Boddie goes on to state that the original programmers design documents, plus the comments that were put in the code should be adequate enough for the maintenance staff to pick up the system and maintain it. "These comments are the 'program documentation'", and project leaders will insist on it as good programing practice. Unfortunately, in the past, project leaders have not insisted on this, and many do not to this day. As for the design documents and program comments being adequate program documentation, could you imagine trying to piece together the relationship of a complex system from the program design documents and the source code?

Don't let your staff, or your management try to avoid the issue of program documentation by using any of the excuses mentioned in this section. Any program or system that is of any value (and why would we bother to create them if they weren't?) will remain active for some period of time, increasing the odds of some other individuals coming into contact with it. Perhaps one of the best ways to impress the importance of this on a young programmer is to give them a 'rats nest' program to maintain, debug and modify (you know, the kind we all used to write). If this is done to them early in their career it can have a strong and beneficial impact on their programming habits. This in turn will only make things that much easier for you to convince them of the benefits of program documentation, and they in turn may help you to convince the rest of your staff.

3.0 Application Software Documentation

Application software documentation (often referred to as the 'users manual') serves as the primary interface between the end user and the application software. Despite the importance of this documentation as a factor in both program and system success, software maintainability, proper system use and user satisfaction, application software is often paid little more than lip service by DP departments. Application documents have long been considered evils of doubtful necessity, and because of this, the manuals that are produced often try the users patience. It would appear that when programmers are good, they are very good; but when they write, they are terrible. The reason for bad writing getting out is the same as for bad programs getting out: tasks aren't planned well enough, plans aren't executed well enough, and the results aren't tested well enough. For these problems to exist, the finger must point at management for letting poor writing and documentation get by them.

3.1 Application Software Documentation - Inherent Problems

The Audience:

It is generally well known that most occupational reading is 'reading-to-do' rather than 'required reading'. Many people only use application documentation to improve performance of seldom performed tasks. Due to this, if the documentation is difficult to understand, users may abandon the written material in favor of alternative methods: trial and error, consulting more experienced users, or forgetting about the whole thing if possible. Users view the application documentation the same way we view documentation from companies whose software we use. If the documentation is poorly written or contains errors and inconsistencies, we attribute the same negative quality to their software. If we feel this way about the documentation we use, why shouldn't the end users feel the same about the documentation we provide for them?

Structure Differences:

The problem that arises here is due to the way documentation is created, especially in smaller DP shops. In many DP shops, the programmers or analysts are responsible for creating the application documentation. Unfortunately, when there are multiple systems being developed by different people, there will be different styles of documentation produced. Differences will appear in terms of layout, scope, wording, technical orientation, etc. Ideally, having access to a technical writer would help simplify the problem. In reality, since most shops cannot afford this luxury, documentation standards should be communicated to all responsible parties so consistent documentation will be produced.

Inadequate Current Documentation:

One of the most prevalent problems with the current state of documentation is the amount of it that is missing (whereabouts unknown), incomplete (lacking relevant information), inaccurate (missing latest revision), or obsolete (program no longer in production). Existing user manuals often lack a sufficient number of relevant examples to accommodate the needs of users. Error codes may go unexplained, and recovery procedures in case of error may be inadequately described.

Resistance to Document:

This topic has been discussed so many times that we should be able to abandon it by now (see section 2.1 on implementing change). DP personnel involved in the software development process are often those responsible for documenting the systems due to their higher understanding of the end product. The problem is that writing is one of the least interesting software related activities and little linkage is perceived between improved documentation and the organizational reward structure. Another part of this resistance to document may come from the basic educational system our programmers are now coming from. In a College or University setting there is no incentive to document your programs as you are the only person who ever has to deal with them. Users manuals are not required, and the whole issue of how educators view documentation can be summed up in a discussion I had with one of my college professors during a 3rd year systems design course. Being naive as I was at the time, I asked Ivan Chapman just what we would do with ourselves once we had been hired by a firm and had completed computerizing every possible activity known to mankind. His response: "Then you document". This attitude clearly makes documentation look like an unnecessary task, something to do once everything else has been completed. It is only in the newer systems texts that the concept of complete system documentation is being covered, and in fact, there is now an entire body of texts dedicated to the topic of creating documentation for computer systems. Eventually, this trend will filter its way into the educational system, and when it does, we should finally be able to hire programmers who do not view documentation as undesirable.

Inadequate Managerial Planning:

Often there is a perceived lack of managerial guidance, policy, support or review of documentation efforts. Efforts to minimize the software development time and cost may occur at the expense of perceived minimum benefits from documentation activities.

Lack of Testing:

You must schedule writing, editing and rewriting of documentation as carefully as you schedule design, programming and testing for the code. Documentation should not be left to the last two days before system delivery. If possible have your documentation written by people who like to write and are proficient at it rather than by the programmers who wrote the code (and who probably don't want to anyway). As well, you must test your documents. No, you didn't misread that last sentence -- you must test your documents. This can be done through the use of structured walkthroughs and review sessions. The user manual is really the only tangible item that you deliver to your users, and how they view it will often be how they view your system. Test your document by giving copies of it to your systems people who had nothing to do with the design or programming of the system and see if they can follow the logic. If everything makes sense, turn them loose on the test system, as systems people love to try to crash software and they may try things the users wouldn't think of. As well, use key personnel from the user areas if possible, as their understanding of what their system is supposed to do may expose flaws in the design, or highlight areas that need to be more clearly defined in the manual.

3.2 Application Software Documentation - Putting it Together

In order to create good user documentation, we must begin by asking questions. Who will be reading this? How much to they know already? What do they need to know to do their job? What aspects will be confusing to them? Given the task we have to complete, what information should we provide the user with? The following chart summarizes one of many possible alternatives for creating application software documentation.

1) Introduction

The introduction should contain the purpose of the system, objectives it accomplishes, and relationships with other systems.

2) Equipment

If possible, provide pictures of the work environment the user will be in. There are still many cases where we install systems in areas where terminals, printers and modems are foreign objects.

3) Operation

Provide the user with brief descriptions of the following items: Terminals, keyboards, printers and modems. Descriptions should include how to turn all equipment on/off and operating features of each device.

4) Using the Terminal

This section will cover the basic operations required before the system is active. It should cover basic user questions such as: How do I sign-on the system? What are function keys? Who do I call if it doesn't work? How do I sign off the system?

5) System Features — This section will comprise the majority of the user manual. Explanations of the system menus, types of operations available, explanations of how each transaction works, limitations and security in the system, processing flow, numerous relevant examples, where to turn for help, descriptions of all forms/screens/reports used, and what the user responsibilities are.

6) Error Recovery — Even though it is difficult and time consuming, all possible errors should be described in tabular fashion, listing symptoms and cures. Problems indicitive of hardware should be separated from those associated with system problems and application problems.

7) Hardware Maintenance — It is surprising that many people feel computer equipment needs no care. This section might include basic maintenance the user can carry out (cleaning screens and keyboards, adding paper to the printer, changing printer ribbons, etc). As well include a list of items to be referred to the service department and appropriate contacts when things go wrong.

As I stated earlier, the above list is only one possible setup for a user manual, your own needs will dictate your end result. Once the design has been chosen, there are still various hurdles to overcome in this process that will directly affect the creation of your manual, and they are listed below:

1) The orientation of user manuals should be to work functions where the terminal is just a tool, instead of a manual solely about terminal procedures.

2) Some familiarity with the subject matter should be presumed. This allows entire sections to be devoted to specific tasks, such as, "How to perform a Query", "How to perform a Delete", etc.

3) For ease of training, pictures of screens, keyboards, printers, and other equipment should be included near the beginning.

4) References to other manuals are confusing; any situation that is not 'normal' to a user usually results in a request for assistance.

5) Jargon, mnemonics and excessive abbreviations should be avoided.

6) If the same physical screen layout is used to perform more than one procedure it is better to repeat it than refer to another section; this ensures that a single section can cover an entire procedure.

7) Any reference to function keys should be emphasized by bold type or preferably, a drawing of a key top. Rather than, "When a field has been entered - press 'SEND'", it may be more effective to have:

   "When a field has been entered - press |SEND|

3.3 Application Software Documentation - Perceived Benefits

In the abstract for this paper, I mentioned there are benefits to be gained by maintaining accurate documentation. While one may not be able to assign a dollar value to all of them, the list below covers some of the key benefits.

Cost Savings:

While good documentation will in fact save you money, it is not always obvious how much. As mentioned earlier in the analogy regarding insurance, the cost savings may only be realized once things start to go wrong. In the case of a software firm which produces documentation to accompany its products, the cost savings may be viewed as money not lost through sales. If you had purchased a piece of software and the documentation was so poor it made the program unusable, would you recommend it to someone else? Software with good documentation gets recommended, therefore, if you produce software for the marketplace, it is worthwhile to spend the time and effort to produce quality documentation.

I would like to be able to tell you that every hour you spend in documenting programs/systems would yield you a cost savings of $15.00-20.00 but I cannot. Well designed documentation will help facilitate efficient and effective software development and will decrease training, operation and maintenance costs. In addition, having current documentation of software under development can reduce the risk of duplication of effort by your staff.

Managerial Benefits:

Documentation will increase the flexibility of managers in dealing with turnover or reassignment problems with respect to both end users and system staff. Given the traditional rates of turnover, the benefits should be obvious.

Software Marketing Tool:

Presence of comprehensive, understandable documentation attests to the quality of the related software, and can lead to favorable user beliefs concerning system integrity and reliability. The best conceived, written and implemented system will fail if the accompanying documentation renders it useless. On the other hand, excellent documentation can make a somewhat limited system appear to be far better than it is. Although some would feel this applies only to companies producing software for the marketplace, it impacts on systems developed for internal use as well.

Improved Communication:

Documentation can serve as an important tool for communicating within and between phases of a software project that is split among different groups. Documentation can be used as a quick refresher of both user and DP staff memories, and serve to lessen the potential for conflict and misunderstanding between users and DP staff, as well as between different groups on a software project.

Vehicle for User Participation:

Documentation provides a common baseline for discussion within and between groups. In fact, many DP shops (Motorola included) are currently riding a trend to allow the end users to participate in writing the users manuals for systems they will be using. Participation such as this can stimulate user feedback, morale, commitment and confidence in the software the end user will eventually receive.

4.0 Where do we go from here?

Regardless of your role, be it the programmer of a specific software application, programming mangers, DP director or a technical writer, you must have a good understanding of the five primary ground rule for documentation.

1) In order to solve a problem rather than contribute to it, you must first recognize and acknowledge that it exists.

2) Both technical and user documentation must include sufficient information to be used as both reference and instructional material in order to be considered valid.

3) Writers of computer documentation are instructors. Therefore, they must understand the needs of the people they are going to write for and the level of detail required to satisfy them.

4) Every project must be treated as a training assignment in order to maximize the instructional value of the content and the context of the documentation.

5) The eventual success of documentation depends on the writer's abilities and the company's willingness to provide end users with sufficient detailed and instructional information.

It would probably come as a major surprise to many writers, DP managers and programmers that their documentation often fails to meet the needs of the user. In many cases the documents were never tested or subjected to a formal/informal review process. A study cited by Hastings and King revealed that over 85% of all supportive documentation offends the intellect of the end user while failing to do what it is supposed to -- instruct.

Data processing departments generally sense that something is wrong when systems start to fail after implementation, but rarely do they associate this problem with their documentation. This should not be surprising because these are the same people who have the attitude, "Documentation is just a necessary evil and no one is going to read it anyway!". Take a minute to think about that, for if the people who create the documentation have this attitude, who would want to read the results of their documentation process?

4.1 A little commitment please...

It cannot be stressed enough that the documentation effort must be treated as an integral part of the system development process if it is to support the product/systems. DP departments must have a commitment to turning out quality documentation for every system they produce. Unfortunately, this commitment must be more than a mental attitude; it is knowing the tasks to be performed and who will be performing them when the project is started. True commitment requires taking the time to give everyone involved a complete understanding of what is expected of them and refusing to accept anything less. As overused as the term "management control" seems to be, the push for better documentation must come from the top, it will not happen if left to the programmers. Once a dedicated effort is begun to improve the documentation standards, enforce proper and consistent compliance, improvements will certainly be seen. Documentation is not 'evil', but a necessity that we can no longer afford to ignore.

DOCUMENTATION: THE NECESSARY EVIL (APPENDIX)

Fig. #1

```
                Motorola Information Systems
                    Program Documentation
-----------------------------------------------------------

Program: SMCRP075 (Daily Widget Counting)    Eff: 01/01/88
                                            Page: 1 of 3

Input:   INTRANS.FLS.PROD (Verified Transactions)
         METTRAN.FLS.PROD (Metric Conversion File)

Output:  AUDTRAN.FLS.PROD (Audit Transactions)
         REPORT;DEV=LASER,10,4;CCTL (Widget Count Report)

Database Files:

         SYSDB.DBM.PROD (System Database)  Read Add Chg Del
         - SYS-CTL-DTL  (Control File)       X

         MTLDB.DBM.PROD (Materials Base)
         - MTL-IMF-MST  (Item Master File)   X    X
         - MTL-SCF-DTL  (Cost File)          X        X
         - MTL-OBS-DTL  (Obsolete File)      X            X

         PCSDB.DBM.PROD (Production Base)
         - PCS-PIF-MST  (Purchased Items)    X        X

Frequency: Daily

Prerequisite: SMCAN070 (Production Analysis)

Special Forms: N/A

Additional Resources:

         SORTFILE;DISC=450000;DEV=14




 Written by:                       Date:

Approved by:                       Date:

Approved by:                       Date:
```

Fig. #2

Motorola Information Systems
Program Documentation

Program: SMCRP075 (Daily Widget Counting)   Eff: 01/01/88
Page: 2 of 3

Purpose: This program will access the validated transaction file and access the database to verify inventory levels in the distributed stockroom. Items that fall outside control levels will be reported.

Input: INTRANS.FLS.PROD (Validated Transactions)
METTRAN.FLS.PROD (Metric Conversion File)

Output: AUDTRAN.FLS.PROD (Audit Transactions)
REPORT;DEV=LASER,10,4;CCTL (Widget Count Report)

Reports: Daily Widget Count Report - 4 copies

Langauage: Informix-4GL

Estimate: 2-3 days

Frequency: Daily

Process Flow:

Access validated transaction file, search item master for matching key, if exists, update quantity counts, check for cost changes, see if item exists as suspected obsolete. If below safety stock levels access purchased item file and issue order message.The printed report will contain the Item-nbr, Qty Used, Qty on hand, Qty on order, Value of stock in-house and on order, and a warning message if the item is suspected obsolete.

Written by: Date:

Approved by: Date:

Approved by: Date:

Fig. #3

Motorola Information Systems
Program Modification Tracking

Program: SMCRP075 (Daily Widget Counting) Eff: 01/01/88
Page: 3 of 3

System Applied To: Codex Canada ____ MCS ____ INT'L ____

Module: ____________________

Modified Program/Form/Menu/Job: _________________________

Screen (If applicable) : _________________________

Program (If applicable) : _________________________

MSR Reference Number : _________________________

Effective at Release : _________________________

Discontinued as of : _________________________

Due to (KPR,MSR,Release): _________________________

Production Release: Codex Canada ____ MCS ____ INT'L ____

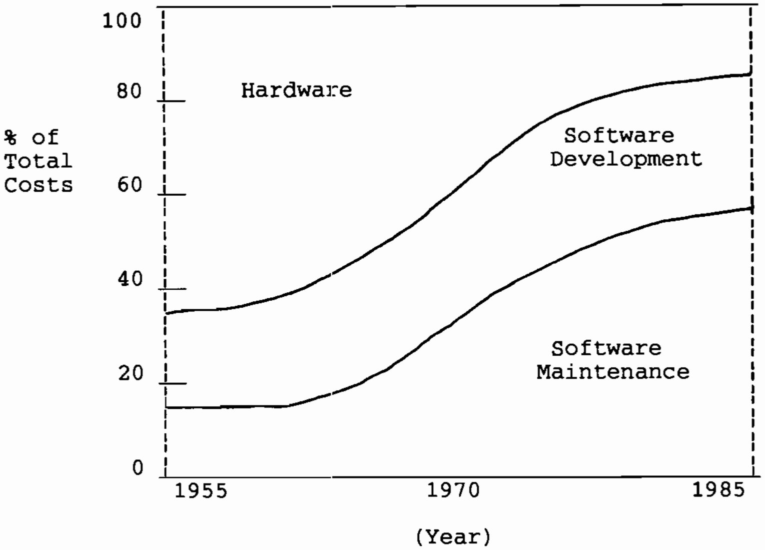
Release	Applied		Responsible	Which Accounts?
	Y/N	Date		

Fig. #4

```
$CONTROL DYNAMIC,BOUNDS
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SMFDR225.
 DATE-WRITTEN. MON. SEP 17, 1987,   2:12 AM.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. HP3000.
 OBJECT-COMPUTER. HP3000.
 SPECIAL-NAMES.
    CONDITION-CODE IS CC.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
$PAGE
**********************************************************
*               INVENTORY CONTROL SUBSYSTEM              *
*                                                        *
 01   PROGRAM-IDENTIFICATION.
     05 PROGRAM-NAME        PIC X(8) VALUE "SMFDR225".
     05 PROGRAM-VUF         PIC X(8) VALUE "A.12.01".
*                                                        *
*    PROGRAM NAME - SUPPLY/DEMAND INQUIRY                *
*    MODULE       - MCS                                  *
*    VIEW FORM    - CUSO                                 *
*    FUNCTIONS    - ADD,CHG,DEL,INQ                      *
*    SUBCOMMANDS  - NONE                                 *
*    SUBPROGRAMS  - SMFMI200, SMFMI210                   *
*                                                        *
*      THIS PROGRAM MAINTAINS INVENTORY COUNT RECORDS    *
*                                                        *
*    FILES           TYPE    GET PUT DEL UPD SRT I O I/O *
*   --------------- -------- --- --- --- --- --- - - --- *
*    MTL-DMF-DTL     IMAGE    X       X   X              *
*    MTL-IMF-MST     IMAGE    X           X              *
*    MTL-OIF-MST     IMAGE    X   X   X   X              *
*    SYS-MSF-DTL     IMAGE        X                      *
*    SYS-TGF-MST     IMAGE    X   X       X              *
*    COUNTERK        KSAM                             X  *
*    AUDITFLE        MPE                           X     *
**********************************************************
```

Fig. #5

Hardware/Software Cost Trends



Source: Barry Boehm, Software Engineering Economics
Prentice-Hall, 1981

References

Boddie, John
Cruch Mode
Yourdon Press
Englewood Cliffs, New Jersey
1987

Gore, Marvin
Stubbe, Jim
Elements of Systems Analysis
Wm. C. Brown Company
Debudue, Iowa
1975

Hastings, C. Prentice
King, Kathryn J.
Creating Effective Documentation for Computer Programs
Prentice-Hall Inc.
Englewood Cliffs, New Jersey
1986

Hedin, Anne
Unburden the User
Data Processing Digest
Vol. 31 No. 3 (March 1985)
Los Angeles, California

Metzger, Phillip
Managing Programming People
Prentice-Hall Inc.
Englewood Cliffs, New Jersey
1987

Shelly, Gary B.
Cashman, Thomas J.
Business Systems Analysis and Design
Aneheim Publishing Company
Fullerton, California
1978

Yourdon, Edward
Techniques of Program Structure and Design
Prentice-Hall Inc.
Englewood Cliffs, New Jersey
1975

The Secrets of Project Management

Robert Mattson
9545 Delphi Road. S.W.
Olympia, Wa 98502

Introduction

So much of the work we do in "systems" is in the form of "projects." The problem seems to be that there are a lot of ways a systems project can get messed up. I've been challenged by the nature of such projects for a number of years. From this I've concluded that there are some key concepts project managers need to understand and focus on to succeed on system projects. Therefore, the first goal, in this paper, is to pass on these "secret" concepts as "food for thought." The second goal is hopefully to provide something that will help further the success of your projects.

The Seven Secrets

With those goals in mind, I'm going to discuss seven "secrets" of system project management. I don't claim that these are the only important concepts/secrets. In fact, there are so many different challenges in system projects that no single set of answers will ever exist. However, I'm convinced that understanding and focusing on **these** seven "secrets" can result in many significant benefits. Further, my experience and research has led me to the conclusion that a lot of experts are missing key points or focusing on the wrong ones. To the extent that I haven't seen the points that follow discussed or emphasized...I call them "secrets." Finally, the "secrets" are applicable to any type of project although my emphasis throughout is that of a system project. So, let's get started at looking at these "secrets."

Understanding "Goals" and "Success"

Understanding "goals" and "success" is where system projects should start...unfortunately I believe, too few do!

Let's take the issue of "goals" first. A common error is to focus on the goals of a "systems" nature over those of the "business" and "user." System nature goals have to do with things like "trying a new language", or "ease of coding". The user's goals are usually oriented toward doing a "business'" or "organization's" function better, faster, easier and cheaper. The key is to understand what this means for the user! Too many times I see the key systems person on a project unable to express these "user's goals" in users terms. I believe this situation accounts for much bad press about how systems people cannot "understand" or "talk" in terms of the "user". Therefore, we need to focus on and clearly understand the user's goals. We, also, need to make these a prime driving force in system projects.

Related to the area of "goals" is the understanding of the meaning of the word "success." I've made the strong point in previous paper, Why System Projects Don't Quite Succeed, that each key player in the system project "game" measures the "success" score differently. The following is a chart showing how key players usually measure "success".

	Functionality ("User Goals")	Time/Cost	Internal Quality	Style
Users	80%	10%	0	10%
"Programmer"	30	40	30	0
User Mgmt	15	80	0	5
Top Mgmt	5	80	0	15

Note the differences in how the various groups and people measure success. Of special significance is how much weight management places on time and cost in measuring success. Equally, note how the user places an equally disproportionate weight on functionality. The significance of these differences is that if you please the user to the detriment of time and cost then many times the project will be judged less than a success by top management. Conversely, there are many completed systems projects that top management thinks were very "successful" which resulted in achieving only a marginal functionality. Further many of these systems may have very high life cycle costs. Top management based its belief on the fact these system projects were on time and on budget. Many "PC" projects are like this.

One may argue with the above percentages...but that doesn't really negate the key point. That is, different people view "success" differently. This knowledge doesn't necessarily solve every trade-off problem a project manager faces. It does ,however, help us understand how different players in the project will judge the project and by association those working on the project team. This knowledge is very important in understanding the "politics" of projects. With it, we can better balance our actions, goals and emphasis to achieve the result we desire.

So the first secret is:

> **One needs to understand a system project's many goals and measures of success in order to maximize desired results.**

People versus Techniques

We now have a clearer understanding of the meaning of "goals" and "success" in system projects. So, let's focus on how we can achieve them. Our first step is to explore the people versus technique issue. To do so, we start with what appears to be an "assumption" hidden in a lot of literature on project management. This assumption is that with the application of the "right" project management "techniques", one can make any project a success. This may sell books, but from my experience and observations it's not that easy! In fact, other things being equal, give me a person with "right" knowledge, skills, experience and attitude over any magic project management techniques.

What I am saying is that the first assignment and emphasis for the software project manager is to find the right people rather than focusing on "technique." Given the right people the chances for success go up tremendously. Conversely, without the right people chances for success are less than good no matter how great the project management.

Unfortunately, most managers appear to have less than full leeway in finding, selecting and getting rid of people on projects. In addition, our knowledge of how to find and select the person we need is less than perfect. It is when we are faced with this common reality, that we'd better understand and know how to apply all the rest of the secrets of project management and any other useful techniques. But let us not lose sight of the fact that the "right" person is worth a pound of techniques.

The place you can really notice the lack of attention to the value of people is in the area of "people resources".

Books on system project management almost universally gloss over the fact that there are ten to one or more differences in capabilities of people. In other words, the time it will take and the very quality of the product in systems is intimately tied to who does it. The nature of most system tasks and system personnel differences is such that no real assumption can be made about how long it will take an "average" systems person to do it. Yet, over and over again I see project management books or lectures treat system people as units such as "programmer" or "system analyst".

A related issue has to do with the commonly held belief of "staff or personnel development." Many managers' performance is rated on their success in developing personnel. Although, it may be heresy, I'm going to suggest that our ability and knowledge about how to "develop" people may be far less than we commonly believe. In addition, on many projects, the time and resources are just not available accomplish the needed development. For validation of this idea, try looking squarely at how much time and energy you have spent trying to "develop/change" a person. How successful was it? Was it really a good "business decision?" I would suggest that project managers may be way ahead to spend their time figuring out how to get "wrong" or "unmatched" people off their projects rather than trying to change them.

So the second secret is:

The "right" people are more important than "right management" techniques in system projects.

Communication

A great deal of the problems that arise in projects stem directly from "communication" related behavior that is in conflict with human limitations. As a consequence, people up and down the project organization "think" they've "communicated" "things" successfully when in fact they haven't. The "things" that are communicated in a project are major items like coordinating directions, specifications and requirements.

If we could, I'd like to magically measure the number of times in projects that an action isn't taken because the person never "heard" the request. Then I'd like magically to measure all the time spent by people doing the task as they thought someone wanted it done, but which later was found to be not on target. The number of times and more importantly the consequential cost of these two situations would be very significant. In fact, many projects fail or are much less than successful because of the poor quality of the communication going on.

Thus from this standpoint, project management is largely about how we communicated what, who, when, why and how much. It's about how successful we are in communicating up and down and across the project organization.

There are many challenges when one is trying to improve communications. For a discussion of many of these see the paper Communications - Why do Systems People Have Such a Hard Time With It?. Although there are many problems related to communications, the heart of many of the solutions involve an emphasis on written over verbal communications. This especially applies to system project management... usually it's the lack of clear "hard copy" of various kinds that leads to many of the projects problems.

So the third secret is:

Good project management must include good communication.

Task Management

There is no more key concept in the "management" of projects than that of understanding "task management." Projects are made up of a whole bunch of sub-projects or "tasks." From this perspective "project management" is made up of a whole bunch of "task management." It amazes me to see people worried about how they are going to manage a project when...they can't tell you how to manage a single task that makes up the project. To belabor the point, if one can't manage the tasks that make up the project...one can't manage the project!

The best example in software systems is that of the lowly program. If you can figure out how to manage the development of a single program and you're on your way to understanding how to plan and manage a "system." Conversely, if you are having problems with managing the development of a single program... then you probably will have even more problems with the whole system.

So to improve, we need to focus our attention on the "tasks" and their management first. How do we manage a task? First we need to understand that each task is a project. Sounds like circular reasoning, but it isn't really. What it means is that the same techniques and concepts that apply to "projects" also apply to the "tasks" in the project. It also follows, if we are having a problem managing this "task" then we need to figure out what its sub-tasks are and how to manage those. Our goal is to reach a point where we are "successfully" managing a task. When we achieve this goal we are ready to move up a "level" in our management.

So the fourth secret is:

Focus on managing "tasks" before "projects."

Managing for Quality

A little discussed issue is that of how to manage for "quality." Quality, in the sense I mean, deals with the measurement of the excellence of the "deliverable" or "product" of a task or project. Quality is the third leg of the project "triangle." A project triangle describes the balance between cost/resources, time and quality. Changing any one of the parameters most likely affects the others. The understanding and "managing" of this relationship is a significant part of what "project management" is all about.

In spite of the fact that we need to manage all three areas, the emphasis in almost all the project management thought is on the time and cost parameters. In fact, out of all the pages in the texts on project management I've read, less than 2% deal with this issue of the management of quality. In part this emphasis is due to a lack of understanding of the importance of quality. Also, the "management" of cost and time is "easier" than that of quality. Finally, remember that "top management" is emphasizing cost and time in its judgement of project success.

Why do I feel this lack of attention is unwarranted and significant? In software projects it is the quality of the product that greatly affects both long term success and costs. If we place too much emphasis on initial project's time and cost then I will almost guarantee that the quality of the product will suffer.

Why will quality suffer? It is because people doing a project will tend to place their emphasis and effort in the areas where they are being measured. So when "managers" emphasize time and cost their "task workers" will do so also. Add to this the fact that the majority of original system estimates for time and cost are understated. And, as commonly happens, these estimates become "gospel." Finally, we usually don't plan for or "measure" product quality. The net result of this scenario is that people know they can let product quality slide a little in order to meet the time and cost deadlines!

So how do we manage for quality? The keys are 1) defining the quality parameters 2) defining levels of achieving on each parameter 3) assigning "value" to each parameter/level agreed to 4) measuring against this definition 5) analyzing and adjusting our actions to improve overall quality for the next time. For a more complete discussion of this topic see the paper Quality - Let's Discuss this "Can of Worms".

We will have achieved a significant milestone in systems when our project management includes managing for quality.

So the fifth secret is:

Manage a project's quality as well as its time and cost.

Planning and Re-planning

Few would disagree that management of a system project can be helped greatly by the use of a "good" plan. And, in general, the best plan is one that is as accurate as possible. But most plans lose their "accuracy" rapidly. So one could conclude that to keep a plan of value it must be constantly updated. The truth is that most plans are not updated anywhere near as often as they should be. As a consequence, most system project plans are nearly useless!

Why are inaccurate plans of little value? It is because one of the prime uses of plans is as a communication tool. A good plan communicates at least the who, what, when and how much of the tasks of the project. When this data is not accurate on a plan, people will rapidly ignore the plan. They will ignore it for planning their own work. More significantly, they will not provide needed updates to a plan they perceive as worthless.

Additionally and unfortunately, when we don't re-plan we don't re-think. Much of the value derived from planning comes from the thinking we do about the tasks. If this thinking is not being done, then critical issues affecting the project may easily be overlooked. Revising a plan should provide the impetus to step back and see the forest as well as the trees.

Based on my experience in the systems arena, most tasks should be re-planned based on the following rules.

Re-plan when:

1) we are at 10%, 50% and 70% of elapsed time

AND

2) once a day

AND

3) when any fact comes to our attention that we feel may affect, by greater than 2%, the schedule completion date, resources required or quality.

In producing a plan or a "re-plan" we need to be sure that we understand what is in a plan. A "plan" tells who, when, how long, why, how much, what products and of what quality. This may sound pretty obvious. Unfortunately, I believe that the majority of the "plans" done each day leave out at least one of these items. This is unfortunate, because leaving out any this data diminishes the usefulness of the plan. It follows that when we "re-plan", we must also re-specify all these items in a plan.

Another often missed feature of most re-plans is that of changes and comparisons. We should in our re-plan compare it to the original plan. The re-plan should point out our tasks added, tasks removed and tasks completed. It should also highlight variances between the "original" plan and the re-plan as to who, when, how long, how much, and the quality.

So the sixth secret is:

The latest "good" plan is worth many times the former.

Self-Management versus Managing Others

The chances for project success go up directly in relationship to how well each person on the team accepts individual responsibility for its success. This in a sense could be called "self-management." Conversely, if we try to "manage" people into doing what is right, in the right amount of time, by the right date, etc.... we will almost surely not achieve it. My experience and observation has convinced me that trying to control a project by externally monitoring and controlling others is much less successful than commonly believed.

If self-management in a project is desired, then how do we achieve it? The way you get everyone to be responsible and their own self manager has to do with the personnel issue. In line with earlier comments, I believe it is not easy to develop or change someone into a self manager. But it can be done sometimes. Mostly, this is another issue where having the people with the right attitudes on the team is worth a great deal. Give me someone who wants to do something over someone who has to do something!

It may be helpful to focus on what the self manager on a project or task takes as their responsibility. They need to take responsibility for implementing the concepts("secrets") discussed so far. It also means accepting responsibility for achieving the plans as to time, resources and product quality. So responsibility in a project or task includes the following:

Understanding Goals and Success
Correct Staffing/Knowledge/Skills
Right Type, Timing and Quality of Communication
Understanding and Managing for Quality
Planning and Re-planning
Reporting of Problems with Re-plans
Periodic Status Reports
Review of Future Plans
Task/Project Reviews for Time, Resources and Quality
Achieving the Plans and Goals for Time, Resources and product Quality.

If you do get **everyone** on the project team to accept responsibility for the above items then the likelihood of success goes up tremendously. The job of project manager then becomes much more one of coordinator. On the other hand, if we try to "manage" and/or "control" people in these areas the job is much harder and success less certain.

So the seventh secret is:

The best management on a project is self-management.

Tools and Techniques

This section really isn't about any secrets. Rather it's about the value of many of the tools and techniques for the management of projects. Some techniques and tools are useful but some are very overrated. But, if you have successfully dealt with the "secrets" I've discussed then these "other" tools and techniques probably become more important.

Project management software is one of the magic answers that has been pushed in recent times. Unfortunately, most of the software available is much better suited for a project such as "building a house" than for the development of software. Two key characteristics of software projects are its high level of change and differences in project personnel. Most project management software works best on projects with much less change and personnel uncertainty than is typical of software projects. There is a real potential for this type of tool to help us in our software projects. We unfortunately are just not there yet in the state of the art.

Here are some additional techniques/concepts which I feel are very overrated for most system projects:

Critical Path Method
Pert Charting
Networks
Contingency Planning
Risk Analysis
Float Calculations
Milestones
Skills Inventory/matrices
Earned Value Concepts

Time and space does not allow me to make the case for why the above are overrated. Nevertheless, one of the real challenges for those people involved in managing software projects is to not waste time trying to apply techniques and methods of dubious value. Focusing on the right techniques and issues will pay the largest dividends.

So the final "non-secret" is really a philosophical challenge:

Knowing what technique not to apply is often times as important as knowing what to apply.

Conclusions

I've spent considerable time studying, dealing with and thinking about system project management. The breadth of the issues and the challenges are amazing. There are no easy "silver bullets" to slay the beast. Many of the tools and techniques exposed are sorely inappropriate to many environments and projects. What **may** have worked on a multi-billion dollar missile project may or may not work on your mult-thousand dollar software project.

I believe that the "secrets" I've outlined are very significant to the successful management of software projects. Maybe I believe it's as simple as that... clear understanding, clear goals, clear people, clear plans, clear tasks, clear responsibility, clear quality means clear success! If that sounds <u>too</u> simple and easy it probably is. If there is one thing I've learned over the years there is usually one or more new challenges on each project. As the saying goes... **I may not have all the answers but I sure admire the problem!**

REFERENCES

Mattson, Robert R., "Why System Projects Don't Quite Succeed," Proceedings 1985 International Meeting - HP3000 Users Group, Washington, D.C., September 8-13, 1985.

Mattson, Robert R., "Communications - Why do Systems People Have Such a Hard Time With It?", Proceedings 1987 North American Conference of Hewlett-Packard Business Computer Users, Las Vegas, Nevada, September 20-25, 1987.

Mattson, Robert R., "Software Quality - Let's Discuss This "Can of Worms.", Proceedings 1988 North American HP Business Computer Users Conference, Orlando, Florida, August 7-12, 1988.

Software Quality

Let's Discuss This "Can of Worms"

Robert Mattson
9545 Delphi Road S.W.
Olympia, Wa 98502

Introduction

I assume everyone developing software wants it to be of the "highest quality." Recently, I've focused my attention on this goal, what it means and how to move toward achieving it. This has led me to some insights about some key concepts that affect the achievement of the goal "high quality software." Further, I've developed a practical technique for "defining" and "measuring" quality. I wish I had understood these concepts and this technique years ago. It would have improved a lot of software with which I've been associated. I hope the following discussion provides "food for thought" and a technique you can apply. My goal is to contribute something that will be used to improve the "quality" of software.

Why Care About Quality?

It seems that our lives are filled with the "reasons" for producing high quality products. Publications, television and people appear to be constantly communicating the reasons for and exhortation to the production of high quality.

The storyline is the same related to systems/software. There are numerous books and technical articles which tell of the high cost of "poor quality" and the savings from "high quality" systems work. The cost of poor quality in terms of system development cost, development time, user dissatisfaction and software maintenance is today staggering, if one extrapolates from the published numbers.

Yet, what would we find if we could "magically" measure the "quality" of all the software produced this last year with that of ten years ago? Have we made significant gains in the "quality" of software? I imagine that we could find some examples of higher quality software than that of ten years ago. But what about the average piece of software from last year? I'm afraid I'd have to say that, in my opinion, on average it isn't significantly better than ten years ago. The literature tends to support this opinion.

I heard a well traveled "guru" of this business recently state that "the majority of systems people haven't read a systems related book since they got out of school." If this is truly the case, is it any wonder our software isn't of the highest quality or at least getting better.

If this is the case, why is it? Partly, I believe it stems from the "cultural attitudes" of the people developing the software. Systems workers in this regard have been influenced by the same "cultural attitudes" as the "blue collar" worker or any other worker in our country. The average worker in the U.S. today would not rate "producing what they produce at the highest quality" as even close to their most important goal in life or more significantly "at work." They might pay lip service to it but you probably wouldn't find it rated very high if you could tap their true value system.

There are other reasons that software is not of the highest quality. Many times we are not asked to produce high quality. Or we are asked but not given the resources. Or we see that what is rewarded is not high quality, so we "play the game." There are numerous other "environmental" causes.

In spite of the gloom, there is a brighter view. There are people and groups of people who greatly value the goal of excellence and believe in producing high quality products including software. These people want to and, in some cases, are producing top quality software. Others want to increase the quality but aren't sure how. Hopefully, it is to this latter group that the rest of the concepts and techniques outlined will be of the greatest benefit.

Why is "Software Quality" a "Can of Worms"?

This issue of "software quality" is a "can of worms"... because there seems to be so many rigid "beliefs" about what is "True". As I started to explore this issue, I asked a number of people to tell me what makes for "high quality software". The answers were anything but in agreement. Some people said "maintainability", some "efficiency", some "meeting the users expectations". Some even talked in terms of the fact that there are "probably a number of things".

But the general trend was definitely toward one or two strongly felt parameters.

Additionally, this area is one that has a lot of "nerves" attached to it. If you want verification of the sensitivity of the subject, try leading or doing some code/software or design "walkthrus." Such discussions can easily and quickly degenerate into heated arguments if someone happens to touch another persons pet quality belief (nerve).

The literature I found didn't seem to add much enlightenment to the issue of software quality. There seem to be a number of "camps" or "approaches." Probably the most visible approaches are the "better testing" and "structured analysis" ones. As an aside, what is most surprising is how little one can find on the subject that goes beyond vague generalities.

A significant consequence of this disagreement and lack of attention about what constitutes software quality is either lower software quality or at least not much improvement. Why is that? Well first of all, two people with rigidly held opposing views seldom talk or learn from each other. And we know, when people can't or don't discuss an issue they seldom can learn much from each other. It follows that if we aren't learning we aren't improving. Additionally, people pay lip service to "quality" without realizing the fuzziness of their meaning. This lack of a clear "model" and definition many times results in each person "doing their own thing."

So, what can we do to get control of this "can of worms?" I believe the first step is to look at the whole issue in a new way. The second step is to apply the new techniques that flow from this new view.

Software Quality Viewed as Quality Parameters and Values

The key to understanding software quality is to apply a new "model" to it. This "Mattson Model of Quality" starts with an understanding of the multi-dimensionality of the "quality parameters" related to software. Additionally, one needs to understand the concept of "values", or the "weight" placed on a parameter. Then to complete the model we need to understand about the differences between the "judges." Finally, for greatest benefit, we need a technique to apply the "model." Let's take each in turn.

Quality Parameters

What do I mean by "quality parameters" or "QPs" if you will? "QP"s is a description for all the categories or areas of measurement by which one might judge software quality. For example a common QP is "speed". Another common QP is "documentation." Following is a reasonable list of the QPs for a "program".

Program Quality Parameters:

Functional Specifications
Suitability / Job Effectiveness
Speed / Responsiveness
Resource Impact
Robustness / Forgivingness
Adaptability / Flexibility
User Acceptance / Satisfaction
Business Cost Effectiveness
User Independence / Support
User Documentation
Ease of Learning
Ease of Use / User Efficiency
Implementation / Installation
User Interface Uniformity
Development Task Management
Cost to Develop
Time to Develop
Test Plan / Testing
Technical Review / Walkthrus
Defects / "Bugs"
Maintenance Time/Cost
Maintainability
System/Internal Documentation
Adherence to Standards
Integration

The most important thing to notice is that there are <u>many</u> different "quality parameters" by which we can measure software. Conversely, there is <u>not</u> just a single measure. In other words, "bugs/defects per line of code" or "structured code" or "maintainability" are only one of many possible QPs. Notice also that each of these QPs has itself potentially "sub-QP". In other words, "documentation" might be divided into "user" and "system" documentation.

Part of the challenge in this approach is coming up with a list of Qp that is comprehensive without being unmanageable.

How then do we measure each QP? Some QPs such as "speed" may have quantitative measures. But for most QPs there is no clear quantitative measurement. For example what is the quantitative measurement for the "quality" of "system documentation". The strategy that seems to work best in these cases is to discuss the parameter in terms of three levels: unacceptable, ok, excellent. Thus, taking our example of the QP "system documentation" we can usually define what is unacceptable, what would be ok, and what would be excellent. Many times it helps to use examples or references to standards to communicate these levels.

Ah, but you say, what is not acceptable "speed" for one program is excellent for another. That is why there is no such thing as <u>THE</u> quality way, technique or point. Rather, excellence must be defined for each situation. That is why we must <u>discuss</u> and come to agreement as to what the various levels are for each particular piece of software (for example each program). This discussion helps define the different views of quality for each QP. This is very important. We assume too often, I'm afraid, that each party involved in the development of software have the same measure of excellent quality for any particular QP.

How does one start to apply this concept? Refer now to Exhibit #1. This is a "Software Quality Form" that is used for documenting "quality" for a program. This form shows a reasonable set of QPs for writing a program and has room for documenting key points related to the different quality levels. See Exhibit #2 for what this form would look like when we fill out the "Quality Level" fields.

This form is filled out <u>before</u> a program is developed. Let's focus for the moment of the columns labeled "Quality Parameters", "Unacceptable", "Ok" and "Excellent". The person who is responsible for doing a program fills out the "Quality Level" columns on the "standard" program version of the form. In addition, they add any other QPs that might be of special significance. The description of the levels for each QP is discussed with the persons project leader and/or supervisor. Some of it can be discussed with the user. The purpose of the discussion is to get an agreed understanding between the parties as to the definition of levels of quality for each QP for this program.

You may have noticed some other columns on the form. To understand the reason for these and their use we need to address the next dimension of the problem... that of "value" ratings.

The Concept of "Values".

I've describe above how to establish the QPs and specific quality levels by which we can judge the software. Now we need to discuss the concept of how much "value" we place on each.

The concept of "value" has to do with the "weighting" we put on achieving the "excellent" or "ok" level of "quality" on any QP for this program. In this technique a "value" is placed on the achieving of the "ok" and "excellent" level of each QP defined. These are "relative" values. In other words giving one QP level a 10 and another a 20 means that achieving the second(20 pt value) is twice as important as achieving the first (10 pt value).

Why do we need/want to do this? First, because sometimes the achieving of an excellent level in two QPs are counter to each other. For example code size and user flexibility are usually mutually exclusive. In this case the developer needs to know which QP to emphasize. Other times the QPs are not counter to each other but there is simply not enough "time" or "resources/dollars" to achieve excellence in all areas. In this latter situation, we must know where to place our emphasis.

The numeric "values" also give us more information than such statements as "speed is important" or "I want us to emphasize maintainability". The relative values of the numbers provide us with much better information on "how much" we want to emphasize one QP over another.

Let me give you an example. I want to develop a program that makes a "fix" to my database and I know I'll only use it one time. This database has 100 million records. I define the QP "speed" in terms of unacceptable, ok, and excellent. I define the QP "system documentation" in terms of its three levels. Note: I'm only using two QPs for clarity but one would have "defined" many QPs and placed "value" on achieving their different levels of quality. Here's how those "values" might look.

	----- Quality Level -----	
	"OK"	"Excellent"
Speed	10	50
System Documentation	5	10

Now if I had to make a choice between achieving the excellent level in "speed" or "documentation" which would I want? What if chosing one means the other QP will only be achieved at the "OK" level?

I believe software professionals make these kind of trade-offs in their head. At the same time I've found that far too many times the trade-offs made by one person are in disagreement with those another might have made. This model/technique allows for the good communication of the possible trade-offs. This model/technique will help avoid the frustration, conflict, wasted dollars and effort associated with making the "wrong" trade-offs.

Implied in this technique is an associated rule. The rule is that unless agreed to by the "players" in the process (i.e. user, manager, project leader, programmer team associates, etc) it is not acceptable to achieve the "unacceptable" level for any QP. This is true even if this QP has little value. If the "significant others" agree to achieving the "unacceptable" level then it is alright to do this. Note, this new agreement has really just been to define "ok" to mean whatever "unacceptable" had been. A special case is when a QP is valued at zero. This means that no value is placed on this parameter and anything is probably acceptable.

Now, the reason for the "value" columns on the Software Quality Form is clear. This is where the "values" can be "set" for a piece of software. The doer assigns "values" after filling out the description of the quality levels. These are also discussed with the supervisor and/or leader and where appropriate with the user. All parties come to agreement on what the relative values of each achievement for each QP will be. Notice, there is not a value column for the "Unacceptable" level; this is because there is NO value in doing this!

A Word About "Judges"

There are a number of people who will ultimately judge the "quality" of a piece of software. These "judges" include the "builder", the builder's associates, the users, the builder's manager, "outsiders", etc. We may not wish to have it judged but it is a "fact of life."

It may be obvious by now, two different people ("judges") would probably do the following differently if asked to do it "separately".

1) Describe the QPs by which to judge software Quality.
2) Describe what "unacceptable", "ok" and "excellent" levels of achievement are for a particular QP for a particular piece of software.
3) Rate the relative importance (values) of the QPs and their levels for the piece of software.

The important thing to understand is that this happens all the time in the "real world." What confounds us many times is that we unconsciously assume that the result of the three steps above is same for both parties. Then we wonder why our supervisor is less than happy about the "excellent" work we just completed. Or if we are a supervisor we wonder about the "competence" of our employee who completed such "poor quality" work. Further, because we have no formal and written quality specification we have to rely solely on our memories of whatever discussions we might have had about this program. If it is obvious that the parties "disagree" on the "quality" of the software, relying on memories will usually not lead to a productive and positive resolution. Rather, what usually happens is either no discussion takes place or one does but it results in emotional exchanges.

It is very enlightening to take a Software Quality Form for which we've completed the quality level specifications only and give it to all "interested" parties. It will be enlightening to see how the different people will value the different QPs and levels of quality.

Is there a better way to deal with the differences that so commonly arise currently? Yes, the better way is to apply the "model/technique" I've outlined.

Using the Technique --- The Software Quality Form

You've already been introduced to the Software Quality Form. The use of it is fairly straight forward but there are a few additional steps in its use. Here are the steps:

1) Discuss, decide and list the QPs that you want to use in the judging of this type of software.
2) Discuss, decide and document what each level of achievement for a QP would be.
3) Discuss, "agree on" and document the relative "values" to place on each level of achievement for each QP.
4) Add up the Excellent column of "values"
5) Have the builder and one or more of the other "judges" rate each QP for achievement of the goal.
6) Total the rating points for each "judge" and calculate the percent of "Excellent" achieved.
7) Discuss the difference in ratings and the ways to increase the "excellence" (percentage) next time.

It sounds simple, doesn't it? And really it is! It does take a little time. But the time is very small compared to most software development efforts. The second time it is done will be faster than the first, the third time faster than the second, and so on. The QPs will tend to be established and not changed for each new program. What will change is the definition of "Ok" and "Excellent" levels of achievement as well as the value placed on each. But even there, you'll see a lot of re-use of descriptions and similar values for somewhat similar software.

The use of this form will improve the quality of software. Why? First, just using this technique will be helpful in discovering how often "quality" is poorly defined. Everyday, you'll see the "word" quality used as if its meaning was unmistakable. Having a better understanding of the nature of "quality" will allow one to handle this fact. Secondly, people will spend time thinking about what excellence is. Consequently, they will have clearer goals for achieving quality. Therefore, they will be much more likely to develop the features that result in "quality." Conversely, less time will be spent doing the wrong thing or emphasizing the wrong QP.

Conclusions

There is no "absolute", "universal", and "always" way to measure the level of achieved software quality. Neither, however, is quality only in the eye of the beholder. There is a middle ground. The quality of most software can be judged on the basis of some "quality parameters" or QPs. For each QP there are unacceptable, ok and excellent levels of achievement. Each of these levels has a relative importance to the various "judges" of the software. The QPs, levels, and values can be established for any piece of software. The product produced can then be evaluated against these pre-established measures. We can then calculate the "level of excellence" achieved as a percentage of total excellence conceived. This model/technique makes significant improvements over the simplistic and/or poorly defined methods commonly employed.

This conceptual model can be applied to the "real world". The use of the Software Quality Form is an efficient way to do this. The benefits of applying this model are easily worth the time/cost spent. "Quality" will improve because we will have a clearer picture of what it is, what we want and whether we are attaining it!

Software Quality Form

System Name: ______________________ Est. Time: ______________

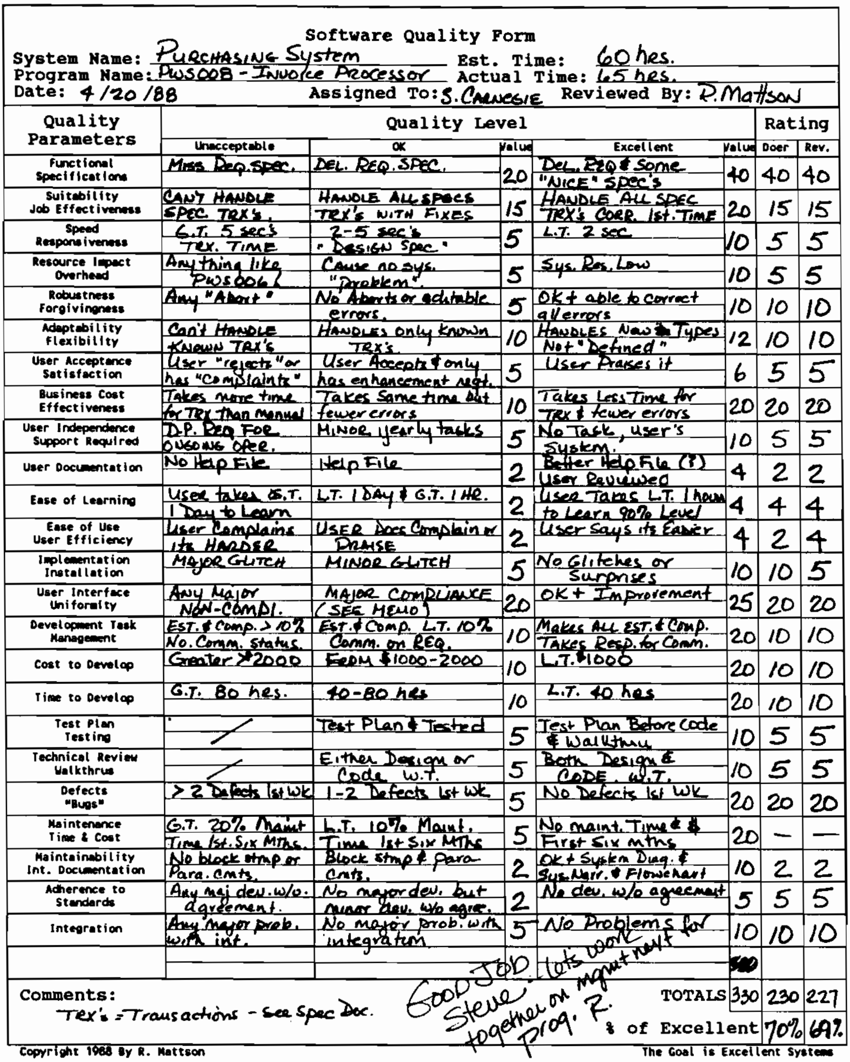
Program Name: ______________________ Actual Time: ______________

Date: / / Assigned To: Reviewed By:

Quality Parameters	Quality Level					Rating	
	Unacceptable	OK	Value	Excellent	Value	Doer	Rev.
Functional Specifications							
Suitability Job Effectiveness							
Speed Responsiveness							
Resource Impact Overhead							
Robustness Forgivingness							
Adaptability Flexibility							
User Acceptance Satisfaction							
Business Cost Effectiveness							
User Independence Support Required							
User Documentation							
Ease of Learning							
Ease of Use User Efficiency							
Implementation Installation							
User Interface Uniformity							
Development Task Management							
Cost to Develop							
Time to Develop							
Test Plan Testing							
Technical Review Walkthrus							
Defects "Bugs"							
Maintenance Time & Cost							
Maintainability Int. Documentation							
Adherence to Standards							
Integration							
Comments:				TOTALS			
				% of Excellent			

The Goal is Excellent Systems

Exhibit #1



Software Quality Form

System Name: Purchasing System Est. Time: 60 hrs.
Program Name: PWS008 - Invoice Processor Actual Time: 65 hrs.
Date: 4/20/88 Assigned To: S. Carnegie Reviewed By: R. Mattson

Quality Parameters	Quality Level					Rating	
	Unacceptable	OK	Value	Excellent	Value	Doer	Rev.
Functional Specifications	Miss Req. Spec.	Del. Req. Spec.	20	Del. Req & Some "Nice" Spec's	40	40	40
Suitability Job Effectiveness	Can't Handle Spec. TRX's.	Handle All Specs TRX's with Fixes	15	Handle All Spec TRX's Corr. 1st. Time	20	15	15
Speed Responsiveness	G.T. 5 sec's TRX. Time	2-5 sec's "Design Spec"	5	L.T. 2 sec	10	5	5
Resource Impact Overhead	Anything like PWS006!	Cause no sys. "Problem".	5	Sys. Res. Low	10	5	5
Robustness Forgivingness	Any "Abort"	No Aborts or editable errors.	5	OK + able to correct all errors	10	10	10
Adaptability Flexibility	Can't Handle Known TRX's	Handles only known TRX's.	10	Handles New Types Not "Defined"	12	10	10
User Acceptance Satisfaction	User "rejects" or has "Complaints"	User Accepts & only has enhancement reqt.	5	User Praises it	6	5	5
Business Cost Effectiveness	Takes more time for TRX than manual	Takes Same time but fewer errors	10	Takes Less Time for TRX & fewer errors	20	20	20
User Independence Support Required	D.P. Req For Ongoing Oper.	Minor yearly tasks	5	No Task, user's System.	10	5	5
User Documentation	No Help File	Help File	2	Better Help File (?) User Reviewed	4	2	2
Ease of Learning	User takes G.T. 1 Day to Learn	L.T. 1 Day & G.T. 1 HR.	2	User Takes L.T. 1 hour to Learn 90% Level	4	4	4
Ease of Use User Efficiency	User Complains its Harder	User Does Complain or Praise	2	User Says its Easier	4	2	4
Implementation Installation	Major Glitch	Minor Glitch	5	No Glitches or Surprises	10	10	5
User Interface Uniformity	Any Major Non-Compl.	Major Compliance (See Memo)	20	OK + Improvement	25	20	20
Development Task Management	Est. & Comp. > 10% No. Comm. Status.	Est. & Comp. L.T. 10% Comm. on Req.	10	Makes All Est. & Comp. Takes Resp. for Comm.	20	10	10
Cost to Develop	Greater $2000	From $1000-2000	10	L.T. $1000	20	10	10
Time to Develop	G.T. 80 hrs.	40-80 hrs	10	L.T. 40 hrs	20	10	10
Test Plan Testing	/	Test Plan & Tested	5	Test Plan Before Code & Walkthru	10	5	5
Technical Review Walkthrus	/	Either Design or Code W.T.	5	Both Design & Code. W.T.	10	5	5
Defects "Bugs"	> 2 Defects 1st Wk	1-2 Defects 1st Wk	5	No Defects 1st Wk	20	20	20
Maintenance Time & Cost	G.T. 20% Maint Time 1st. Six Mths.	L.T. 10% Maint. Time 1st Six Mths	5	No maint. Time & $ First Six mths	20	—	—
Maintainability Int. Documentation	No block stmp or Para. Cmts.	Block stmp & Para Cmts.	2	OK + System Diag. & Sys. Narr. & Flowchart	10	2	2
Adherence to Standards	Any maj dev. w/o. agreement.	No major dev. but minor dev. w/o agree.	2	No dev. w/o agreement	5	5	5
Integration	Any major prob. with int.	No major prob. with integration	5	No Problems for	10	10	10
					[illegible]		
Comments:				TOTALS	330	230	227
				% of Excellent		70%	69%

Comments:
TRX's = Transactions - See Spec Doc.

Good Job Steve. lets work together on mgmt next Prog. R.

Copyright 1988 By R. Mattson

The Goal is Excellent Systems

Exhibit #2

Strategic Planning In Small MIS Shops

Terry W. Simpkins
Spectra-Physics
Retail Systems Division
959 Terry Street
Eugene, OR 97402

Strategic Planning, what is it, why is so much attention being paid to it these days, why should it be done, and assuming that it should be done, how does one go about it? All very good questions that don't have obvious answers but need to be understood by every MIS manager; especially in small shops where resources are extremely limited.

What Is Strategic Planning?

This paper does not pretend to be the definitive explanation of the topic, but rather to highlight the experiences of one small shop that has gone through the exercise. The state of the literature on strategic planning for MIS outlines a massive project covering every conceivable aspect of systems, lasting well over a year, employing several dedicated people, and having a considerable price tag. The results of this process is a document of considerable weight and volume. Every detail of future systems would be explained, and a considerable amount of the initial design work for these future systems might well be included. Detailed forecasts of the expected volume of transactions going through the systems may be outlined along with pages of explanations for the volume changes. While this may be possible (even reasonable) for a large installation (say Ford Motors), the concept is laughable in a small shop of say 5 people. The small HP3000 shop normally consists of 1 to 6 people and is completely buried already, the thought of adding something of this magnitude is simply not within reason. Faced with this situation, we undertook the task of developing a workable strategic plan. However before we could begin the process, we had to develop the methodology that would be used. This is a description of that methodology and some of the reasoning that went into it. For the small shop it is very important to plan, perhaps even more important than for large installations, since we as small entities are more sensitive to change. But we must also consider the very limited resources available to us and keep the amount invested in the planning process in perspective. Because of this our effort will by definition be much smaller and of less detail. The crux of the issue is to develop realistic expectations of the planning process and to resist the desire to develop an all encompassing document that is all things to all people.

Define exactly what you expect the process to produce and focus on the activities that will address those goals, and stick to them. In my opinion, realistic expectations are developing an outline of what the upper management of your company (or division) expects the environment to be for the next several years, and what are the types of systems required to support that environment. Any more detail, and the level of confidence drops quickly.

Why Plan?

If you are considering the development of a Strategic Plan, I assume that you are aware of the reasons planning is important. Perhaps you are here because your boss has told you that you are going to develop a plan. Which ever is true, a quick review of the major reasons that I see for planning is in order.

1) Businesses change over time, as do the systems requirements of the business, while installed software stays perfectly constant without human intervention.

2) Non-trivial computer systems take a considerable length of time to develop and install. This may include definition, design, selection, coding, training, etc. Whether you create or purchase the software, it is not an overnight project.

3) All resources are finite, and we want to make the best possible use of them. If we can avoid spending valuable resources on short term needs in favor of addressing needs that will be with us for a longer period of time, we have probably gotten more value for those resources. Likewise if we address the most important needs of our users first, we develop credibility with management and that has several benefits.

4) By having a better understanding where we are headed we minimize "mid-course corrections". This saves valuable time and effort and gives the appearance that we know what we're doing.

5) We generate enthusiasm and user buy-in when they understand the direction systems are taking. Even if they don't totally agree with the direction, they will be more cooperative if the at least know what is going to happen. There are more reasons why a Strategic plan should be developed, but as MIS professionals, you should already understand them and if you don't there is plenty of published material on the subject.

Purpose of the MIS Plan

1) The MIS plan should provide an overview of the state of the current systems and how they got where they are. This historical perspective may not add value to the future plans, but will provide good background understanding for why we are planning now and some of the problems that can be avoided by planning.

2) The plan should provide insights into the direction of MIS both from a broad general perspective as well as some of the specific needs that are identified. And a general time frame for when these changes should be in place.

3) You need to measure how far you currently are from where you want or expect to be. This provides a basis for priority setting and resource allocation. It also prepares management for the requests you are going to make and gives them a measure of how reasonable and realistic the plan.

4) Finally once the target is established and you understand how far away you are from that target, you can outline what is needed to move toward your goals and how quickly you can expect to get there.

Composition of the MIS Plan

I. Introduction/Narrative Summary

In order to provide a basis for the plan a brief historical recap of the current set of systems is included to help define where we are today. Trends are discussed as well as the reasons for the major decisions that have been made. the purpose is not to justify or condemn, but to provide better understanding of why we are where we are. The next section covers where we want to go in very general terms. This is the visionary portion of the plan, my chance to gaze into the crystal ball and present my vision of what MIS should be and the role it should play in the business. Included in this are the type of resources required to support that vision, people, hardware, software, business commitment, etc. All of the requirements may not exist, but I can describe what I need anyway. A review of the current role of MIS is include to compare and contrast the current environment with my vision of the future. Next a review of the MIS organization. Describing how we are organized, the equipment currently in place and a recap of the departments strengths and weaknesses. Finally a brief comparison with other MIS departments in organizations similar to ours. Size of staff, machine capacity, services performed, and budget as a percent of sales comparisons provide a relative measure of our performance.

II. Existing Systems Profile

For each system, a narrative is created covering in detail the basic function of the system, state of the code, documentation, user understanding, MIS understanding. Also discussed are the things the system does well and the areas that the system doesn't address or that it addresses poorly, including major known bugs or omissions. A general description of the backlog of change requests for the system provides a basis for management to judge the validity of our assessment of the system. We then recap the expected impact of these changes in terms of cost, time, stability and probability of success. Ideally your business has or will develop a Strategic Plan for the entire business, if so you will be able to use that plan as input to the MIS plan and build from it. If not then portions of that general business plan must be developed either explicitly or implicitly in order to proceed further.

III. Comparison of Existing and Future Operating Environments

For each functional area of the business a projection of the future environment is required if we are to understand and project the information and systems requirements. This projection must be that of the functional manager, it is not required that he/she actually create it, but he/she must agree with it and be willing to sign his name to it. The purpose of this projection is to compare the current environment with that of the future and to understand the impact of the changes on the information needs of the business and the ability of the current systems to meet those needs. Each functional area is divided into the processes that compose it. For example Marketing might be broken into the following subcategories: Promotion & Advertising, Sales, Market Research, and Market Planning. For each of these areas, a profile is created listing the important attributes, a description of the current environment and the expected future environment (see form #1). The information on this form is strictly business oriented, it should be completed by the users and not consider systems at all. If a Strategic Plan already exists, or is being developed in conjunction with the MIS plan, this should be a part of that plan. If not, you must develop it at this point in the process, since the rest of the plan builds upon it. The next step in the process identifies the important "information systems" used in the operation of the business. These systems are not necessarily computer systems, but rather the way information of collected arranged or used. Systems could be defined as processes or functions (ie. Master Scheduling, Purchasing, etc), or along the lines of the current computer software packages if that is felt appropriate. The important thing here is not the way "information systems" are defined, but the listing and explanation of the critical factors that impact these systems and the informational nature of these critical factors. The intent here is to highlight the type of information that is needed to support the future business environment. Form #2 is one method of ferreting out this information. Each critical factor is then described according to how is is impacted by the seven information characteristics listed across the top of the form. Any factor listed must, by definition, relate to at least one of these characteristics. It may relate to more than one, but seldom to all of them. These descriptions are then the basis for evaluating current systems and any alternative systems or designs (see forms #3 & #4). Because these factors are "critical" and they impacted by certain information characteristics, they are the obvious basis for analysis and comparison of alternative systems. Of course there will be other criteria in the selection process such as cost, complexity, support available, interconnectivity with other systems, etc.; but this list of critical factors and the informational requirements will be a vital part of the requirements definition. In our case a proposed replacement system had already been identified and we focused our attention on that alternative. The process being that if that alternative didn't sufficiently meet the users needs, then we would start from scratch to evaluate other systems. It should be noted here that the outcome of completing form #3 could be that the current systems satisfactorily meet the needs of the the enterprise. In fact this conclusion would be expected where systems had been recently replaced. This would not mean that the exercise was wasted effort, but rather that those involved in the planning process had verified that business needs were being met. Businesses periodically examine the market segments they participate in to insure that they are in the correct ones, and should likewise examine there systems to insure that they are appropriate and providing the required information. It is critical that user complete these forms. The MIS staff should assist to insure that a proper level of detail is included and that the descriptions are measurable and quantifiable, however in order for there to be a meaningful result the users must "own" the information that comes from the exercise. In my experience, the generation of the forms used in the project is very important. The format of the information gathering must assist in the extraction of the information required to construct the plan or you will find it very difficult to communicate your goals to all involved parties. The forms should lead the users through the process and force them to provide the required focus on business issues and information requirements. Another tool to help and guide the users is to create a "straw man" list of business attributes, information systems and critical factors for each of the areas. This list should not be cast in stone, but should provide "food of thought" for the users, seeds to start their thought process. As the future information needs of the various areas are assembled, the cost of meeting the needs and the impact of not meeting these needs must be understood, along with the relative importance of the needs in order for management to make correct resource allocation decisions.

IV. Plans to Meet Future Needs

Once the future needs of the business have been identified, quantified, and ranked, a plan to meet those needs must be developed. This may include the selection process for new software, a list of modifications to current software, and an installation/project plan. Project management skills and tools play a very important role in this section of the plan. It is not critical that the actual details of the process be included in the plan, rather that the process that will be used is defined and understood so that management can buy off on it. This section will also include the sequence of projects to be undertaken. This will reduce the "mid-course corrections" caused by misunderstood priorities and increase the confidence level in the plan since management has reviewed and "blessed" it. This is the place to address the hardware required to make the software plan possible. Where possible, link hardware needs with specific projects, to reflect the true cost of various alternatives and projects. Keep in mind that hardware requirements are usually a step function and seldom follow a smooth curve. Form #5 is one method of presenting the action items required to "fill the Gaps" between the current state of systems and the state required to meet the future needs of the organization.

Human Resources Planning

As a part of your plan be sure to address the people portion of your department. This is your greatest asset and deserves attention just like your software and hardware. Succession planning, career development and professional training are all part of a complete Human Resources Planning effort (see form #6). Personnel planning is important for several reasons. Much time and money has been invested in your programming and operations staff to get them to their current level of understanding of your systems, by failing to provide an effective career plan for these employees you risk losing them and having to reinvest in training their replacement. This retraining effort also has opportunity costs associated with it in addition to the cost of the training; that being the value of the projects that cannot be undertaken while this training is taking place. These costs can very often be avoided with good career planning. Remember too that happy employees are more willing to work the long odd hours often required in our profession and are always more productive than unhappy ones. By developing your staff and expanding the knowledge base of each employee you also reduce your exposure in the event that an employee does leave. By having several people who can perform each job or support each system, you have designed in back up for everyone and you improve the quality of systems because interactions and interfaces are better understood by everyone.

In order to assist in your understanding the forms and give you a starting point, I have enclosed excerpts from our finished plan. These examples show what we considered to be an appropriate level of detail and some of the major points we are concerned with. I have also included a copy of our planning outline and organizational structure used in the development of the overall Strategic Plan.

As stated at the beginning, this paper is not designed to be an exhaustive study in the art of Strategic Planning, but rather to reflect some of the insights gained by performing it in a small shop environment. All portions will not be applicable to every installation, but the general requirements and approaches I feel are common through all companies. Feel free to use the forms and modify them to best serve your needs.

Form #1

BUSINESS ENVIRONMENT

Today vs. Future

Functional Area

Department

Business Attribute	Today	Future (5 Years)

Form #2

Change in Information Needs
Due To
Changes in Business Environment

Functional Area □

Department □

Information System (Not Computer)	Critical Factor That Drives or Impacts the Information System	**Timeliness** How Quickly Updated & Available	**Detail Level**	Information **Commonality** Same Data Across User ,Groups & Which Groups	**Arrangement** Indexes, Sorted Order	Time **Horizon** How Much History is Required	**Accessibility** Who Can Access Data	**Accuracy** How Critical is Absolute Accuracy

Form #3

GAP ANALYSIS

Information Needs vs. Current System

Functional Area □

Department □

Information/Functional Requirement	Ability of Current System to Provide

Form #4

GAP ANALYSIS

Information Needs vs Proposed System

Functional Area

Department

Information/Functional Requirement	Ability of ASK System to Provide

Form #5

GAP ANALYSIS --- ________ FUNCTION

FUNCTIONAL STRATEGIES	STATE REQUIRED	CURRENT GAPS	ACTION REQUIRED TO FILL GAPS

FORM #6

Are assumptions upon which the strategic plans are based realistic regarding human resource requirements?

Over time, what skills will become obsolete, change in nature, or be eliminated?

For what functional skills/positions are we likely to encounter a shortage of qualified candidates in the marketplace, now or in the future?

Do the present managers within the function have adequate technical/managerial skills to meet the strategic changes occurring at RSD?

What are the principal H/R obstacles to achieving the function's strategic objectives?

Are age patterns in the organization imbalanced, suggesting high future attrition or career path blockage?

Is there adequate or excessive turnover in any group, at any particular level?

Is there a proper balance (staff mix) of managerial, professional, technical, and support staff within the function?

What are the most significant skill deficiencies within the function organization? How will such gaps be addressed?

FORM #6 (continued)

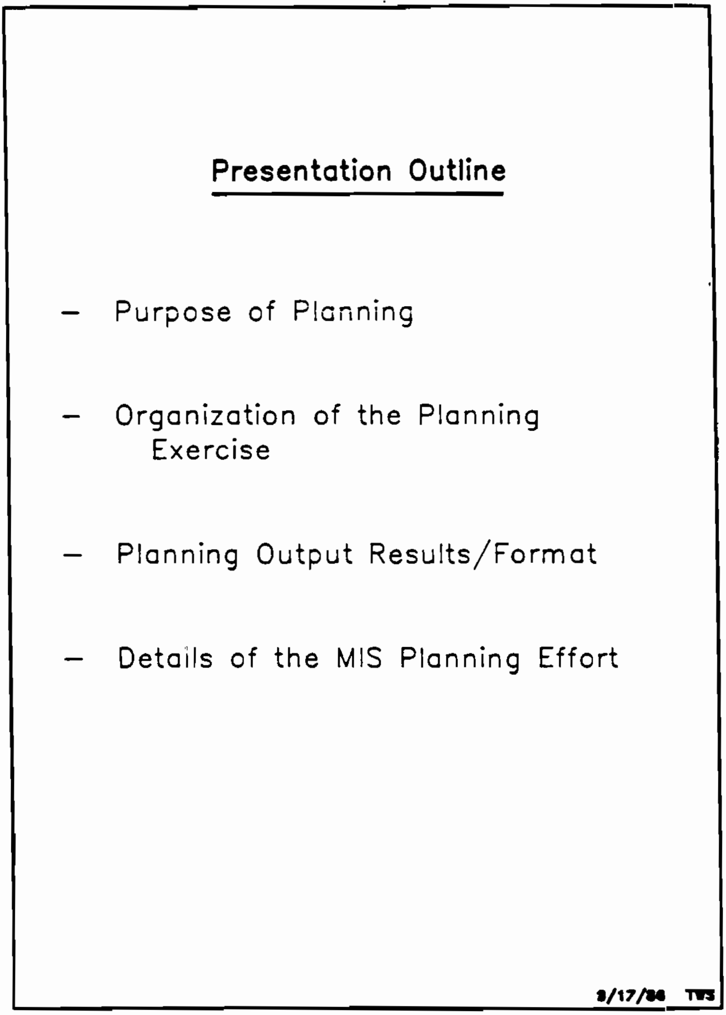
Are the organization and structure and staffing of the function appropriate for the achievement of strategic objectives?

To what extent will qualifications for existing positions change in light of strategic plans? How will such changes be addressed?

Do the strategic plans/objectives call for projects or processes that have no precedent at RSD? What are the implications for staffing requirements? Design of the present function organization?

Which positions, if not filled, will have the most detrimental effect on achieving the function's objectives?

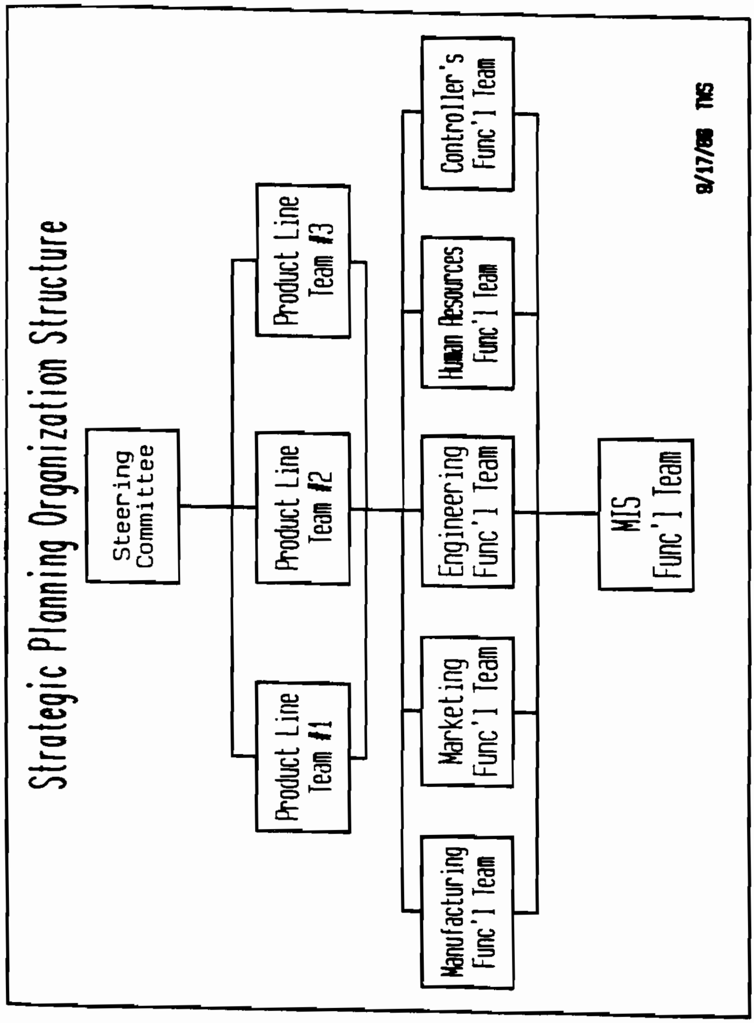
What impact would a product line de-emphasis or discontinuance have on the responsibilities of the function staff?



Presentation Outline

- Purpose of Planning
- Organization of the Planning Exercise
- Planning Output Results/Format
- Details of the MIS Planning Effort

8/17/88 TWS



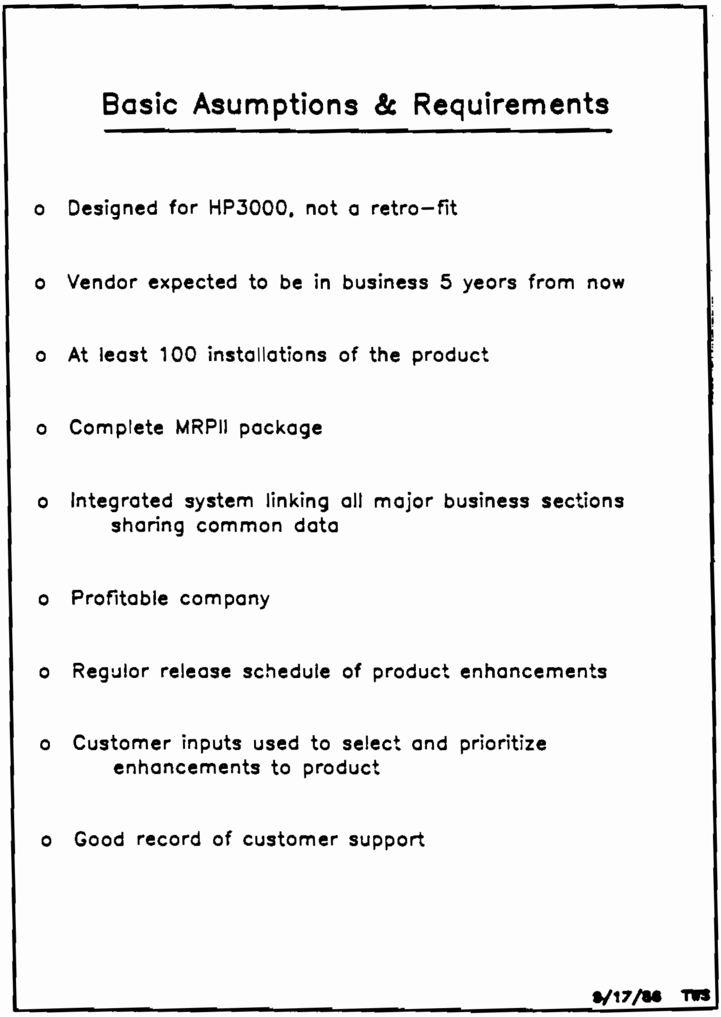
Strategic Planning Organization Structure

9/17/88 TWS

Details of MIS Planning Effort

- o Basic Assumptions/Requirements
- o Information Needs Derived from Functional Plans
- o GAPS Identified & Analyzed
- o GAP Analysis Form Generated
- o Specific System Reviewed for Fit
- o Comparision of Current System to Candidate System
- o Specific Recommendations Made
- o Rough Time-Line Created
- o Implementation Structure

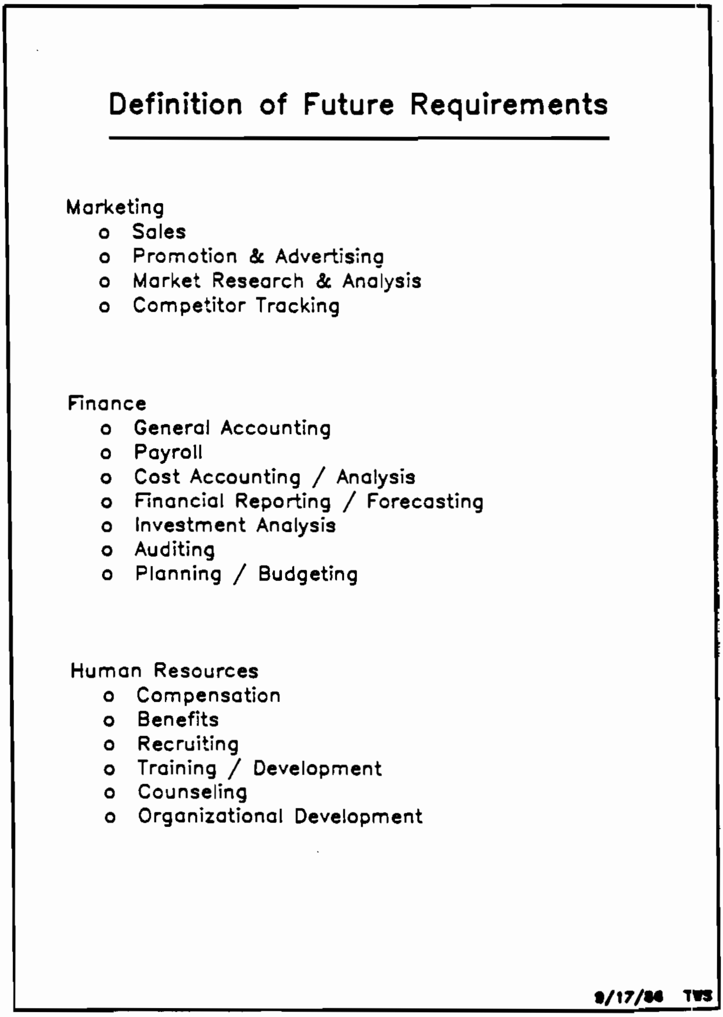
9/17/96 TWS



Basic Asumptions & Requirements

- Designed for HP3000, not a retro—fit
- Vendor expected to be in business 5 yeors from now
- At leost 100 instollotions of the product
- Complete MRPII pockoge
- Integroted system linking oll mojor business sections sharing common doto
- Profitoble compony
- Regulor releose schedule of product enhoncements
- Customer inputs used to select ond prioritize enhoncements to product
- Good record of customer support

9/17/86 TWS



Definition of Future Requirements

Marketing

- Sales
- Promotion & Advertising
- Market Research & Analysis
- Competitor Tracking

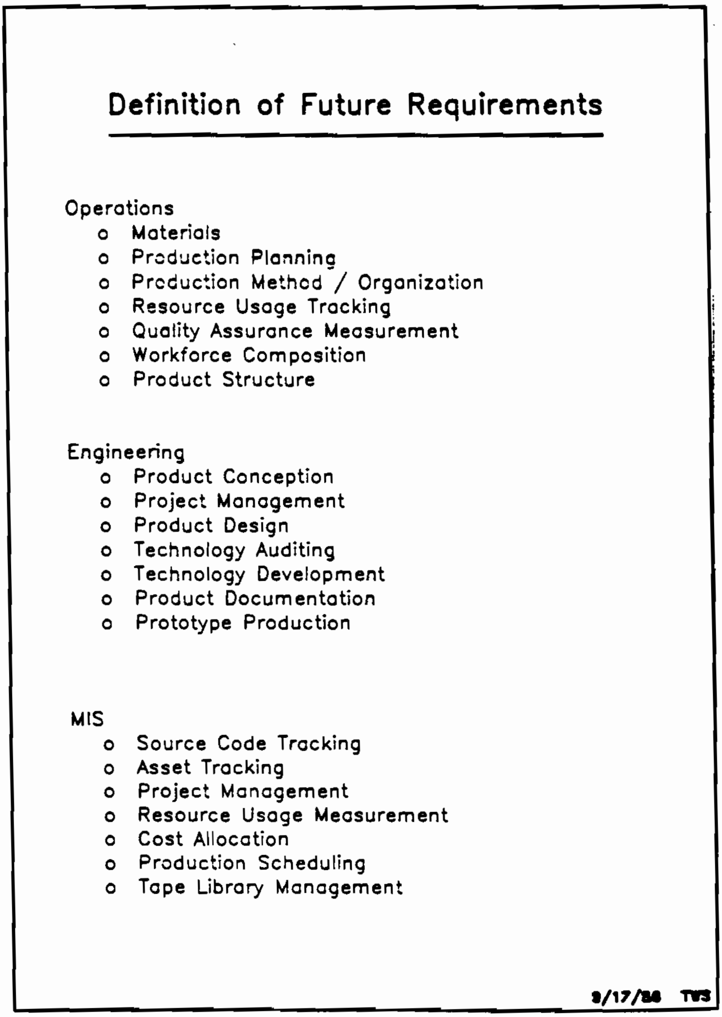
Finance

- General Accounting
- Payroll
- Cost Accounting / Analysis
- Financial Reporting / Forecasting
- Investment Analysis
- Auditing
- Planning / Budgeting

Human Resources

- Compensation
- Benefits
- Recruiting
- Training / Development
- Counseling
- Organizational Development

9/17/86 TWS



Definition of Future Requirements

Operations
- o Materials
- o Production Planning
- o Production Method / Organization
- o Resource Usage Tracking
- o Quality Assurance Measurement
- o Workforce Composition
- o Product Structure

Engineering
- o Product Conception
- o Project Management
- o Product Design
- o Technology Auditing
- o Technology Development
- o Product Documentation
- o Prototype Production

MIS
- o Source Code Tracking
- o Asset Tracking
- o Project Management
- o Resource Usage Measurement
- o Cost Allocation
- o Production Scheduling
- o Tape Library Management

9/17/88 TWS

Change in Information Needs
Due to
Changes in Business Environment

Functional Area: CONTROLLER'S

Department: COST-ACCOUNTING

Information System (Not Computers)	Critical Factor That Drives or Impacts the Information System	Timeliness How Quickly Updated & Available	Detail Level	Information Commonality Same Data Across User Groups & Which Groups	Arrangement Indexes, Sorted Order	Time Horizon How Much History is Required	Accessibility Who Can Access Data	Accuracy How Critical is Absolute Accuracy
Measurement of manufacturing cost	Use of repetitive manufacturing work flow	Next day analysis of cost input and production output	By defined work centers or process steps for labor and indirect costs	Labor input common with payroll system	By assembly within work center	12 months rolling	Accounting, production supervisors and managers, material control	I/O must balance to payroll system, inventory control systems
		Monthly summaries one day following last shipping day	By part number/assembly type for materials	Materials input and production output with material control				
			Costs collected in process "buckets" rather than J.O.s					
	Number of assemblies		Ability to group common assemblies for averaging of cost data		By model families			

GAP ANALYSIS

Information Needs vs. Current System

Functional Area	CONTROLLER'S
Department	COST ACCOUNTING

Information/Functional Requirement	Ability of Current System to Provide
I. Measurement of Manufacturing Costs	
A. Use of repetitive manufacturing work flow	
1. Next day analysis of cost input and production output	o Transactions for labor, A/P and inventory movement collected weekly o JOCS is a weekly batch process system
2. Monthly summaries one day following last shipping day	o Due to labor processing at corporate, we receive the labor files 2 working days after last shipping day. JOCS weekly and monthly processing done on 3rd working day with reports available the morning of the 4th working day
3. Data defined by work centers or process steps for labor and indirect costs	o OP codes available but not currently used in JOCS
4. Monthly summaries by part-number/assembly type for materials	o Current MacPac/JOCS interface reports by JO. Modification of existing program may allow reporting as desired
5. Collect costs in process "buckets" rather than JOS	o Costs must be collected through JOs on JOCS

GAP ANALYSIS

Information Needs vs. Current System

Functional Area	CONTROLLER'S
Department	COST ACCOUNTING

Information/Functional Requirement	Ability of Current System to Provide
6. Labor input common with payroll system	o Labor is currently collected via our timecard entry/payroll system
7. Material input and production common with material control	o Multiple system interfaces are used to collect inventory movement transactions for processing within JOCS
8. Data arrangement by assembly within work center	o Primary data arrangement is by JO w/OP code as secondary for labor
9. Historical data - 12 mos. rolling	o JOCS provides a cummulative history report through the year. At year end history files are archived to tape and purged off the system. With modifications could provide 12 months rolling. (JOCS year, June - May)
10. Data accessible to accounting, production supervisor and managers, and material control	o Multiple copies of reports are currently distributed
11. Material and labor transactions must balance to the payroll and inventory control systems	o Interface systems, currently in place, do not provide 100% accuracy
B. Number of Assemblies	
1. Ability to group common assembly for averaging of cost data	o Blocks of JO numbers are assigned for each product line. Within the block assignment to assembly/model numbers is random

Change In Information Needs
Due to
Changes In Business Environment

Functional Area: MIS

Department:

Information System (Not Computers)	Critical Factor That Drives or Impacts the Information System	Timeliness How Quickly Updated & Available	Detail Level	Information Commonality Same Data Across User Groups & Which Groups	Arrangement Indexes, Sorted Order	Time Horizon How Much History Is Required	Accessibility Who Can Access Data	Accuracy How Critical Is Absolute Accuracy
o Project cost tracking	Labor cost reporting	Weekly batch on demand reports	By resource	Payroll, HR	Project, resource	Forever back	MIS and project leader	±5%
	Number of projects	Periodic on demand	All aspects of projects		Project, category, user group	Forever back 1 yr. future	MIS only	Not critical
o Project planning	Number of projects, size/complexity of projects	On demand, in real-time	All aspects of project and tasks and resources		Resource, task, dependencies	Length of project	MIS and project leader	
o Hardware resource usage	Number of jobs Number of sessions Hardware capacity	Auto update Auto update Auto update	At job/session level to track trends		By: Job user/ acct. resource	2 yrs. min.	MIS only	±5%, more if known and explained
o Asset tracking system	Number of assets	Monthly updates		Gen. acct./FA, receiving	Asset number Asset type		MIS/facilities/ gen. acct.-FA	Absolute
	Frequency of asset moves	Daily input, weekly updates		GA/FA - facilities			MIS/facilities	Absolute

us/TWS/INFONEED/Page 7

GAP ANALYSIS

Information Needs vs. Current System

Functional Area	MIS
Department	

Information/Functional Requirement	Ability of Current System to Provide
Project cost tracking	o Timely reports not available o JOCS system cumbersome and not well suited for detailed project reporting
Project planning	o Available on PC only
Hardware resource usage	o Current software meets 75% of needs o Not as flexible as needed o All requirements can be met with current software if enhanced
Asset tracking system	o Current system does not provide timely or reliable information o All maintenance tracking done manually o All PCs manually tracked due to no "category" or "group" feature in current asset system o Current system tracks at a high level so components have no visibility o Transfer of assets is cumbersome and time consuming o No "assigned-to" feature for tracking loanable assets
Production scheduling	o All done completely manually
Tape tracking	o All done completely manually
Source code tracking	o All done completely manually
Office supply charge-back	o Issues manually logged and totalled o Manual JVs created to transfer costs

ws/D2/TWS/gapanal.18

FUNCTIONAL GAP ANALYSIS -- MIS FUNCTION

Functional Strategies	State Required	Current Gaps	Action Required to Fill Gaps
o Focus MIS efforts on unique LSD business issues	o Adaptable vendor supported software that provides basic transaction processing and is modifiable to meet unique LSD needs. MIS free to address LSD unique needs and opportunities.	o Patchwork software used only by Spectra-Physics. o Unsound, unclear transactional software. o MIS resources dedicated to maintaining unsound software. o Software difficult and risky to modify. o Some software impossible to modify.	o Install software. o Provide additional business training to MIS staff through involvement in professional groups such as APICS, NAA, etc. o Develop software in-house to augment system for specific division needs.
o Provide integrated systems that match business methods and are: - Flexible - Shared information - Timely - Accurate - Appropriate	o Usable information available as needed.	o Not timely. o Requires manipulation. o Not complete.	o Department managers determine specific information requirements. o Install and other systems as required.
	o Software that cleanly links all parts of the business.	o Redundant/conflicting data. o Messy/unreliable interfaces. o Multiple entry of same data.	o Install and integrate additional systems as required.
	o Software that supports: -State-of-the-art mfg. methods (JIT, etc.). -Option ready products. -Market development/analysis. -Sales order and service tracking. -Repetitive costing. -Real-time decision making. -Project management. -Product restructuring. -Asset management.	o JO based only. o No configurations flexibility/efficiency. o Non-existent. o Inadequate/manual. o Non-existent/manual. o Fragmented/inadequate. o Fragmented/inadequate. o Manual/awkward. o Inadequate/manual.	o Install software modules: - -Accounts Payable. -Quality. - with options entry. -Examine when released. o Reconfigure BOMs to modular o Evaluate IFPS and FCS, choose one. o Evaluate o Link marketing PC database to

FUNCTIONAL GAP ANALYSIS -- MIS FUNCTION

Functional Strategies	State Required	Current Gaps	Action Required to Fill Gaps
			o Evaluate use of o Develop and install fixed asset tracking system. o Evaluate inventory tracking/analysis and develop additions as required.
o Eliminate manual data exchange between LSD and other SP entities	o Automatic exchange of data between LSD and other S-P entities. Direct data access capability. File transfer capability.	o Mail and Fax only. o Hard copy must be produced. o Telephone used extensively. o Limited file transfer.	o HP mail installed. o Users trained on HP mail and file transfer function. o installed to replace SOPS/BARS. o MIS trained on HP mail support and trouble-shooting.
o Selective use of office automation to increase efficiency	o Adequate access to personal computers based upon usage. Centralized in-house expertise in both hardware and software. o Networked computing resources to facilitate data transfer and exchange.	o Fragmented knowledge, no central point. o Problems and needs handled reactively, not proactively o Inadequate access in some areas. o Mainly stand-alone PCs requiring redundant data entry.	o Identify specific resource to be focus point. o Develop in-house resource to keep current on trends and technology. o Provide more general use computers. o Develop comprehensive networking strategy.
o Automate computer operations to reduce headcount growth rate	o Flexible automatic program scheduling software.	o None installed.	o Examine available software and select best fit. o Train Operations staff and install software.
o Be able to meet computing needs if normal computing resources disrupted	o Risks identified and evaluated. o Recovery options identified and evaluated. o Plan of action ready.	o All risks not known. o Options unexplored. o No plan in place.	o Develop comprehensive disaster recovery plan.

Comparison of Current Systems to Future Systems

Rated on a scale of 0 - 5 where:
where : 0 = doesn't provide
5 = perfect match to needs

Attribute	Current Systems	Future Systems
Real Time	1	3
Data Integration	1	3
Product Option Support	0	4
Repetitive Mfg	0	3
JIT Methods	1	3
BOM Manipulation	1	2
Product Line Differentiation	2	3
Planning/Forecasting	0	3
Purchasing Module	1	3
Quality Module	0	1
MRP	3	3
Barcode Support	0	1
"What-if" Modeling	0	1
Project Management	1	1
Asset Management		
- Accounts Recievable	2	4
- Inventory	2	3
- Fixed Assets	0	0
Accuracy/Confidence	3	4
Adaptability/Flexibility	1	3
Support Required	1	3
	-----	-----
total	20	51
possible : 105		
percentage	19%	49%

Systems match-up

Current	Future
MACPAC	
JOCS	
SOPS/BARS	
FGI	

Misc Systems

GAPs to be filled by: additional systems ie. Fixed Asset
Project Mgmt
(?)
(?)

File down-load to PC software

Security Tips and Techniques for Beginners

Terry W. Simpkins
Spectra-Physics
Retail Systems Division
959 Terry Street
Eugene, OR 97402

This paper is not the definitive answer for all security issues or questions. It is designed to be a relatively high-level primer for people who are either new to the 3000 environment, who have just taken over responsibilities for security, who have suddenly become interested and aware of security, or who are looking for some general tips to help them get started in the right direction.

I will not go into dramatic detail on any one of the areas, but rather give you several areas and parameters of security to look at, consider and evaluate. You decide which ones are the biggest risk for you and would have the most value for you to pursue further.

The general areas that I will talk about are: passwords; hardware access security; message system that MPE provides; specific programs to be aware of that represent a potential security risk; and monitoring activity and looking for trends--closing the barn door after the cows have escaped, so to speak.

First, let's talk about passwords. Everybody knows passwords are good, kind of like motherhood and apple pie. Auditors believe that everybody should have a password and I probably agree with them. At least one level of password security is appropriate for almost everything. Although there may be a couple of exceptions, they're few and far between. Should you use the MPE security system and the password mechanisms in it, or one of the aftermarket products? That's up to you. Your decision will be driven by your user's preference, your preference, your budget, and what kind of system you want to design and provide for your users. Any answer can be the correct one, if your logic is sound.

There are a couple things that are important: one, passwords should be non-trivial, that is, they should not be obvious, they should not be one letter, and they should not be the defaults from your vendor; two, they should be at least three letters long and relatively easy to remember. Passwords can be non-trivial, non-obvious, and still easy to remember. The worst scenario is to create a great password, just to have people write it down and tape it to their terminal. You have to strike a balance between security and workable security. Passwords should not be carved in stone, that is, they should be changed on a fairly frequent basis. I have found every three to six months is a reasonable timeframe. You want people to remember their password so don't change it every week because they'll have to write it down to remember it, and you don't want them to write it down. At the same time, you don't want passwords to become widely known. You should, if possible, utilize a scheme where passwords are only valid for certain people at certain devices. For example, "ASK," the software package that we primarily use, has a mechanism for restricting what commands are executable by which password, and this is not your MPE password, but rather your ASK password which you have to supply to get into the software. The ASK password allows you to use specific passwords to restrict users to specific databases. If you have a test database, a production database, and a development database, a given password may only allow access to the test database, or it may restrict access to the development and the test databases but not the production, or allow access to all three, or any combination. The password can also restrict you to what commands you can run and let you restrict what MPE user may log into the ASK system under this password. For example, if my ASK password is Terry and my MPE logon is Simpkins.Manman, I can setup the system so that to use the ASK password, Terry, one has to be logged onto the MPE user Simpkins.Manman. Now, "Terry" (my first name) is a lousy password to have in ASK, but it is a good mechanism to use so that you can control more closely what users are doing in your system. Don't misunderstand me, I'm not doing a sales pitch for ASK--I'm only one of their customers. I'm merely using ASK features as examples of what you can do in an applications security scheme. There are other methods, there are other applications security schemes, I'm not encouraging one over the other.

The next security area we need to talk about is hardware access. Hardware access can be gained three ways: direct connect, i.e., to an ATP or ADCC; through a phone line via a modem; or a DS line from another system. Hardwired--pretty straight forward. If people have access to your building and if they have access to a terminal, they're in. At that point, you're faced with controlling physical access to your building and physical access to the terminal. Are your terminals in a locked room, or do you encourage terminal use and set them on peoples' desks? You probably do the latter--you set them on peoples' desks. If you do that, you have to rely upon an informal security network which is someone walking in on an unauthorized user and saying, "hey Bob, what are you doing on this terminal, you're not supposed to be here." Or, you rely on your password and applications software security mechanisms to prevent unauthorized access.

Telephone lines are a different issue. Dial back modems, modems with passwords, etc., are all options you may choose. What you use will depend upon how many phone lines you have, how frequent the access is, and how varied the audience or the user base is. If only your programmers use the modem, it's probably less of an issue, or actually, maybe it's more of an issue, because you don't want those call-back mechanisms or passwords to have to be used. One scheme that I've seen used successfully is to write a very simple SPL program, that is a logon, no break, UDC, that every user goes through when logging on. All this program does is ask, "am I running on LDev21? Yes or no." If no, I'm done. If yes, "Is the person logging on authorized to use LDev21?" If yes, fine, let them proceed, if no, bounce them. You can define what is an acceptable logon as narrowly or broadly as is appropriate for your organization. This is a pretty simple program to write. All you need is the "WHO" intrinsic, and the "WHO" intrinsic is relatively simple to call from SPL, or COBOL.

With dial-up-lines, I'm a firm believer in changing your telephone number to the modem on a frequent basis. The exchange that the phone number comes in on (the first three numbers in the telephone number) should be different than your company's voice telephone lines (your published phone number) just to make it a little bit more difficult for people to discover. It's not that difficult for your average grade school or teenage hacker to write the program described in War Games, one that dials all the telephone numbers in all the local exchanges and finds all the ones with carriers. If you wrote such a program, you'd probably be surprised how many carriers you would find--there are a lot of computers out there with a large number of dial-in lines. I recommend that you change the phone number on a fairly frequent basis and always change it when someone that knows the number leaves your company. When you change it, don't just increment the number by one or decrement it by two, take it to a different exchange. The price associated with changing the phone number is not very great, where we are it's about $50; therefore, we do it every three to six months. We notify the programming staff the day the number changes--we don't give them a lot of notice. If they happen to be absent that day, they don't find out about it until they return.

If you are more concerned about restricting access to your phone lines, you have a couple of other options. One, you can "down" the phone lines. If you have 24-hour a day, 7-day a week coverage by your operations staff, this is probably a viable alternative. For example, say a programmer wants to logon, he calls in and says, "hey Bob, up the modem, I'm going to call in and do some work." Then the operator can "down" the modem when he sees the programmer log off. If you don't have that much coverage, or maybe you don't have any coverage at all by the operations staff, (you're it) this is not the easiest solution. There are lots of ways you can get around it, they just take more effort. I would venture to say that at least 90% of us have at least one modem on our machine, and that's the one HP sent us. Even though you don't use the support link for anything other than sending your HPTREND Reports to HP, or the occasional Response Center call, someone out there is trying to find your system, and how to get into it.

DS access is a little bit tougher to control because you probably have a couple of machines DSed together, and you have batch jobs logging on back and forth all the time at random occasions to transfer files, or trip flags, or look up data for validation. In this case, "downing" the device isn't necessarily a very good option for you. The best alternative I can come up with if you're extremely concerned about access is a variation on the modem security program. Have another table that you check for valid logons that says, "oh, yea, he's coming in across the DS lines and he's logged on to that corporate account, that's ok, we'll let him go," otherwise bounce him.

If you have a person that you have reason to suspect as being a security risk at another division or location of your company, or on another computer, a variation on the modem security program might be a viable, and fairly intelligent solution for you. Remember this about DS access, on certain versions of MPE, capabilities of the session on the source machine travel with the user to his session on the target machine. I don't remember which version(s) of MPE had this "feature" but it used to exist. One of your alternatives is obviously to down your DS lines and require people on the remote machine to ask you to up the DS line. The problem is if you have batch-oriented processes this procedure can get a little bit cumbersome and hard to manage. You will need to either have the operators at the other machine call up, or you'll have to define a very specific time window for batch processes access your machine.

The point I want to make with all of these scenarios, Passwords, Modem Access, DS Access, and Local Hardwired Access, is not that any one particular security method is better or worse than another. They all have pluses and minuses and they're all appropriate in some instances and inappropriate in others. The message here is that to make points with your auditor, you first have to think through the issues as they relate to your installation. You need to have defined your objectives--what it is you're trying to accomplish--to spell out the alternative you're going to use and then have clear, concise reasons regarding why you're using one particular approach versus another to meet the needs as they're defined for your installation. If you have that, what you'll find is that the auditors will give you a lot of points. They may or may not agree with your approach, but they will at least understand what you're doing and why you're doing it. They will be able to score you in a rational, thoughtful manner, as opposed to reacting like, "you haven't thought about it, therefore, it's bad."

Best laid plans of mice of men often fail and, for some reason, somebody that you don't necessarily want to access your machine has, in fact, gained access. How can you minimize the damage? First, don't make it easy for them. Don't paint them a road-map. This is where the message file can be a real friend of yours, or, as it's delivered from HP, actually be somewhat of an enemy. Take a good look at some of those messages. The messages are very user-friendly and try to help people understand exactly what they are doing correctly and incorrectly. As System Manager in charge of security this is a problem for you because message files can be a road map to hacking. I would refer you to an article in the September 1987 issue of Interact on this very topic. Here again, let's refer to our War Games example. You have a kid that doesn't know a HP 3000 from a bag of M & Ms, but he's got a carrier and now he's going to try and get in.

The point is that with trial and error, it doesn't take very long with the way the messages are structured, and he's going to have the exact format and the context that he needs to logon. Now, it's a matter of hitting a valid combination of user and account name. Then he simply takes his little Basic program that he used to find your dial-up phone number, modifies it to try every combination of alpha numeric, eight character long words, and records the message that he gets back. Pretty soon, he's going to get you. Then he's going to be down to trying passwords, and guess what, now he can take the exact same program and try it with all 36th-to-the-8th power combinations of letters and numbers and he's going to come up with a valid password--he's going to get in. I've never really tried this, but I would guess that within the course of a couple of nights, he could break almost every machine. If you're diligent, and you happen to look at your console logs, you might catch this and you'll say, "oh, jeez, somebody's out there hacking away at my modem," and you'll down your modem. But, what if you happen to have the scenario where you can't "down the modem." Now, you have problems. Change the phone number! Quickly! Think about Police involvement and traces!

Ok, you need to keep an eye on what's happening on your machine. Like I mentioned, you get somebody hacking away at night and, unless you happen to read the console log every morning (which I'm sure is not on the latest best seller list) you're not going to see a lot of those messages. What we have done and I recommend that you all do, is to write several little programs to monitor what's going on in our system. Recap the information in a format that is easy and quick to read so you can make some sense out of it.

First thing we've done is write a program that looks at the log files. You tell it which log files you want to look at and it will scan those log files and report out facts or information I want to see on a regular basis.

First thing you do is modify your Cold Load configuration to log everything. It doesn't take up that much disc space and you're not going to keep these log files in your system for very long anyway. All you need to do is use this information once to solve a problem and you're going to pay for a lot of $15 tapes used to store these log files.

What I do is log all console messages. This lets me go back, read the log files, and report security violations that have appeared on my console. I can use log files, then, to look for trends. Do I see a lot of them coming for one particular LDev? Do I see people trying to hack a given user? Does there seem to be a lot of them coming late at night when there's nobody here but the security guard? Things like that. Do a lot of them come in across my modem? Ah, maybe somebody has learned my phone number, I need to go change my modem phone number--that's the first thing.

Secondly, I can keep track of who's purging files. By logging file closes, you can record all file purges. This might be real interesting to help you discover if the purging of a file was an accident. It might also show you that somebody was trying to cover their tracks. If you see somebody purging one of your log files, then you might want to go look at that log file very closely and find out why somebody would want that particular log file to disappear. What kind of incriminating evidence is in that log file?

Lastly, certainly not least, is that I look at all logons on certain LDev's. Specifically, the ones that I look at very closely are, anybody logging in on my dial-up modem, and anybody logging in across the DS line. Are those the people I expect, or is there something funny going on?

All these programs I've just talked about are on the swap tape here in Orlando. These programs were originally written by Harold Jensen. Harold used to be a tech support programmer with Spectra Physics. He is now an SE at HP. I do not know if Harold has ever contributed these programs before; they were a part of a system that he used to track and report resource usage and trends. We have cloned some of his software, combined a couple of programs, taken out some functionalities to fit our specific needs.

The next thing we have done is to make use of a part of MPEX to read the system directory. We used MPEX because it uses Privmode to go right into the directory, we did not want to write that kind of code and then have to maintain it. We had the MPEX product already and it was a relatively simple thing to create some job streams to do what we're looking for. We use MPEX to look at all programs on the system that are prepped with PM, tell us what those are, tell us if those have lock words on them, tell us when they were last accessed, and tell us if they're released. The way we do that is through a series of "listf" steps with our own defined listf mode which MPEX allows.

The last thing we did was to use the indirect reference to a disk file of the LISTDIR5 command. We list off all accounts, groups, and users to a disk file. Then we have a straight forward little COBOL program that reads the file created as output from LISTDIR5, and reports any user, any account, or any group that has specific capabilities, and whether or not it has a password. By definition, all of our users should at least have one password. This method lets us track our compliance to our standards. We want to make sure that we have certain accounts and groups passworded because of the relatively sensitive nature of information in those groups and accounts. At this point, I'd also like to remind people, just because you have a password on a group, a user, an account, doesn't mean it's secure. A lot of the third party vendors (HP falls into this category) have default passwords that go with their software. That password is the same on every system in the whole wide world with that piece of software. Why would that make your system secure just because there's a password on it? Everybody in the world knows what the password is. So, in the article published in the September 1987 Interact, I've listed some of the more commonly used 3rd party software and the default passwords that come from the vendor. You should look at these and make sure you are using different passwords.

All these problems go away if you change your passwords on a regular basis as discussed earlier. Take a look at the list, if nothing else, pick up a phone and call up your vendor. Tell them who you are, ask them what their default password is, and make sure yours isn't the same. A perfect example of this would be your HP support accounts, like TeleSup and Support. These accounts are standard on every HP machine in the world. These accounts contain PM code and their passwords should be changed to one that is unique to your site. Also access to the accounts should be restricted to account users.

The last thing I want to mention is some of the PM (privilege mode) programs that you need to be aware of that could be on your system and represent a hazard to you. I mentioned how we used MPEX, to look for released PM programs that are not locked or protected.

Let me bring your attention to a few programs you should keep your eyes on.

The first one, named "God" is from VeSoft. "God" is a neat program because it'll let you do about anything you want to as far as giving yourself capabilities during your session. Fortunately, VeSoft lockwords that program when they ship it to you. First thing I do is "Restore @.@.VSoft," and as soon as that's done, purge "God." I would recommend the same thing, you have the tape, you can always restore it. Get it off the system.

The next one I would lock up is called MakeSmop. It comes from Kelly if any of you have their RAM disc. I don't have personal experience with this one, but I've heard that it will let you give yourself SM or OP capabilities no matter where or who you are so that's one you want to be aware of.

In addition, there are many such programs in the Telesup and Support Accounts. I'm not going to list all of them here, but I would recommend you sit down, talk to your SE about what all of these programs are and what do they do. These programs should be stored on a tape. Put the tape in your desk drawer; you can always restore the program if needed or restrict access to the Support and Telesup accounts to account users and keep them well passworded.

As mentioned earlier, this is not an all-encompassing security tutorial. Rather, use this as a starting point in evaluating your system's security needs. Security is much more than a password.

MAKING SHORT SHRIFT OF SORTS

Charles Sullivan
RunningMate Software
3001 I Street
Sacramento, California 95816

INTRODUCTION

This discussion examines the variables that affect sort performance. Benchmarks will be presented which compare both the hardware and the software of the various HP3000 systems. An attempt is made to compare HP3000 sort performance to the Digital Equipment VAX 11/780.

ABOUT THE BENCHMARKS

Most of the benchmarks shown here were run several times to ensure that the results presented are reasonable. Unless otherwise noted, the Series 48 and Series 70 had MPE disc caching turned on, the Series 48 had three megabytes of main memory, the Series 70 had eight, and the Series 950 had thirty-two. Each system had two or three disc drives which were a combination of 7933H and 7937H discs. The Series 950 was running the 1.0 release of MPE XL and the other two were using MPE V, either UB-Delta-1 or V-Delta-1.

MPE DISC CACHING

MPE disc caching, as implemented by MPE V software, has an interesting effect on sort performance. When there is plenty of stack space for the sort intrinsics, disc caching actually degrades performance. But when stack space becomes relatively scarce, disc caching speeds up sort performance, sometimes very dramatically.

For these tests, whose results are shown on the next page, the parameters for MPE disc caching, when turned on, were: sequential fetch quantum = 96 sectors; random fetch quantum = 16 sectors; block on write = no. The computer was a Series 70 and 100,000 records were processed.

TABLE 1: EFFECT OF DISC CACHING ON SORT PERFORMANCE

CPU TIME DATA

Words available for Sort/V workspace	CPU seconds MPE disc caching OFF	 ON
24,400	128	163
20,000	134	181
16,000	143	204
12,000	159	234
8,000	190	309
4,000	312	568

WALL TIME DATA

Words available for Sort/V workspace	Elapsed seconds MPE disc caching OFF	 ON
24,400	346	366
20,000	392	417
16,000	444	490
12,000	584	498
8,000	838	770
4,000	1886	1053

PROCESSOR COMPARISON

As you would expect, the more powerful the CPU, the quicker the sort. At least this is true between the MPE V computers. But the Series 950 is quite a contrast to the Series 70. A sort running with no competing jobs will usually finish sooner on the Series 950, but will consume more CPU resources. One supposes that the "fault" for this lies in the software of the 950, rather than in the hardware.

TABLE 2: EFFECT OF SYSTEM PROCESSOR ON PERFORMANCE

CPU TIME DATA

Number of 128-byte records	CPU Minutes		
	Series 48	Series 70	Series 950
10,000	0.74	0.22	
20,000	1.61	0.49	
30,000	2.52	0.76	
40,000	3.51	1.07	
50,000	4.48	1.40	2.02
60,000	5.45	1.72	
70,000	6.54	2.03	
80,000	7.49	2.35	
90,000	8.54	2.67	
100,000	9.56	3.01	4.20
200,000		7.00	9.21
300,000		10.54	14.15
400,000		14.60	19.83
500,000		18.05	25.08

WALL TIME DATA

Number of 128-byte records	Total Elapsed Minutes		
	Series 48	Series 70	Series 950
10,000	1.20	0.58	
20,000	2.61	1.24	
30,000	4.10	1.95	
40,000	5.66	3.00	
50,000	7.35	3.85	2.18
60,000	9.09	4.69	
70,000	10.70	5.64	
80,000	12.42	6.46	
90,000	13.94	7.05	
100,000	15.65	7.73	5.28
200,000		17.20	18.04
300,000		29.58	27.13
400,000		39.95	36.51
500,000		50.56	45.97

ALGORITHM COMPARISON

Hewlett-Packard's standard sort package uses Floyd's Treesort algorithm while SortMatePlus uses Singleton's Quickersort variant. Another, equally important, consideration is that HP's sort confines itself to the user's stack while SortMate uses extra data segments.

For these tests, 100,000 128-byte records were sorted with MPE disc caching turned on. The sort was initialized by calling SORTINIT, the records were sent to the sort with the SORTINPUT procedure, and records were retrieved by the SORTOUTPUT procedure.

TABLE 3: EFFECT OF ALGORITHM ON SORT PERFORMANCE

CPU TIME DATA		CPU Seconds	
Processor	Work Space	HP Sort	SortMatePlus
Series 48	16K words	638	380
Series 48	25K words	523	380
Series 70	16K words	204	114
Series 70	24K words	163	114

WALL TIME DATA		Total Elapsed Seconds	
Processor	Work Space	HP Sort	SortMatePlus
Series 48	16K words	1034	482
Series 48	25K words	800	482
Series 70	16K words	490	189
Series 70	24K words	366	189

THE VAX, THE HP3000, AND BACK-END PROCESSORS

For better or worse, Digital Equipment's VAX 11/780 has become an industry-standard reference point. For years, it was stated that the VAX 11/780 was rated at about one million instructions per second (1 MIP). [I use the MIP only because it is a widely-used way to compare different computers.] Now most observers believe that the 11/780 executes at about 0.5 MIP in a commercial processing environment. The upshot of all this is that the HP3000 Series 68 and 70, which were always considered merely the equal of the VAX 11/780, can now be seen as clearly superior machines.

The VAX benchmarks are taken from the February 1, 1986 issue of Computer Design, *Database Accelerator System Relieves Sorting Bottlenecks,* by Walter A. Foley. Mr. Foley is president of Accel Technologies (San Diego). Accel makes the DBA 1000, a specialized sorting machine which can be used to off-load a host processor. In the following benchmarks, the DBA 1000 was attached to the VAX 11/780 via Ethernet. According to Mr. Foley, the Ethernet connection was not a bottleneck in the test, rather it was the speed of the VAX file system which prevented even better results for the DBA 1000 benchmark.

TABLE 4: VAX 11/780 VS. HP3000 SORT PERFORMANCE

HOST CPU TIME DATA	Host CPU seconds		
	Number of 20-byte records		
Sort Environment	50,000	250,000	500,000
VAX 11/780	100	500	1150
VAX 11/780 and DBA 1000	20	30	50
HP3000/70 [Sort/V]	48	270	609
HP3000/70 [SortMate]	29	161	323
HP3000/950 [Sort/XL]	27	153	312

WALL TIME DATA	Total Elapsed seconds		
	Number of 20-byte records		
Sort Environment	50,000	250,000	500,000
VAX 11/780	550	2600	5700
VAX 11/780 and DBA 1000	120	225	600
HP3000/70 [Sort/V]	69	429	900
HP3000/70 [SortMate]	35	205	413
HP3000/950 [Sort/XL]	27	177	480

SORTING PECULIARITIES AND TIPS

Sort/V grabs all its necessary resources when you call SORTINIT. If your system is out of disc space, you will know immediately. This is better than having to wait for two hours before finding out that you need to free up some more disc sectors. However, this has an unwanted side-effect which is caused by the way the file system allocates disc file extents. When you allocate all the extents for a file when it is created, all the extents must reside on a single disc drive. Therefore, allocating all extents at once will increase the probability of having your job flushed because you are "out of disc space." To eliminate this problem with the Hewlett-Packard sort, simply issue the following file equation:

```
        :FILE SORTSCR;DEV=,32,1
```

The COBOL compiler, probably for simplicity and reliability, opens files for buffered access. This means that the sort intrinsics will probably find enough room on the stack for their work area, but it also means that a COBOL file-to-file sort will almost always run slower than necessary. (SortMatePlus, which replaces the sort intrinsics, will attempt to re-open buffered files for MR-nobuff access. This can lead to a measurable speed increase.) If you can, you should remove sorts from within COBOL programs and use a sort utility program such as SORT.PUB.SYS or SortMate.

Sort/V allows you to alter the collating sequence. If you need to sort upper- and lower-case letters properly, using an alternate collating sequence is necessary. Here is how you do it.

```
        :RUN SORT.PUB.SYS
        >DATA IS ASCII SEQUENCE IS ASCII
        >ALTSEQ MERGE "A-Z" WITH "a-z"
```

Pretty simple, no? However, using an alternate collating sequence does slow down the sort process, but that's another story.

ANOTHER STORY

When developing SortMatePlus, I needed to allow for alternate collating sequences. There is a powerful machine instruction which is tailor-made for just such a purpose: the "compare translated strings" [CMPT] instruction (which is probably used by Sort/V). After about 4 hours of puzzlement and growing frustration, I concluded that CMPT does not work in split-stack mode. Knowing that I would encounter difficulties trying to convince Hewlett-Packard to modify the microcode on thirty thousand installed computers, I began writing a software routine that emulates the CMPT instruction.

On the next page you will find a program which makes use of the final software routine which emulates the CMPT machine instruction. Notice how cumbersome (and incomprehensible) it appears.

```
$CONTROL USLINIT,NOLIST
BEGIN
  INTEGER INDEX,
          KEY'POSITION:=0,  << starting position of key >>
          KEY'LENGTH:=10;   << byte length of key >>
  BYTE ARRAY BYTERECORD1(0:9):="CHARLIE001";
  BYTE ARRAY BYTERECORD2(0:9):="CHARLIE002";
  BYTE ARRAY TRANSLATIONTABLE(0:255);

  << Initialize the translation table >>

  FOR INDEX:=0 UNTIL 255 DO TRANSLATIONTABLE(INDEX):=INDEX;

  ASSEMBLE (LDX KEY'FIRSTPOSITION;
            LRA BYTERECORD1,I,X;
            LRA BYTERECORD2,I,X);
  TOS := KEY'LENGTH;
  GOTO ENTRYPOINT;
LOOP:
  ASSEMBLE (DABZ EQUAL;  LDXI 1;  LRA S-2,I,X;  STOR S-3;
            LRA S-1,I,X; STOR S-2);
ENTRYPOINT:
  ASSEMBLE (CMPB 0);
  IF = THEN GOTO EQUAL;
  ASSEMBLE (LDB S-2,I;  STAX,NOP;  LDB TRANSLATIONTABLE,I,X;
            LDB S-2,I;  STAX,NOP;  LDB TRANSLATIONTABLE,I,X;
            CMP,NOP);
  IF = THEN GOTO LOOP
  ELSE IF < THEN BEGIN  << record1 < record2 >>  END
  ELSE IF > THEN BEGIN  << record1 > record2 >>  END
  ELSE
    BEGIN
EQUAL:  << record1 = record2 >>
    END;
  ASSEMBLE (SUBS 3);  << must delete words left on stack >>
END.
```

Now an example of a program which uses the CMPT machine instruction. Notice how clear and simple the code appears in contrast to the emulation code above.

```
$CONTROL USLINIT,NOLIST
BEGIN
  INTEGER INDEX,
          KEY'POSITION:=0,  << starting position of key >>
          KEY'LENGTH:=10;   << byte length of key >>
  BYTE ARRAY BYTERECORD1(0:9):="CHARLIE001";
  BYTE ARRAY BYTERECORD2(0:9):="CHARLIE002";
  BYTE ARRAY TRANSLATIONTABLE(0:255);

  << Initialize the translation table >>

  FOR INDEX:=0 UNTIL 255 DO TRANSLATIONTABLE(INDEX):=INDEX;

  TOS := @TRANSLATIONTABLE;
  TOS := @BYTERECORD1(KEY'FIRSTPOSITION);
  TOS := KEY'LENGTH;
  TOS := @BYTERECORD2(KEY'FIRSTPOSITION);
  TOS := KEY'LENGTH;
  ASSEMBLE (CON %20477, %7);  << creates the CMPT code >>
  IF < THEN BEGIN  << record2 < record1 >>  END
  ELSE IF > THEN BEGIN  << record2 > record1 >>  END
  ELSE BEGIN  << record2 = record1 >>  END;
END.
```

The surprising fact is this: the machine-level CMPT instruction is faster than the software routine in only one case--when the first characters are not equivalent. The software routine gains its efficienty because it only goes to the translation table when necessary; the CMPT code goes to the translation table for every comparison, even when it is obviously unnecessary. For example, there is no need to go to the table if string1 is "CHARLIE" and string2 is also "CHARLIE". Similarly, you do not need to go to the translation table until the 10th byte if the first nine bytes are exactly the same.

Still, you might be wondering which way is better in reality. If, during a sort, 75% of the comparisons need to examine only one byte, then the CMPT instruction will probably be faster. So here is some "real" data to examine. I extracted all the keys from a master dataset where each key was 12 bytes long. I sorted on the first 10 bytes so some of the keys were "duplicates." These 23,553 records required 375,453 comparisons before they were sorted. Here is how the comparisons were broken down:

Comparisons decided by the 1st byte	114,383
Comparisons decided by the 2nd byte	86,971
Comparisons decided by the 3rd byte	65,911
Comparisons decided by the 4th byte	28,593
Comparisons decided by the 5th byte	50,666
Comparisons decided by the 6th byte	10,152
Comparisons decided by the 7th byte	2,197
Comparisons decided by the 8th byte	492
Comparisons decided by the 9th byte	153
Comparisons decided by the 10th byte	2,984
Comparisons in which keys were equal	13,005

About 70% of the comparisons needed to examine more than one byte, so in this case, using the CMPT instruction would be slower than using the software routine. A large proportion (probably 90%) of sorts encountered in practice will run faster with the software routine.

DEVELOPING A FASTER IMAGE

Charles Sullivan
RunningMate Software
3001 I Street
Sacramento, California 95816

INTRODUCTION

This paper examines some performance concerns most of us have about TurboIMAGE. I have tried to examine topics which have not been empirically studied, such as the degradation caused by logging and the correlation between capacities and DBPUT performance for master datasets.

ABOUT THE BENCHMARKS

Any benchmarks shown here were run several times to ensure that the results presented are reasonable. Unless otherwise noted, the Series 48 and 70 had MPE disc caching turned on. The Series 48 had 3 megabytes of main memory, the Series 70 had eight, and the Series 950 had thirty-two. Each system had two or three disc drives which were a combination of 7933H and 7937H drives. The Series 48 and 70 were using either UB-Delta-1 or V-Delta-1 of MPE V and the Series 950 was running the 1.0 release of MPE XL.

TRANSACTION LOGGING PERFORMANCE

Transaction logging has not been treated with a great deal of rigor when discussing performance. For many years the received wisdom was that transaction logging exacted a severe performance penalty. Now the party line has become that the overhead associated with logging every DBUPDATE, DBPUT, and DBDELETE is negligible.

The database used to obtain the following results was configured as follows. The master dataset had a 10-byte key with a capacity of 9999, an entry size of 110 words and a blocking factor of four. The detail dataset had an entry size of 110 words, a blocking factor of four, and one search path. In all tests, 5000 records were added, updated, or deleted. Auto defer was off.

TABLE 1: DEGRADATION CAUSED BY TURBOIMAGE LOGGING

SERIES 70

MPE disc caching parameters Sequential fetch=96 Random fetch=32 Block on write=YES	Degradation	
	CPU Time	Wall Time
Master dataset DBPUT	18%	11%
Master dataset DBDELETE	20%	15%
Master dataset DBUPDATE	21%	19%
Detail dataset DBPUT	12%	5%

SERIES 48

MPE disc caching parameters Sequential fetch=96 Random fetch=16 Block on write=NO	Degradation	
	CPU Time	Wall Time
Master dataset DBPUT	18%	16%
Master dataset DBUPDATE	20%	15%

SERIES 950

32 megabytes main memory	Degradation	
	CPU Time	Wall Time
Master dataset DBPUT	29-57%	36-59%
Master dataset DBDELETE	19-45%	34-48%
Master dataset DBUPDATE	28-50%	31-60%

The results on the Series 70 and Series 48 were easy to reproduce. But tests run on the Series 950 had a wide range of values. Logging degrades performance much more on the Series 950 than on machines running MPE V.

Before you begin TurboIMAGE logging, the performance penalty of ten to twenty percent for most DBPUTs, DBDELETEs, and DBUPDATEs is worth pondering.

SELECTING A MASTER DATASET CAPACITY

B. David Cathell presented at the Interex conference in Los Angeles in 1984 a pioneering paper about master dataset capacities called *IMAGE: An Empirical Study*. He concluded that capacities for master datasets based on a prime number were not better (or worse) than non-prime capacities. He examined the synonym distribution over different capacities of the same data.

In the course of solving a severe response-time problem that we experienced, I had occasion to write a program which has allowed me to confirm Cathell's result and to extend it. The program simulates taking the current entries in a master dataset and loading them into a dataset of different capacity. The program determines the number of synonyms that will exist and a "clustering" index. This clustering index is the number of 50-record chunks which contain 50 entries. For example, a dataset which is empty except for entries in the first 100 records would have a cluster index value of fifty-one. The higher this value, the longer a DBPUT will take to execute, on average. [See *Identifying Opportunities for Performance Improvement* by George B. Scott in the 1986 Detroit Interex proceedings for data about clustering.]

On the next page you will find a sample of the values that were obtained for a dataset with a 20-character (X20) key and 29967 entries.

Cathell concluded that the only capacity to absolutely avoid was one which is a power of two. Look at the result for a capacity of 65,536. Here the cluster index becomes the "Custer" index--whoever chooses that capacity is going to get massacred by unhappy data-entry employees.

Given a particular set of data and a particular dataset capacity, you cannot predict its DBPUT performance without examining the distribution of ALL the data in the dataset. Here are some guidelines I now use:

1. Half-empty datasets usually produce an excellent cluster index. Datasets which are 60% full are acceptable, but 70% produces DBPUT performance which is erratic. Eighty percent and above is to be avoided. Selecting a prime number guarantees nothing.

2. Be very careful selecting capacities which are very close to being a power of two. From the following table, you can see that 65536 and 65537 are execrably bad, but 65540 would be tolerable.

TABLE 2: SYNONYM COUNT AND CLUSTER INDEX VALUES FOR SELECTED MASTER DATASET CAPACITIES

KEY TYPE = X20 DATASET ENTRIES = 29,967

New Capacity	Percent Full	Synonyms	Cluster Index
40,007	74.9	8,977	883
40,008	74.9	9,847	1,129
40,009 PRIME	74.9	8,927	1,314
40,010	74.9	9,160	1,183
.			
45,006	66.6	8,356	348
45,007 PRIME	66.6	8,268	225
45,008	66.6	9,608	239
.			
49,998	59.9	7,825	54
49,999 PRIME	59.9	7,697	15
50,000	59.9	8,737	17
50,001	59.9	7,593	44
50,002	59.9	7,704	0
.			
55,000	54.5	7,939	15
55,001 PRIME	54.5	7,130	8
55,002	54.5	7,281	0
.			
59,998	49.9	6,803	0
59,999 PRIME	49.9	6,613	0
60,000	49.9	8,262	0
.			
64,000	46.8	10,872	162
64,001	46.8	6,631	3
64,002	46.8	6,804	0
.			
65,535	45.7	10,092	1,772
65,536	45.7	16,165	14,165
65,537 PRIME	45.7	11,366	11,751
65,538	45.7	8,479	986
65,539 PRIME	45.7	7,372	245
65,540	45.7	7,501	161
.			
69,999	42.8	5,789	0
70,000	42.8	6,897	0
70,001 PRIME	42.8	5,802	0

SELECTING A GOOD VALUE FOR BUFFSPECS

TurboIMAGE on MPE V machines allocates an entire extra data segment for data buffering. I tried to determine if altering the number of buffers affected performance. My working assumption was that, in a multi-user environment, setting BUFFSPECS=16(1/120) would produce poor throughput compared to setting BUFFSPECS=64(1/120).

My results showed no difference between such divergent values for the BUFFSPECS parameter. I do not, however, consider this result conclusive. More testing needs to be done.

SINGLE THREADING

TurboIMAGE allows some multi-threading of database intrinsics. The Hewlett-Packard reference manual states that a "two level resource priority locking scheme is used within the DBB to allow single-buffer operations to access the control block concurrently. This involves DBGET, DBFIND and DBUPDATE processes. DBPUT and DBDELETE operations are unable to access the DBB concurrently. These multi-buffer operations must hold a global lock on the DBB throughout the operation." Although the wording is not exactly clear, it does appear that serial read DBGETs also lock the DBB until intrinsic completion.

Single-threading should only be a concern when most of your programs access a single database and your CPU is spending a good portion of its time paused for disc I/O. Although each CPU essentially holds all data in a single database, we have found that our CPUs are not often paused for disc I/O despite single-threaded DBPUTs. The reasons for this are three-fold: 1) MPE disc caching tends to make a system CPU-bound rather than disc-bound; 2) Handling terminal I/O for sixty users over an X.25 network consumes any free time the CPU might have; 3) All our serial reads bypass the DBB entirely (see the next paragraph), thus we have essentially made TurboIMAGE multi-threaded.

If you have performance problems because of single-threading, I recommend that you invest in one of the several software products available which executes fast serial reads and use it with your reporting programs and job streams. These MR-nobuff products do not lock the DBB and hence you should experience more database concurrency and throughput.

SPEEDING UP SERIAL AND CHAINED READS

Several software products are available which can speed serial access to datasets by a factor of three to ten. An early discussion regarding such MR-nobuff techniques can be found in *Overview of Optimizing (On-Line and Batch)* by Robert M. Green from the Interex 1982 proceedings in San Antonio.

Speeding up non-serial access to TurboIMAGE databases is less of a sure-thing. In developing I/O-Mate, it was found that the best overall performance for a calculated (mode=7) DBGET was achieved when just one block of data was fetched at a time. On the other hand, it was found useful to vary the fetch size for chained (modes=5,6) and random (mode=4) DBGETs. I/O-Mate dynamically adjusts fetch size based on the efficiency of recent fetches. I/O-Mate also tries to do things only once. For example, it keeps the two most recent lists for each dataset handy. I/O-Mate also caches the two most recently accessed datasets for even faster access. Finally, although it stores data in extra data segments, it never performs an EXCHANGEDB procedure call. All these optimizations allow I/O-Mate to outperform a "vanilla" DBGET(modes=4,5,6) by fifteen to fifty percent.

APPENDIX: HASHING ALGORITHM

```
DOUBLE PROCEDURE HASH(KEY,HASHEDKEY,KEYLENGTH,CAPACITY);
    VALUE HASHEDKEY,KEYLENGTH,CAPACITY;
    ARRAY KEY; LOGICAL HASHEDKEY; INTEGER KEYLENGTH; DOUBLE CAPACITY;

  BEGIN

    PUSH(STATUS); ASSEMBLE(TRBC 2); SET(STATUS);

    TOS := CAPACITY;
    IF HASHEDKEY THEN
      BEGIN
        ASSEMBLE (LDD KEY, I;  LOAD KEYLENGTH;  DUP, STAX;
                  BRE HASH1;  DLSR 16;
        HASH1:    DLSL 1;  LOAD KEY;  ADXA, LDXA;  INCA, NOP;
                  LSR 1;  DXCH, NOP;
        HASH2:    DECM S-2;  BLE HASH3;  DECM S-3;  DECM S-3;
                  LDD S-3, I;  DDUP, NOP;  LDI 31;  LDIV, DELB;
                  STAX, DXCH;  DCSL 1,X;  DADD, NOP;  BR HASH2;
        HASH3:    DXCH, DDEL;  DLSR 1);
      END
    ELSE
      BEGIN
        ASSEMBLE (LOAD KEY;  LOAD KEYLENGTH;  LADD, DECA;  DECA;
                  LDD S-0,I);
        IF KEYLENGTH = 1 THEN ASSEMBLE (ZROB)
        ELSE ASSEMBLE (DLSL 1; DLSR 1);
        TOS := TOS - 1D;
        IF < THEN
          BEGIN
            ASSEMBLE (DDEL,DEL);
            GOTO ALL'DONE;
          END;
        ASSEMBLE (CAB,DEL);
      END;

    ASSEMBLE (LOAD S-3; STAX);
    IF <> THEN
      ASSEMBLE (LDXA;  SCAN 0;  XAX, NEG;  STAX, DDUP;  DLSR 16,X;
                DELB, LDIV;  DEL, DUP;  LDD S-5;  CAB, LMPY;
                DXCH, MPY;  ZERO, DADD;  DSUB, DZRO;  INCA, DADD;
                LDD S-3;  DSUB;
      CHECKCC:  BE EQ;  BG MORE;  LDD S-3;  DADD;  BR CHECKCC;
      MORE:     DDUP;  LDD S-5;  DSUB;  BL OK;  DXCH, DDEL;  BR MORE;
      OK:       DDEL;  DXCH, DDEL)
    ELSE ASSEMBLE (DXCH,DELB;
                   LDIV, ZROB);

    ASSEMBLE (DZRO, INCA;
    EQ:       DADD);

ALL'DONE:
    HASH := TOS;

  END;  << DOUBLE PROCEDURE HASH >>
```

DISASTER RECOVERY

CAN YOUR BUSINESS REALLY RECOVER?

Presented by:

THOMAS J. DOOLEY, JR.

Consultant to Management
9422 Braeburn Glen
Houston, Texas

(713) 774-8846

1. INTRODUCTION

DISASTER RECOVERY PLAN: A collection of well planned and tested procedures and data assembled into a single document which, when used, will direct the re-establishment of ALL functions necessary to support day to day business operations, which have been interrupted or rendered inoperative by some uncontrollable event.

One can cite many technical and business reasons for developing and maintaining a disaster recovery strategy, but the fact of the matter is that the subject rarely comes up, and when it does it always gets the lowest priority. In fact, however, it should be first and foremost on the minds and in the planning efforts of the top management of all companies. Trite as it may sound, most companies today can do very little business, if any at all, without the capabilities provided by their computers and networks. Being without a computer is comparable to not having anyone to answer the telephone.

Corporate management and the Boards of Directors are directly liable to the owners and shareholders for the stewardship of the company, and as such are constantly reminded by their auditors that they should maintain a reliable Business Recovery Plan, which includes the complete recovery of all computing facilities available for business operations. One may be led to believe that this type of admonition would cause more companies to put a high priority on developing such a plan, but unfortunately, the advice goes unheeded or gets the lowest of all priorities.

For those that heed the admonition of their auditors and shareholders more advice can be passed along. A business recovery plan is not something that can be developed overnight, scribbled into a 5 page document and then stuffed into the DP Manager's center desk drawer, merely to satisfy the auditor's request in his management report. It is a very serious facet of corporate management's responsibilities. It involves the entire company (personnel, equipment, facilities, profits, etc.) and should receive as much attention as would be given to the development of a major business system.

Preparation of the Business Recovery Plan should also follow the same procedures that are used to develop other systems. Definition, preparation, testing, implementation, and maintenance are the majors areas of concern, and the requirements, as well as the necessary project steps, are the subject of this paper.

Each portion of the development phase is important in its own right, but without the proper planning and definition, none of the other phases can be properly executed. Figure 1.1 illustrates the relative importance of these phases and tasks. Once Phase 1 or the Detailed Requirements Definition is properly completed, the remaining phases become very easy to accomplish and the time for their execution will have been greatly reduced.

DETAILED REQUIREMENTS DEFINITION			
DOCUMENT PREPARATION	IMPLEMENTATION	TEST	MAINT

Figure 1.1

2. DETAILED REQUIREMENTS DEFINITION

There are at least eleven major tasks that must take place in order to complete the major definition of the direction and scope of the Business Recovery Plan. Just as in systems design, the more effort put into the requirements definition phase, the more complete and efficient the Plan will be. These tasks are not all encompassing, but are considered essential to a good plan. They are detailed here to give the developer some guidance and insight into the development process.

2.1 Definition of Scope

How inclusive should the recovery plan be? Document ALL areas of the business that should be included in the plan. This list should include all computer sites to be covered (mainframes, mini's and micros). Other areas of the corporation which are considered critical to its existence should be covered in the plan.

2.2 Catastrophic Events

What types of calamities should be expected? Prepare a list of all possible catastrophic events which could disrupt the operations of the business facilities previously identified.

2.3 Results of Catastrophic Events

What disruptions can be expected as a result of the previously identified calamities? A list must be prepared, detailing specific disastrous results to be expected from each catastrophe that has been identified. In addition, the impact of each catastrophe on the various areas of the corporation should also be defined.

How long should it take to recover? Determine the time required to recover from the results of each catastrophic event that has been defined. Each result of a catastrophic event should be considered individually and an estimate of the time that will be required to return to normal operation for each event should be completed.

2.4 Methods of Recovery

How will we recover? This is decision that must be made by top management. In order for them to make the decision, all alternatives must be documented and presented (Hot sites, cold sites, mutual agreements, dual sites, etc.). Each alternative should be accompanied by a broad range of associated costs.

2.5 Critical System Definition

What recovery priority should be given to each system? This is probably the lengthiest and most important aspect of the detailed requirements definition phase. Each system must be examined in light of being impacted by the defined exposures and by the resulting impact on the corporation's existence. Standard risk analysis programs are available for this process. Having determined the risk, each system should be given a priority and ranked accordingly. This list will greatly influence the definition of the recovery objectives.

2.6 Resource Requirements

What resources are required to execute each system? All resources required by each system must be identified. This includes manual systems as well as those that are automated. Resources include hardware requirements, communication requirements, special operating software, personnel, special supplies, special facilities, vendor participation, outside services, etc. The final document will be a matrix of applications and resources. It will also be used in the definition of the recovery objectives.

2.7 Plan Objectives

Based on the information that has been gathered and refined in the above project steps, a detailed statement of recovery requirements must be composed. This statement, in detail, must set forth all contingency or recovery plan objectives, all areas or functions that will be affected, and a statement concerning the importance of each function's continued operation.

After a complete review of this document, it should be presented to Management for their concurrence.

2.8 Current Plan Review

Is the existing recovery plan, if one exists, still current? If a plan exists, it must be reviewed and compared with the requirements that have been developed in this current exercise. The results of this review will be a schedule identifying the existing plan's strength for each of the currently defined critical systems. These strengths will be considered for entry into the new recovery plan.

2.9 External Services and Products.

What types of products and services are available to accommodate all requirements? In order to accomplish this task, a list of services and products that are required to ensure the plan's success, must be completed. A determination can then be made regarding vendor organizations which can supply these services and products. Meetings should be scheduled with each reliable vendor to determine the scope of each particular service or product and to gather other pertinent information. After the meetings are concluded, a schedule must be prepared listing all requirements which can be satisfied with a particular vendor's product or service. A second schedule will list all requirements not satisfied with external services or products.

2.10 Internal Services

Are there services within the company which will satisfy some requirements? A schedule should be prepared, identifying all processing alternatives which may be available within the organization. Once this is accomplished, a determination may be made as to the capacities required and those available to absorb the processing loads. A schedule can then be prepared showing all costs related to internal processing alternatives.

Finally, a list must be prepared showing all requirements that have not been satisfied with either external or internal services.

2.11 Alternative Summary

The final process that must be accomplished in the detailed requirements definition is the preparation of analysis schedules for each critical system or area of the company, identifying all requirements and all available alternatives. All costs should also be included in the schedule.

In addition, all unsatisfied requirements must be identified and resolved, by determining possible methods, techniques or services which can be utilized to meet the requirements.

Finally, these requirements and their specific solutions must be evaluated, alternatives recommended and documented for a presentation to management. The presentation should result in management's approval to proceed with the actual preparation of a Business Recovery Document.

3. DOCUMENT PREPARATION

All of the detail requirements that have been approved by management must now be translated in detailed procedures and lists which will become the actual document, called "The Business Recovery Plan" or "The Contingency Plan". Figure 3.1 indicates the structure of the document. While this may not exactly fit every company's needs, it represents the very basic elements of a plan which can be tailored to fit any company.

3.1 Table of Contents

This, of course, is the very last section that is completed. It merely indicates the sections and all sub sections with their respective reference numbers. If the plan is developed using a good word processing package, this section will be generated automatically. The table of contents should list procedure numbers rather than page numbers.

DISASTER RECOVERY PLAN STRUCTURE

Section 1	---	Table of contents
Section 2	---	Organization
Section 3	---	Team Definitions
Section 4	---	Backup Procedures
Section 5	---	Recovery Procedures
Section 6	---	Off-Site Recovery Facilities
Section 7	---	Lists/Inventories
Section 8	---	Specifications and Testing
Section 9	---	General Items

Figure 3.1

3.2 Organization

This section deals mainly with the organization of the plan, maintenance responsibility, and the overall structure of a recovery effort. It should contain entries detailing at least all of the following procedures and definitions:

A statement of objectives developed in the detailed requirements definition and formatted for inclusion in the document.

A definition of the number of copies to be distributed and the location and owner of each copy.

A procedure for the on-going maintenance of the plan, with the name and phone number of those responsible.

A form listing all revisions that have been made to the original plan.

A bar chart or gantt chart depicting each event in the recovery process as it relates time wise to all other events.

A procedure detailing the initiation of the recovery plan in the event of a disaster and for the notification of all parties affected by the disaster or participating in the recovery effort.

3.3 Teams

The successful execution of the business recovery plan depends upon the personnel assigned to carry out the various tasks defined in the procedure section of the plan. Figure 3.2 lists many of the various areas requiring some sort of team effort. Each of these areas should have a page in the plan listing the names of a team captain, an alternate captain, and team members. The responsibilities of each team must also be listed on the team page. These responsibilities will be expanded on in procedural form later in the document.

These are just a few of the essential teams needed for effecting a business recovery. Some companies may decide to name the teams differently or to subdivide others to more closely match their own situation. The smaller the business, the fewer number of teams, but the essential functions still remain the same.

DISASTER RECOVERY - TEAM REQUIREMENTS
Recovery Management
Damage Assessment
Facilities Recovery
Operations Recovery
Software Recovery
Communications Recovery
Input Recovery
Special Forms
Logistics

Figure 3.2

3.4 Backup Procedures

Most businesses have a portion of this topic under control, but more than likely it only relates to computer data and programs. The scope of total backup is much broader than computer data and programs. It should cover peripheral systems, documentation, special forms, remote site data, original data, micro computer data and programs, etc.

This section should contain procedures for all backup scenarios, including frequencies and methods for storing or filing of all backup data. All systems design should consider backup as a part of the original design project, thus insuring that the programs and the data are automatically insured against a disaster.

All areas of a business must also consider backup as a part of the normal way of conducting business. Original documents must be marked and stored for easy retrieval, and personnel must be trained in the execution of both the backup as well as the recovery procedures.

3.5 Recovery Procedures

The engine that makes the recovery plan run properly is the recovery and restoration procedure section. Procedures are required for each recovery team as well as for a variety of other recovery related tasks. Figure 3.3 lists a number of procedures that should be included in the plan. This list is by no means complete, but those procedures listed are considered essential.

Each procedure should be written in script form, lelative to each responsible party. No detail should be spared or omitted. A walk-through should be conducted for each procedure before it is included in the plan.

DISASTER RECOVERY - RECOVERY PROCEDURES

Recovery Management
Damage Assessment
Facilities Recovery
Operations Recovery
Software Recovery
Communications Recovery
Input Recovery
Special Forms
Logistics
Alternate Site Recovery
Micro Computer Recovery
Return of Backup Materials
Emergency Shutdown
Halon System
Peripheral Subsystem Recovery
Evacuation
Salvage

Figure 3.3

3.6 Off-site Recovery Facilities

Even if a decision has been made not to employ a hot-site for recovery purposes, certain information relative to off-site recovery processes is essential. Information regarding contracts and other negotiations or agreements should be located in this section. Each off-site location should be described in detail, complete with names and phone numbers of contacts, maps of the immediate area as well as maps of the location itself, and a detailed listing of all available equipment and services.

3.7 Lists and Inventories

Perhaps the most difficult portion of the recovery plan is the collection of all the lists and inventories required to support the many procedures in the recovery section. These lists are varied and any particular list may have many parts, such as the Inventory Checklist. Figure 3.4 contains some of the many lists and inventories necessary for a well thought out recovery plan.

3.8 Specifications and Testing

This portion of the plan should contain specifications describing the construction requirements for the recovery library or off-site storage facility. It should also contain a list of all the required contents. This list will be used in the auditing procedure, which must also be included. The audit procedure should detail the frequency of audits and a method for evaluating the audit results.

The most important part of this section, however, is the procedure for initiating and evaluating tests of the recovery plan.

3.9 General Topics

This section is a catch all for anything that doesn't fit in any of the other sections, such as a procedure for the use of plastic equipment covers.

DISASTER RECOVERY - LISTS & INVENTORIES

- Contact Checklists
  - Employees, Users
  - Vendors, Contractors
- Inventory Checklists
  - Hardware, Software
  - Special Forms, Facilities
- Off-site Storage Facilities
  - Locations, Contents, Maps
- Insurance Information
  - Coverage, Pictures
- Recovery Headquarters
- Data Entry Backup Facilities
- Hot Site/Shell Facilities
- Printing Services
- Travel and Transportation
  - Agencies, Accommodations
  - Delivery Services
- Facility Layouts
- Emergency/First Aid
  - Equipment, Personnel
- Radio & TV Stations
- Emergency Phone Numbers

Figure 3.4

4. IMPLEMENTATION AND TESTING

Now that the recovery plan has been researched, proceduralized and put on paper, it can be installed properly. This is accomplished through training, a final pass to check for inaccuracies, and a thorough test cycle. Without any of these three elements, the recovery plan is totally incomplete.

4.1 Training

To properly insure that all parties involved with the recovery operation are fully informed and versed in the procedures, it is mandatory that orientation and training sessions be conducted prior to full acceptance.

The trainers must first prepare a training course outline, covering both orientation sessions for management and others who need only to be aware of the major functions of the plan, as well as full training sessions for those who will be involved in the execution of all procedures. All training course materials must be prepared for both types of sessions, followed by a schedule for all courses and sessions. Once the courses are completed, the plan is ready to be fully tested.

4.2 Implementation

Before testing commences, a quality assurance test should be made of all required items in the plan. The following assurances should be tested:

That all contracts for alternate processing sites have been negotiated and signed.

That all systems and programming changes that were to be implemented as a result of the plan have been completed.

That all additional equipment has been delivered and installed.

That all additional communications services have been or will be delivered and installed on time.

That all construction, resulting from the plan, has been completed.

That all required supplies and materials have been ordered and will be delivered on time.

4.3 Testing

The final phase of the business recovery plan, before the final stamp of approval can be issued, is the testing phase. Without a proper and conclusive test the plan is nothing but a book, but with the proper testing, management can rest assured, knowing that their business can rise from the ashes of a disaster.

The two most important steps in the testing phase are the "walk-through" and the "test audit". The walk-through will acquaint all parties with their responsibilities, without a large expenditure, and the audit will provide an outside opinion of the viability of the entire plan. Figure 4.1 details some of the items to be included in the testing phase.

DISASTER RECOVERY - TESTING PHASE

Prepare procedures for testing the recovery of all systems
Prepare a test schedule for all systems
Assemble all test materials
Conduct a user "walk-through" to review all test plans
Conduct the test
Record problems and resolve procedures and methods where necessary
Distribute corrections
Notify all parties of formal implementation

Figure 4.1

5. CONCLUSION

If you have followed and completed all the items referred to in this paper, you will have spent a great deal of time and effort for a worthy cause. You probably will have discovered a few weaknesses in some of your systems and hopefully will have corrected them. You have also had to review your hardware situation, your communication network and your micro computer proliferation, and you have probably had to make some adjustments in all three. But, for all your work, your management can rest easier now and the stockholders can feel a little more secure about their investments.

INFORMATION AS A COMPETITIVE WEAPON

BY

DAVID ASHTON

COGNOS INCORPORATED
3755 RIVERSIDE DRIVE
P. O. BOX 9707
OTTAWA, ONTARIO
CANADA K1G 3Z4

Introduction

This paper will evaluate the trend behind Mission Critical Systems (M.C.S.) also known as Strategic Information Systems (S.I.S.). We will see what the forces are behind the S.I.S., including the formulation of corporate strategies. After reviewing some definitions and examples we will explore how to look for and capitalize on opportunities for S.I.S. This will then bring us to evaluate the impact that S.I.S. will have on corporations and in the industry at large.

Although these are four phases to developing S.I.S. (Business strategic plan, Information Systems strategic plan, detailed systems analysis and file structures design), this paper will only address the first two stages. These stages differ the most from regular development process.

Historical Perspective

In the early years of data processing by computers, the limits imposed by hardware (mostly batch oriented main frames) and the software (sequentiale file systems and single tasking operating systems) made it hard to automate anything but very basic functions. So the original areas of automation were the easy ones to automate. We did not spend much time on cost benefits or economic justification analysis. The basis for automation was driven by some characteristics:

- Need to process large number of homogenous transactions
- Regular processing at pre-determined schedules
- Routine processing of highly repetitive transactions that could be collected in batches

By the 1960's, higher level languages like COBOL made programming easier. In addition many companies, having been successful at automating some repetitive paper crunching tasks, started to automate the balance of their back office applications. This expansion into new application areas still dealt mostly with back office operations that were not mission critical. The risks of implementation were still quite low.

During the 1970's, the trend to automation continued, as the costs of hardware kept dropping, while the power of computers increased. Most back office functions were automated one-at-a-time with little or no integration among them. Two significant trends of the 1970's were the mini-computers and the packaged software solutions.

The early 1980's saw tremendous advances in the industry. The hardware environment has seen 32 bit mini-computers and powerful micro-computers dramatically increasing the price/performance relationship. The software component has seen fourth generation languages and user driven tools for the managers. Also, advances in telecommunications and networking are now opening up databases to users at remote locations.

A New Era: Mission Critical Systems

The rapid evolution of our industry has increased the limits of what can be done. In the past decade, the raw power of computers has increased 18 fold. Between 1958 and 1980, the time to execute one electronic operation has decreased by a factor of 80,000,000. It is now 8,000 times less expensive to process information by computer than it was by manual operation 30 years ago. Through the use of 4GL's, it is 50 to 100 times faster to build applications than it was in the early 1960's.

As a result, we are now poised at the edge of a second, and ultimately more important, wave of automation that holds the promise of changing the way that business is conducted. We are now entering the phase of mission critical (or strategic) automation that will see Information Services (I.S.) evolve from the operational and tactical to the strategic level of corporations.

A mission critical (or strategic information) system (S.I.S.) is a system which directly supports the creation and implementation of an organization's strategic plan. If successfully implemented, an S.I.S. can provide significant competitive advantages through increased product differentiation, improved customer and supplier relationships, altered industry structures and even brand new business opportunities.

The classic case of a S.I.S. is American Airline's Sabre reservation system. When the system was started management decided to use it as a competitive tool. It was decided that Sabre should provide all airlines' schedules to make it more attractive to travel agents, who represent the distribution network for airline seats. As a result of this strategic decision, Sabre became a tool for travel agents. American Airlines listed their own flights first and many travel agents never went further, resulting in increased market share for them. Today, Sabre is used by 48 percent of the automated travel agents in the U.S. and it generates net earnings of over $170 million for it's parent company.

Formulation of the Corporate Strategy

Before we take a look at the M.I.S. component of Strategic Information Systems, let's discuss how a corporation sets up a strategy. The ideas for the formulation of a strategy came from Michael E. Porter's, " How Competitive Forces Shape Strategy" in the Harvard Business Review of March-April 1979.

Once a coporation's strategic planning team has assessed the forces affecting competition in their industry, they can identify their own strengths and weaknesses.

The action plan is then devised. It can include:

1) Positioning the company in a way that its capabilities provide the best defense against competitive forces;

2) Influencing the balance of the forces through strategic moves in order to improve the company's position;

3) Anticipate shifts in the market place and respond to them with a competition strategy before the competitors take action.

The most forward thinking companies have already established the new function of Chief Information Officer who participates in the strategic planning process. Their role is to look for ways that the information technology can help meet the strategic plan. If there is no such person in an organization, it is the M.I.S. director's role to understand the corporate mission, it's business and the competitors. To help understand the business, the M.I.S. director might arrange to spend a few days on the road with one of the sales representatives, meeting customers and hearing them talk about their needs. Another way would be to spend a few days on the telephone, answering requests from customers and/or suppliers.

Next, the M.I.S. Managers need to understand how their senior executives think. It is important to understand the style as well as the vision of those executives. The M.I.S. Manager must then become creative in thinking about a way that Information Technology (I.T.) can serve. While dealing with the executives, it is important to remember that there are two distinct classes of executives: the administrator and the manager. An administrator believes that whatever is not specifically permitted is prohibited. A manager thinks that what is not forbidden must be permitted. It is usually the "manager" type that will drive new strategic applications.

How to look for S.I.S. Opportunities

We have seen that the new technology has opened up new opportunities for a company to re-deploy its assets and re-think its strategy. With a S.I.S., the stakes are so much higher than with regular back office applications that this must be a well-planned decision. When technology had limited function, you were not betting the company on it; with mission critical systems, you do.

In "Information Technology Changes the Way You Compete", Harvard Business Review May-June 1984, F. Warren McFarlane came up with a way to evaluate the ultimate impact of S.I.S. Companies must search answers to five questions.

1. Can we build barriers to entry with a S.I.S.?

   For example, a distributor built an on-line network allowing customers to enter orders directly in their computer. As a result, customers can get access to delivery dates and availability of stock, as well as suggested replacement items. This move was very successful and gained high acceptance by customers. It also built an entry barrier preventing other competitors to replicate the approach. Customers did not want computer equipment from several vendors on their premises.

2. Can switching costs be increased with a S.I.S.?

Can we encourage customers to rely increasingly on our electronic support? This is done by building the system into their operations to create increased operational dependency, making switching to a competitor more expensive. A kitchen cabinet manufacturer has built a system to help determine the materials needed to remodel or build a kitchen. Contractors can dial in, get a plan and quote for material that the cabinet maker sells. Switching to another supplier would be very difficult as this tool is saving time for contractors.

3. Can we change the basis of competition with a S.I.S.?

In a significant paper for Harvard Business Review of July-August 1985, M.E. Porter and V.E. Millar introduced the concept of the value chain to evaluate competitive moves. A company's value chain is a system of interdependent activities connected by linkages. To gain competitive advantage over competitors a company must perform those activities at a lower cost or in a way that leads to differenciation and a premium price (more value).

In the mid 1970's a major distributor of magazines concluded that they were in an industry segment dominated by cost driven competition. It changed the basis of competition by using I.S. to identify what was selling on the customer's newsstand and compared the profitability by square foot with data from other newsstands in similar socio-economic areas. It was able to improve the product mix of its customers, allowing the company to raise it's prices and shift the competitive nature of the segment from cost to product differentiation.

4. Can the S.I.S. change the balance of power in supplier relationships?

   A retailer has hooked his purchasing system to the order entry systems of his major suppliers. The result means that the retailer can monitor inventory levels of suppliers, alert them when they expect to place a large order and shop for best prices all around. In a cost competitive segment like retailing, a system that supports better, more reliable and cheaper buying activities is viewed as strategic.

5. Can the S.I.S. generate a new business?

   Could the computer systems' extra capacity or the company's corporate data be turned into a product?

   For example, Citibank, which offers financial data and services has teamed up with McGraw-Hill, which collected data on commodities. They created a joint venture called Global Electronic Markets Co. that allows traders to get information instantly, 24-hours per day, and to make deals and transfer money.

How to Capitalize on the Opportunities

Now that we have discovered S.I.S. opportunities, how do we start capitalizing on them?

1. Evaluate the information intensity of products

   A company's executives should start by evaluating the information content of their own products as well as the information intensity of the value chain. The degree to which information technology can be used to competitive advantage depends on actual and potential information intensity of the business' products. As implied in Table 1, a high intensity in the value chain might include a large number of complex processes to build the product, a large number of customers or suppliers dealing directly with the company, products whose acquisition requires the customer to be extensively trained. The information intensity of the product would include products and/or services whose content is mostly information.

The most likely S.I.S. areas would be in the high product information and high intensity of the value chain.

2. Determine the role of I.T. in industry structure

The company's executives must understand how I.T. may structurally charge their industry and look for ways to lead the change. USA Today is an example of fundamental change to an industry's structure. This daily newspaper, with distributed printing facilities and with articles shipped by electronic transmission, changed the competition from a regional scope to a national one. They have also used technology to give a different look to the paper.

3. Find and rank potential competitive advantages for S.I.S.

Each product or service has both a physical and an information component. An Information Technology competitive advantage is created when the information component of the product is altered, resulting in an advantage over rival forces. The questions that executives must ask themselves is: what additional information could be bundled with the product? Will this result in a sustainable competitive advantage?

The activities of the value chain that represent a significant portion of the total cost or that could best be used for differenciation usually have the most potential for automation.

INFORMATION

INTENSITY MATRIX

INFORMATION INTENSITY OF THE VALUE CHAIN	LOW	HIGH
HIGH	OIL REFINING	BANKING NEWSPAPERS AIRLINES
LOW	CEMENT MANUFACTURING	INSURANCE BROKER

INFORMATION CONTENT OF THE PRODUCT

4. Develop a S.I.S. plan

The action plan will have to include the investments required in hardware, software tools and development time. Any structural changes induced by the S.I.S. should also be documented and agreed upon. The impact on existing management processes and systems must also be evaluated, as some systems could become absolete with the implementation of the S.I.S.

Because of the complex nature of S.I.S., outside help may be needed to support the system. Finally, a realistic implementation plan must be drawn. The implementation schedule must include time for presentation to, and approval by, senior mangement as well as enough time to possibly change parts of the company's culture.

The S.I.S. plan should support the overall corporate strategic plan; it should not dominate it.

The Challenge of S.I.S.: What Needs to Change Inside Companies.

In order to effectively make use of the new advances in information technology, corporations and their information services will need to make several changes:

1. Information Services Officers

It is important for companies to create a new function of Chief Information Officer (C.I.O.) reporting directly to the C.O.O (Chief Operation Officer). The C.I.O. must become part of the strategic planning team to ensure that information services are sought, understood and planned for within the executive committee. The presence of the C.I.O. will also bring options that otherwise would not be even dreamt of.

This also means that the traditional role of the M.I.S. manager will evolve. He/she will have to grow into a business strategist and problem solver. Rather than controlling the technology, the C.I.O. will be coordinating architectures and standards and provide coaching to several information systems units for their development needs.

2. Companies will need to increase I.S. spending

As mission critical systems become a competitive necessity, corporations will have to increase their expenditures on I.S. by a factor of two to three times over the next 5-10 years. This will be caused by the enormous complexity of S.I.S. as well as by the fact that many companies will have to replace or significantly upgrade their current systems in order to integrate them with the new S.I.S.

Companies that resist this increase will find themselves fighting competitors at a rapidly increasing disadvantage. Even companies that take a technology lead will have to keep investing to remain ahead of a reacting competition.

3. Simple rules guiding I.S. expenditures must disappear

Comparing I.S. costs to other companies' performance via a percentage of sales or profits is very dangerous and short sighted. The market leader of tomorrow may be the greatest spender of today. Then again, the spender of today may be dead tomorrow. The I.S. expenditures should be measured against the total strategic potential and appropriately funded.

This also means that systems justification should no longer be based on a R.O.I. (return on investment) or cost savings alone. This justification process is typically cost driven and made by a financial officer. It generally excludes the positive impact of a S.I.S. on revenues.

4. Corporations must learn to protect the confidentiality of their S.I.S. plans

   Nobody would think of discussing a new manufacturing process with a competitor. Yet when it comes to data processing, papers are presented at industry meetings discussing the corporation's strategic implementation of systems. This co-operation has been with the industry since its inception. It existed because no strategic advantages were derived from data processing. Co-operation helped solve computer/technical problems. Now that strategic advantages can be obtained from technology, companies must become more careful and protective of their I.S. plans.

5. Creativity must enhance logic

   Problems should be tackled with a new wave of creativity to complete the logic that has driven I.S. New technologies can now be incorporated into very creative solutions that may change the way that business is conducted, who the competition is and even what makes the product. The boundaries around the problem solving have expanded considerably. This new dimension must be included in the creation process of problem solving.

Impact of S.I.S. on the Industry

The whole data processing industry will feel the impact of strategic information systems being developed by more companies. The first impact will be an increase in spending that will create good opportunities for well positioned suppliers.

We can also expect more standards to be enforced by the vendors of hardware, software, and telecommunications. These will be required by companies as they try to integrate solutions from several sources.

The software tools and languages vendors in particular will be forced to view logical transactions in network environments and provide solutions for that new concept. The best positionned software tools vendors are those that already support complete logical transactions across different file structures. The logical transactions will be needed to ensure data integrity, rollback and recovery across several systems.

Companies will demand application solutions that have built-in logic and data exchange facilities to integrate with other systems. The corporations will also require that any application package be easily customizable to satisfy their strategic needs. This will favor solutions written with fourth generation languages.

We can also expect new methodologies and approaches to be built for developing and implementing S.I.S. Most likely these methodologies will be developed for fourth generation languages.

A new consulting/system integration industry segment will emerge to tackle S.I.S needs. Their main suppliers will be the leading edge and multi-environments hardware vendors, the fourth generation languages and associated methodologies creators and the packaged software developers using 4GL technology and open design architectures.

Conclusion

We are now entering the next era of data processing. Some have rightly called it the information age. Information systems are emerging as the critical next battle groud of corporate wars. Information is moving from being a defensive weapon to an offensive one that carries a lot of might. It will leave behind a lot of market leaders and create many new and exciting opportunities. Is your company ready for it?

Experiences in Migration

James S. L. Cohen
Mecca Leisure Ltd
76 Southwark Street
London SE1 0PP

1. Introduction

Much has been written and said over the last eighteen months about the ease (or lack of ease) while migrating from a 'classic' HP3000 to a Precision Architecture HP3000. It is the intention of this paper to look at the steps taken at Mecca Leisure to complete a successful migration from an HP3000 Series 70 to an HP3000 Series 950 - in a true user environment.

The paper will discuss the hardware and software related issues of migration, and the preparations that should be made well prior to delivery of the HP3000 Series 900.

It is intended to answer as many of the questions, you might have regarding the ins and outs of migration, without clouding the issue by getting unnecessarily technical.

Since it is the intention to look at the experiences that Mecca actually had, it is necessary to undertake a very brief look at the configuration prior to the migration. Mecca Leisure was running two, HP3000 Series 70's, one being used for Office Automation and one being used for commercial applications, such as Accounts, Marketing Systems and Payroll. The intention was to upgrade the second of the two HP3000's. Figure 1.1 is a schematic of the configuration prior to the installation of the HP3000 Series 950 and Figure 1.2 the final configuration.

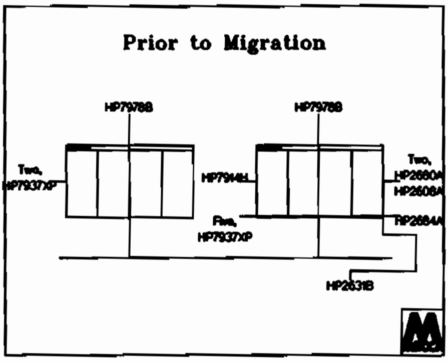


Figure 1.1

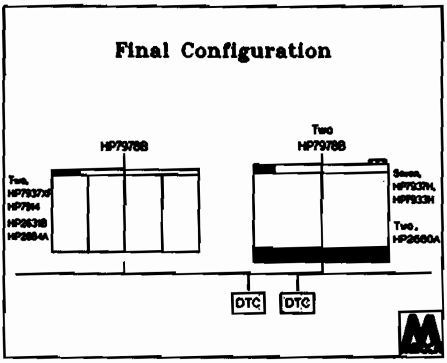


Figure 1.2

Connected to the machine that was to be migrated, were a number of devices that would not be supported on the HP3000 Series 950 (for example X.25 connected via INP's) and printers such as an HP2608A (these too would not be supported on the HP3000 Series 950).

When migrating from a 'classic' HP3000 to a Precision Architecture system, it is important that you decide what level of migration is acceptable in the first instance. Many of your application systems might not be best suited to migration. If there are many calls to subsystems that require compatibility mode segmented library calls, these programs might best be left in compatibility mode. Programs written in SPL, that do not require Privilege Mode, could be well left as they are, and your finite resources (programmers) made to concentrate on those programs that can be easily migrated to the maximum benefit of the company.

2. Prior to Order

Once you have decided that for what ever reasons you require to upgrade from a 'classic' HP3000 to a Precision Architecture HP3000, it is essential that you have a careful look at the peripherals you have hung off your current HP3000. There could well be a number of peripherals, that will not be supported at first (or subsequent) release of MPE/XL. Failure to list and check every single terminal type (sometimes with ROM release), every printer, every tape drive and every disc drive, could well result in you having to order peripherals at the last moment, which in turn could well escalate the costs significantly.

For example :-

- Have you invested in HP7933/37 XP disc drives ?

  These are not supported, but if you ask your Engineering Manager nicely, he will downgrade them to HP7933/37 H drives for you. In due course, when the XP drives are supported, they will be upgraded again for you.

- Have you any HP2624A's left ?

  These need upgrading. However, beware of the HP700 series terminals; these are not supported (but if you configure the terminal identifier to 2392A, they seem to work fine).

Why is it that so few current devices are supported on the HP3000 Series 900's ? Is it because they are so fundamentally different ? It is simply because Hewlett Packard, didn't have time to test every existing/emerging peripheral device, so although you should step with care, it is not impossible to use nominally unsupported peripherals. Similarly it would have been a waste of effort to write the necessary drivers for obsolete devices, rather than concentrate of optimizing MPE/XL for current peripherals.

If you have any doubts, as to peripheral support the best single point of reference is a document published by Interex following the Detroit conference in which all peripheral and their levels of support across the entire HP3000 range is listed.

3. After Order has been Placed

Now that you have finalised your order and convinced your management into the purchase of a Series 900, the fun really can start. Assuming that Hewlett Packard have given you a delivery date, you have little time to relax.

- You must analyse all the code you have currently running on your system, and if you use code developed and maintained by third parties you must get them to analyse their code.

- You must decide which applications (or part thereof) you are going to migrate fully or partially to Native Mode and which applications are going to remain in Compatibility Mode. Remembering of course that the only Hewlett Packard's supplied Native Mode compilers are COBOL (74 and 85 ANSI standard), Fortran '77 and Pascal. If you have code in SPL it either needs rewriting or the use of a third party product.

- You must develop a full migration plan, covering the installation, testing and final switch over to the new system.

What criteria should you use in determining whether or not to migrate to Native Mode ?

Below is a list of questions, that I believe has to be answered :-

- Is the program written in a supported Native Mode Language ?

- Does the program call any routines, that themselves cannot be migrated into Native Mode ?

- Is the program regularly run ?

- Is the program to be run on both Classic and Precision Architecture systems, perhaps at different sites ?

- What is your Disaster Plan ? Do you intend to utilize a Classic HP3000, as part of your Disaster Plan ?

- Are you making use of third party products, (that themselves remain in Compatibility Mode) ?

The answer to all these questions effects your decision in how far to go in migration, how many of the new features you are going to make use of in the short term.

In effect the question could be :

"How much RISC are you going to take ?"

because not only have you to decide as to how far you are to go with Native Mode, but also how many of the new features you are going to permit your programmers to use. These features are not just limited to new intrinsic calls (SORTINIT/HPSORTINIT) but also do you use SETVAR instead of SETJCW.

So now lets start looking at specific things for you to do.

4. Training

It may appear almost irrelevant, but get your training booked now, ensure that all relevant staff attend the two MPE/XL courses (Systems Manager and Programmers). Beware however, these courses are not suitable for staff who have not had considerable exposure to HP3000's, they are very much conversion courses. At time of writing no MPE/XL for beginners courses are available, although they must be under consideration by Hewlett Packard.

The two courses are a three day, MPE/XL Systems Manager Course and seven day MPE/XL Programmers Course. While the Systems Manager course is common across most HP3000 installations, the Programmers course covers a multitude of systems based applications that you may not need to know or migrate. For example, how to migrate Fortran'77 where real numbers are used, how to migrate SQL/V to SQL/XL. It is quite likely that many of the constituent parts of the course are not relevant to your installation, if this is the case consider getting Hewlett Packard to run your own subset of the Programmers course. In the case of Mecca the seven day course was reduced to three days, without any problems.

The reduction in length of the programmers' course to three days would only be cost efficient if you are running your own in house training.

5. Analysis of your Application Software

Now that you are aware of the new features and internal architecture of the Precision Architecture HP3000's, it is time to take a long clear look at the application software running on your HP3000.

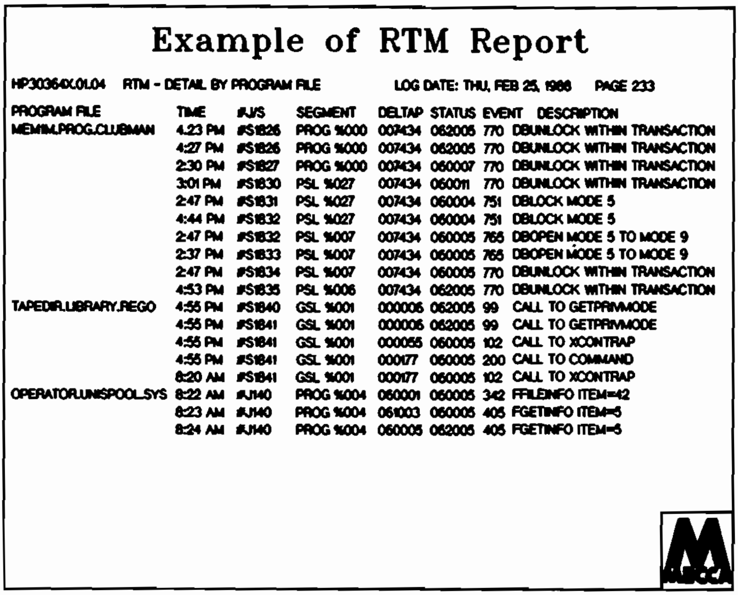
Hewlett Packard have produced a number of analysis tools to help you in the process (Product Number HP32428A) and you should arrange to have them installed as early as possible in your migration process on your classic HP3000.

There are a number of elements to the Migration Toolset, that will help you identify most, but not all, of the areas where some effort in migration is required. The Migration Toolset will find most of the code incompatibilities, such as intrinsics that are no longer available on the Precision Architecture HP3000's. The Run Time Monitor makes heavy use of the system log files. and it is essential that you have enough disc space to handle these.

5.1 Run Time Monitor

The RTM must be run for at least one full period (ie at least one calendar/accounting month), so that it has a chance to monitor all the events you expect to migrate, since it monitors executing applications recording events into the MPE log file. When starting RTM it is possible to determine which of eight events to monitor, however there is practically no system overhead in using RTM and I would recommend that all events be logged. Keep a close eye on the disc space being used by MPE log files, if you were using one 2046 sector log file a day, you could now be using four. Your system could soon clog up with MPE log files, if this is the case - then update your MPT database and then store the files to tape using LOGSNAP (from the TELESUP account). There is no noticeable overhead in running this product, although Hewlett Paçkard do warn you that you could overrun a stack (this was never experienced at Mecca, and can be avoided by running programs with NOCB at runtime).

A report program is provided to report directly on the MPE log files, but a more meaningful report can be generated by using the log files in conjunction with MPT (See 5.3). A quick look at a summary report from RTM is worth while and why it reports on certain events.



Example of RTM Report

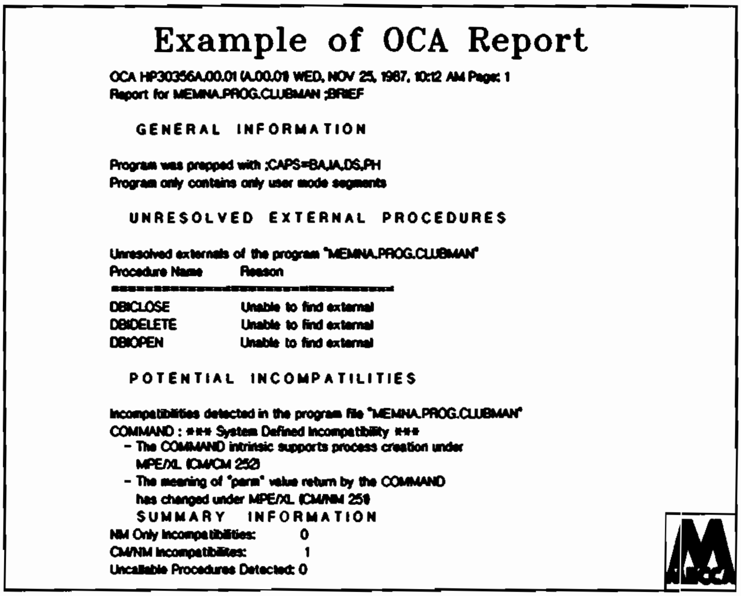
```
HP30364X.01.04   RTM - DETAIL BY PROGRAM FILE          LOG DATE: THU, FEB 25, 1988     PAGE 233

PROGRAM FILE            TIME     #J/S     SEGMENT     DELTAP STATUS EVENT  DESCRIPTION
MEMIM.PROG.CLUBMAN      4:23 PM  #S1826   PROG %000   007434 062005 770 DBUNLOCK WITHIN TRANSACTION
                        4:27 PM  #S1826   PROG %000   007434 062005 770 DBUNLOCK WITHIN TRANSACTION
                        2:30 PM  #S1827   PROG %000   007434 060007 770 DBUNLOCK WITHIN TRANSACTION
                        3:01 PM  #S1830   PSL %027    007434 060011 770 DBUNLOCK WITHIN TRANSACTION
                        2:47 PM  #S1831   PSL %027    007434 060004 751 DBLOCK MODE 5
                        4:44 PM  #S1832   PSL %027    007434 060004 751 DBLOCK MODE 5
                        2:47 PM  #S1832   PSL %007    007434 060005 765 DBOPEN MODE 5 TO MODE 9
                        2:37 PM  #S1833   PSL %007    007434 060005 765 DBOPEN MODE 5 TO MODE 9
                        2:47 PM  #S1834   PSL %007    007434 060005 770 DBUNLOCK WITHIN TRANSACTION
                        4:53 PM  #S1835   PSL %006    007434 062005 770 DBUNLOCK WITHIN TRANSACTION
TAPEDIR.LIBRARY.REGO    4:55 PM  #S1840   GSL %001    000006 062005 99  CALL TO GETPRIVMODE
                        4:55 PM  #S1841   GSL %001    000006 062005 99  CALL TO GETPRIVMODE
                        4:55 PM  #S1841   GSL %001    000055 060005 102 CALL TO XCONTRAP
                        4:55 PM  #S1841   GSL %001    000177 060005 200 CALL TO COMMAND
                        8:20 AM  #S1841   GSL %001    000177 060005 102 CALL TO XCONTRAP
OPERATOR.UNISPOOL.SYS   8:22 AM  #J140    PROG %004   060001 060005 342 FFILEINFO ITEM=42
                        8:23 AM  #J140    PROG %004   061003 060005 405 FGETINFO ITEM=5
                        8:24 AM  #J140    PROG %004   060005 062005 405 FGETINFO ITEM=5
```

Figure 5.1

5.2 Object Code Analyser

A second application called the Object Code Analyser should be run in conjunction with RTM. The main difference between RTM and OCA, is that RTM analyses object code as it is run, while OCA cannot determine run time parameters. The OCA scans designated accounts and analyses SL and files with the file code of 'PROG' and reports on potential incompatibilities in all object code, within the designated account(s). I would highly recommend that when you are running OCA that you logon into the DS queue, otherwise you might well grind your existing applications to a halt (or run OCA in batch).

Since OCA cannot determine run time parameters, it cannot be sure if an intrinsic call is incompatible in Native Mode or not. Therefore it reports on all suspect intrinsic calls.



Example of OCA Report

```
OCA HP30356A.00.01 (A.00.01) WED, NOV 25, 1987, 10:12 AM Page: 1
Report for MEMNA.PROG.CLUBMAN ;BRIEF

   GENERAL INFORMATION

Program was prepped with ;CAPS=BA,IA,DS,PH
Program only contains only user mode segments

  UNRESOLVED EXTERNAL PROCEDURES

Unresolved externals of the program "MEMNA.PROG.CLUBMAN"
Procedure Name        Reason
==============================================
DBICLOSE              Unable to find external
DBIDELETE             Unable to find external
DBIOPEN               Unable to find external

  POTENTIAL INCOMPATILITIES

Incompatibilities detected in the program file "MEMNA.PROG.CLUBMAN"
COMMAND : ### System Defined Incompatibility ###
  - The COMMAND intrinsic supports process creation under
    MPE/XL (CM/CM 252)
  - The meaning of "parm" value return by the COMMAND
    has changed under MPE/XL (CM/NM 251)
    SUMMARY  INFORMATION
NM Only Incompatibilities:        0
CM/NM Incompatibilites:           1
Uncallable Procedures Detected: 0
```

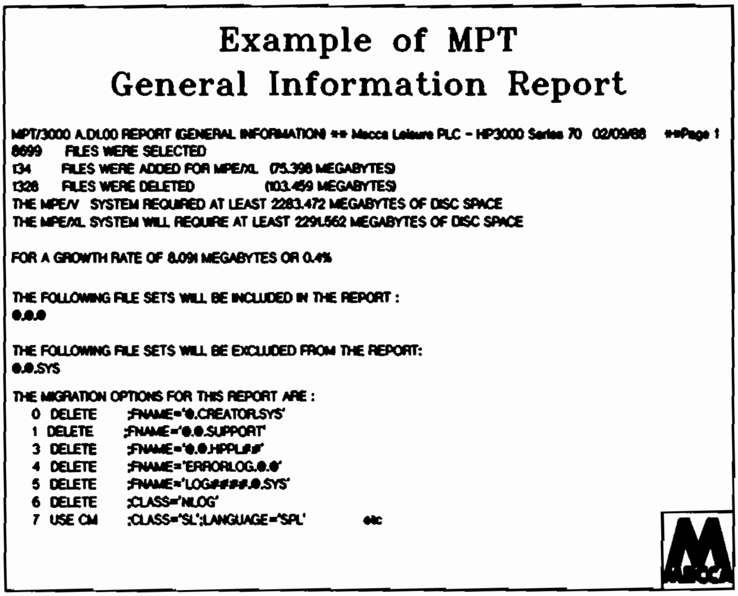
Figure 5.2

Figure 5.2 is extracted from an OCA report and as can be seen it produces a considerable amount of information, most of it is purely detail, however it is well worth the effort of checking as it reduces the amount of work required in migration, by identifying only those programs that require any effort.

5.3 Migration Planning Tool

The third application and the one that pulls all the migration toolset products together is called the Migration Planning Tool. MPT fulfils two issues, the first is predicts the disc requirements required for Native Mode (of which more later) and extracts from the MPE log files events recorded by RTM. All the information from MPT is recorded in an Image database and a number of reports can be generated from it. Each one serves a different purpose and it is worth having a brief look at each. There are seven levels of report and a General Information Report.

The General Information Report (Figure 5.3) gives you an estimate of the growth in disc space required. The growth in disc space assumes a fully Native Mode final system, if you have any Compatibility Mode code (that you have run through the Object Code Translator) your programs will occupy at least five times more disc space than they currently occupy. Some data files are also likely to grow, if you align all your data on 32 bit, rather than 16 bit word boundaries.



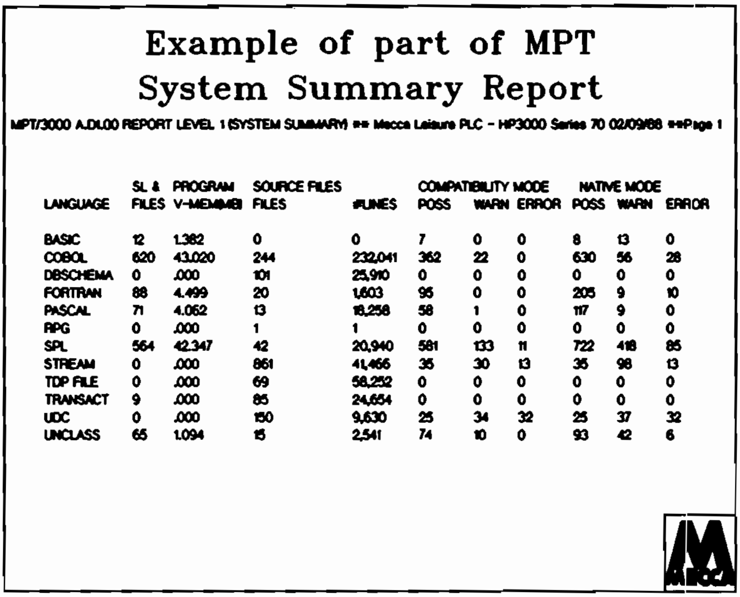
Example of MPT General Information Report

```
MPT/3000 A.01.00 REPORT (GENERAL INFORMATION) ** Macca Leisure PLC - HP3000 Series 70  02/09/88    **Page 1
8699    FILES WERE SELECTED
134     FILES WERE ADDED FOR MPE/XL  (75.398 MEGABYTES)
1328    FILES WERE DELETED              (103.459 MEGABYTES)
THE MPE/V  SYSTEM REQUIRED AT LEAST 2283.472 MEGABYTES OF DISC SPACE
THE MPE/XL SYSTEM WILL REQUIRE AT LEAST 2291.562 MEGABYTES OF DISC SPACE

FOR A GROWTH RATE OF 8.091 MEGABYTES OR 0.4%

THE FOLLOWING FILE SETS WILL BE INCLUDED IN THE REPORT :
@.@.@

THE FOLLOWING FILE SETS WILL BE EXCLUDED FROM THE REPORT:
@.@.SYS

THE MIGRATION OPTIONS FOR THIS REPORT ARE :
   0  DELETE      ;FNAME='@.CREATOR.SYS'
   1  DELETE      ;FNAME='@.@.SUPPORT'
   3  DELETE      ;FNAME='@.@.HPPL##'
   4  DELETE      ;FNAME='ERRORLOG.@.@'
   5  DELETE      ;FNAME='LOG####.@.SYS'
   6  DELETE      ;CLASS='NLOG'
   7  USE CM      ;CLASS='SL';LANGUAGE='SPL'          etc
```

Figure 5.3

The Level 1 System Summary Report (Figure 5.4), will tell you something that you probably never knew before, namely how many programs you have are written in each language, it also gives you the file count by file code.

Given that TELESUP, PUB.SYS etc are ignored, this gives you a fairly clear picture of what object code you have on your system. In Figure 5.4 is an abstract of the actual report from Mecca Leisure's HP3000 Series 70, with the object code the report shows how many potential problems there may be when moving to Native Mode. The second page, dealing with non object code files, gives an indication of the growth in disc space required to handle the data files (not shown).



Example of part of MPT System Summary Report

MPT/3000 A.01.00 REPORT LEVEL 1 (SYSTEM SUMMARY) ** Mecca Leisure PLC - HP3000 Series 70 02/09/88 **Page 1

	SL &	PROGRAM	SOURCE FILES		COMPATIBILITY MODE			NATIVE MODE		
LANGUAGE	FILES	V-MEM(MB)	FILES	#LINES	POSS	WARN	ERROR	POSS	WARN	ERROR
BASIC	12	1.382	0	0	7	0	0	8	13	0
COBOL	620	43.020	244	232,041	362	22	0	630	56	28
DBSCHEMA	0	.000	101	25,910	0	0	0	0	0	0
FORTRAN	88	4.499	20	1,603	95	0	0	205	9	10
PASCAL	71	4.062	13	18,258	58	1	0	117	9	0
RPG	0	.000	1	1	0	0	0	0	0	0
SPL	564	42.347	42	20,940	581	133	11	722	418	85
STREAM	0	.000	861	41,466	35	30	13	35	98	13
TDP FILE	0	.000	69	58,252	0	0	0	0	0	0
TRANSACT	9	.000	85	24,654	0	0	0	0	0	0
UDC	0	.000	150	9,630	25	34	32	25	37	32
UNCLASS	65	1.094	15	2,541	74	10	0	93	42	6

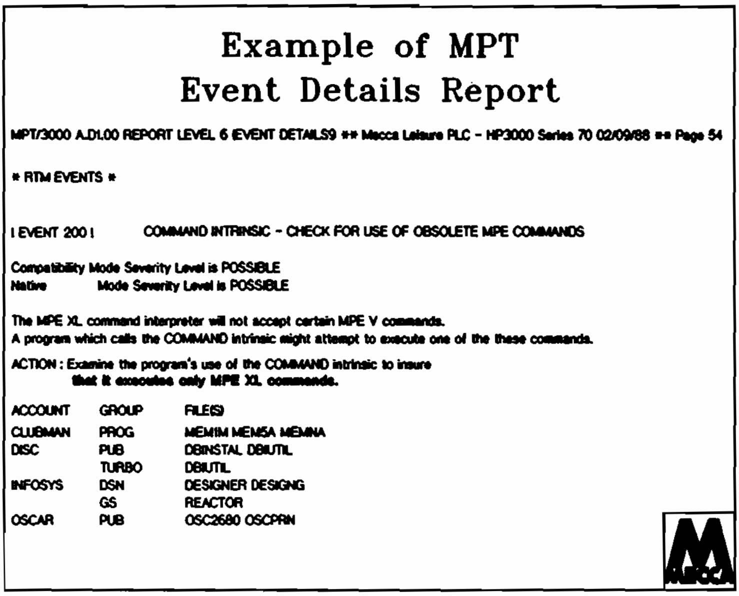
Figure 5.4

The Level 2 report is an account summary of the information given on a system wide scale in the Level 1 report, and the Level 3 report a group summary.

Level 4 is a file summary of all the files on the system, their file code, file size etc. If the file is a program it lists the number of possible inconstancies, the number of warnings and the number of errors likely to be encountered on a Compatibility and Native Mode basis. The report also includes such details as the stack size (Compatibility Mode only) and the likely increase in disc space in Native Mode. On a medium size system the Level 4 report runs to about 400 pages.

Level 5 of the MPT report is an even longer report, but reports by file each possible error encountered. For example, a job (or UDC) that compiles is likely to have reference to the PREP command, since this is replaced by the LINK command under MPE/XL, this file has as possible error reported on it.

However the report that was found to be the most useful, and only one hundred pages long for the full system, was the Level 6 report on event details. This contains a page per questionable event, and then lists all the files in which the event occurs.



Example of MPT Event Details Report

```
MPT/3000 A.D1.00 REPORT LEVEL 6 (EVENT DETAILS9 ** Mecca Leisure PLC - HP3000 Series 70 02/09/88 ** Page 54

* RTM EVENTS *

! EVENT 200 !         COMMAND INTRINSIC - CHECK FOR USE OF OBSOLETE MPE COMMANDS

Compatibility Mode Severity Level is POSSIBLE
Native        Mode Severity Level is POSSIBLE

The MPE XL command interpreter will not accept certain MPE V commands.
A program which calls the COMMAND intrinsic might attempt to execute one of the these commands.

ACTION : Examine the program's use of the COMMAND intrinsic to insure
         that it executes only MPE XL commands.

ACCOUNT    GROUP    FILE(S)

CLUBMAN    PROG     MEM1M MEM5A MEMNA
DISC       PUB      DBINSTAL DBIUTIL
           TURBO    DBIUTIL
INFOSYS    DSN      DESIGNER DESIGNG
           GS       REACTOR
OSCAR      PUB      OSC2680 OSCPRN
```

Figure 5.5

The only slight problem with this report is that it was written with HPImage in mind. All 'DB' calls that cannot be directly changed to an 'HP' call are reported on. More usefully any program calling the COMMAND intrinsic is listed, since the formats of some MPE commands have changed and could affect programs (for example LISTF returns a slightly different result).

Using the information you have gathered you are now in a position to consider migrating. Hopefully Hewlett Packard would have assigned to you a migration trained AE, he will help you analyse the reports and advise you on what changes are required. For example, job streams that compile can be changed well in advance to any attempt to migrate.

Given that Mecca Leisure do not utilise any non standard code, what doesn't the MPT/OCA reports tell you ?

- There are problems (bugs) in linking Fortran'77 routines called from Cobol. In fact it appears easier either to recode the Fortran into Pascal or Cobol, or to leave the programs in Compatibility Mode.

- There is a change in the number of parameters when calling SORTINIT/XL. The sixteenth (called 'spare' on SORTINIT/V) no longer exists.

- Calls to the KSAM intrinsics from Cobol are suspect, but can easily be replaced by REWRITE etc.

- If you attempt a CKOPEN to a non KSAM file, on the MPE/V machines you will receive a warning tombstone, on an MPE/XL machine you will suffer an abort (with no tombstone).

- If you divide by zero in COBOL/V the result of the divide is zero ?! On COBOL/XL an attempt to complete a divide by zero results in an error, which can be either fixed by using the ON SIZE ERROR code or by setting a run time variable.

- Does not report on calls to COBOLLOCK/COBOLUNLOCK intrinsics which are obsolete on Precision Architecture HP3000's, as they are superseded by the EXCLUSIVE/UNEXCLUSIVE statements in Cobol.

- You might have heard that a file can be built of infinite size, yet occupy zero sectors and zero extents. This is true, so long as you call the HPFOPEN intrinsic programmatically. If you use the BUILD command, it calls the FOPEN intrinsic, and you remain restricted to current MPE/V file limitations.

5.4 Analysis from Migration Toolset

Using all this information, makes determining where effort is needed considerably easier, but the reports need careful filtering. Much of the information is hidden in a cloud of errors reported on many Image calls, that would have needed changing when and if you move from TurboImage to HPImage. It is a great shame that the effort was not made to remove the HPImage errors, once it had been decided to migrate TurboImage to Native Mode and to delay HPImage.

6. Suggested Steps in Migration

Now that you have the information necessary in determining what changes are required in migrating your system, you can make many of the changes prior to formal migration :-

- any SPL routines that you are going to rewrite, say into Pascal, should be rewritten and tested on an existing HP3000. This is important as it will ensure that any problems you have will be limited to purely migration issues. Any attempt to change code simultaneously with migration is liable to fail.

- when you come to migrate, migrate initially using 16 bit alignment (as used on the 'classic' HP3000), once you are sure that the application is fully migrated, then and only then consider recoding to utilise 32 bit alignment.

- never consider making program changes, as part of the migration process. Make any changes on a classic HP3000, fully test the changes, and then migrate.

6.1 Migration Steps

The steps to be followed, once you have access to a Precision Architecture HP3000, whether in a Migration Centre or your own utility, should be as follows :-

- Test Programs in Compatibility Mode

- Put any programs that are to remain in Compatibility Mode, through the Object Code Translator

- Test Programs in Native Mode, with no optimization

- Test Programs in Native Mode, with optimization

- Change data alignment

The decision as to whether or not to change the data alignment should be made based on the answers to the questions listed in Section 3. However, if there is any intention to run the same programs and data on a 'classic' HP3000 as on the Precision Architecture HP3000, then remember to keep Native Mode and CM mode programs (in separate groups).

To make the transfer of accounting and UDC information as simple as possible there is a program on the Precision Architecture HP3000's called DIRMIG that will migrate these. Using the DIRMIG command on a Precision Architecture HP3000, rebuilds your accounting structure and sets the UDC's. Restore the accounts and ensure that all your systems (with the possible exception of some Privilege Mode code) work in Compatibility Mode. Mecca's experience of this is that all our own and third party applications work well. Before using DIRMIG use the opportunity to clean up your COMMAND.PUB.SYS file - over the years this file has probably become cluttered with UDC's from accounts that have been purged etc. Use the program UDCWHO.PUB.TELESUP to list all the UDC's that are recorded in the COMMAND file, then build the accounts/users and reset the UDC's. This will give a clean COMMAND.PUB.SYS file (at least to start with) on your new HP3000. One area that DIRMIG does not work, is in the moving of any groups you may have set up in the SYS or TELESUP accounts, therefore you will have to restore these separately from the main migration.

The next stage is to recompile into Native Mode, those programs for which you have source code. If you have third party products that are called as intrinsics, then ensure that the suppliers have provided you with Switch Stubs, that enable your Native Mode applications to call Compatibility Mode intrinsics (if they have not yet produced them you can produce the Pascal code yourself using the program SWAT).

The programs which are written in SPL (or any other language that is not available in Native Mode) and third party products that you do not have the source code for, should be run through the Object Code Translator (OCT). OCT adds to the end of the program code that would normally be produced at run time by the CM Emulator. For a program, this reduces considerably the time taken to load the program. Any program that has been run through OCT or compiled using the compatibility mode compilers, will still run on a 'classic' HP3000 - so long as you ensure that the program occupies one disc extent on the 'classic' HP3000 (the system does not automatically ensure this).

Compatibility Mode is clearly as good as Hewlett Packard suggest it is going to be, but how does it perform ? Our tests indicate that CM on an HP3000 Series 930 is very similar in performance to an HP3000 Series 70, with the HP3000 Series 950 running about 1.6 times faster. If object code is run through OCT then the CM program runs about 15% faster than before it was enhanced.

The area of real improvements are seen when code is run through a Native Mode compiler and where possible optimized to level 2 (Optimization level 2, at time of writing, does not work for Cobol). If performance is the be all and end all of life, then a number of small programming changes can be made to increase performance greatly. For example change all data definitions from 16 bit integers to 32 bit integers, and ensuring those 16 bit integers there are, are word aligned.

To give an example of what can be achieved on an HP3000/930, by making a number of small programming changes, a program was written in Fortran and Cobol to complete an iterative loop 4 million times. The performance figures are as follows :-

Test Description	Run Time in Milliseconds Fortran	 Cobol
Compatibility Mode	96,198	187,050
Compatibility Mode, with OCT	20,059	80,120
Native Mode, no code changes	9,034	12,537
Native Mode, Level 1 (Cobol)		11,535
Level 2 (Fortran)	3,020	
Native Mode, 32 bit aligned	6,031	11,037
with in line performs		8,034
Native Mode, 32 bit aligned		
level 2 optimization	1,010	

As can be seen 32 bit alignment is a slight benefit to Cobol, but of enormous benefit to Fortran (and Pascal).

6.2 Precautionary Step

At all times ensure that you maintain a copy of object code and job streams that will run on a 'classic' HP3000. Until such time that you are sure that you will never have to attempt a backwards migration, ensure that you maintain groups for true Compatibility Mode code and for job streams that do not include any of the MPE/XL extensions. While this is a management overhead, it is well worth the trouble.

Disaster Recovery must always be at the back of your mind, since no disaster service currently offers an HP3000 Series 900 as a backup. So if you have a disaster plan based on an HP3000 Series 70, remember that you must keep compatible programs, job streams and data. Only when you can be sure that you will never again need to recover onto an existing 'classic' HP3000 can you finally and fully migrate.

While looking at precautionary steps, remember that the format of store tapes has changed. If you are going to produce tapes for use on a 'classic' HP3000, then you must use the TRANSPORT option on the store command. The set back in using the 'transport' option is that you cannot make use of a number of the new features of the MPE/XL store command such as FULLSTORE (which makes partial stores, less easy), DIRECTORY (which stores the directory elements needed for the equivalent of an accounts reload) or STORESET, used when storing to multiple tapes simultaneously. Once you are sure that you need never move back to a 'classic' HP3000, then you can stop using the TRANSPORT option and make use of all the new features. This in turn will increase the time your system is available to users, by dramatically reducing store times.

6.3 Database Logging Migration

If you have database transaction logging in place, you may well have to reconsider your recovery strategy. Currently if you use a private volume for rollback recovery, this should not be used on an HP3000 Series 900, since the rollback recovery transaction log files has to be in the same disc class domain as the database it is logging (because of the file recovery system, implemented under MPE/XL).

This gives you a number of options :

- Continue to use rollback recovery, but risk the fact that your log file could be lost - in a disc failure with your database.
- Use rollback recovery, but log to tape
- Continue using a private volume, but use roll forward recovery.

We chose to use roll forward recovery, since this maintained the integrity of keeping the transaction log file on a private volume. It had the added benefit of the use of autodefer at all times, with the performance benefits that autodefer implies.

6.4 Job Stream Migration

Linked with program migration is job stream migration. Jobs that complete stores, sysdump's etc have to be changed to reflect the changes in the store systems. Equally jobs that stream other jobs, just to switch logon groups can be changed to make use of the CHGROUP command.

Since it could well be that you need streams to run both on MPE/V and MPE/XL system, you may well not wish to make changes beyond those required by MPE/XL.

6.5 House Keeping Activities

Since FULLBACKUP and PARTBACKUP are not part of MPE/XL and there are numerous other changes to the SYSDUMP type functions :-

- You must very carefully document your revised house keeping instructions.
- You must ensure that cold boot tapes are regularly completed (using SYSGEN).
- That the accounting directory is regularly stored (using the DIRECTORY option on the MPE/XL store command).

I would recommend that any store that is not being prepared with the TRANSPORT mode option, is always prepared with the DIRECTORY option.

If you have more than one tape drive, and do not need to produce transport mode tapes at all times; the MPE/XL store command will allow concurrent backup across a number of tape drives. This would allow very fast backup's of the system and increase the time that your system is available to your users. Since transport mode tapes take much longer to produce, only cut them when it is absolutely necessary.

7. Conclusions

Migration is a complex process and should be planned, in conjunction with your migration application engineer. If it planned meticulously it should be relatively easy to achieve a successful migration, so long as you remember what I consider the three golden rules in migration :-

- any SPL routines that you are going to rewrite, say into Pascal, should be rewritten and tested on an existing HP3000.
- when you come to migrate, migrate initially using 16 bit alignment (as used on the 'classic' HP3000), then once you are sure that the application is fully migrated, and only then consider recoding to utilise 32 bit alignment.
- never consider making program changes, as part of the migration process. Make any changes on an classic HP3000, fully test the changes, and then migrate.

Expect there to be problems, so leave yourself plenty of time for the unexpected. So long as you test all your job streams, as well as the programs, there should be no insurmountable problems.

Acknowledgements

I would like to record my thanks to the following, who have been of invaluable assistance during the Migration Project at Mecca Leisure :-

Nick Blaney	*Coggon Computers, Sheffield*
Chris Cotton	*Hewlett Packard, Cheadle*
Howard Gillman	*Hewlett Packard, Redhill*
David Guy	*Hewlett Packard, Redhill*
Chris Haird	*Hewlett Packard, Redhill*
Jim Harrington	*TJ Systems, Beaconsfield*
Thomas Lacker	*Hewlett Packard, Boeblingen*
Andrew Popplewell	*Hewlett Packard, London*
David Poskett	*Hewlett Packard, London*

DON'T LET YOUR PROGRAMMER GROW UP TO WRITE OPERATIONAL DOCUMENTATION --- OR SHOULD YOU?

J. B. Watterson
ORI/CALCULON Corporation
P.O. Box 270
Germantown, MD 20874

SOME FOOD FOR THOUGHT

How many of you actually enjoy writing and maintaining documentation? I thought so. At the risk of incurring the reaction "Oh not again!", I submit that you can actually enjoy developing online system operational documentation. Although there are no magic remedies for documentation problems, there are some tools and practices to assist us in creating good, useful documents. Creating good documentation can be easy and it can be fun! The best way to accomplish this is to make documentation development an integral part of the programming and maintenance process.

After all, you are accustomed to working at your terminal. You actually enjoy programming, and aren't afraid of the computer hardware or software. With some guidelines, techniques, and encouragement, you can document. And, you might even enjoy doing it.

Let's first look at some observations.

- Most of us don't like to write! Since we don't enjoy writing, we often don't create the documents we need. And when we do create them, they are often unreadable. Unreadable documents are unusable documents.

- Most of us would agree that an application is only as good as its documentation. Yet documentation is typically inadequate, and is a neglected by-product of the software development process.

- Poor documentation adds costs. User training costs increase because it takes longer to bring the user up to speed. Maintenance costs are higher because program applications are not documented properly - internally or externally.

- Good documentation requires some extra effort on the part of each of us. If provided with guidelines and time, documentation can become a fun and challenging part of your job. You will accept and enjoy your documentation responsibilities. The result -- good documentation.

So, let's take a brief look at how we do it today:

- Our systems documentation generally consists of a System Reference Manual and a System Maintenance Directory.

- Traditionally, operational documents are created as paper documents, maintained by support programmers or technical writers, and distributed by a technical reference librarian.

OUR APPROACH TO OPERATIONAL SYSTEMS DOCUMENTATION GENERALLY CONSISTS OF A SYSTEM REFERENCE MANUAL FOR USERS, AND A SYSTEM MAINTENANCE DIRECTORY FOR SUPPORT PROGRAMMERS.

We have taken the general FIPS PUB Federal government standards and designed customized operational documentation guidelines tailored to our environment. These guidelines provide for consistency without rigidity. Our documentation then reflects the needs of the users of a wide variety of systems.

Our structural approach to operational systems documentation generally consists of:

- System Reference Manual for users.

- System Maintenance Directory for support programmers.

The System Reference (User) Manual describes interactive terminal operating procedures and software/operational capabilities for user organizations. We use non-ADP terminology. This manual is the primary user document, serving both first-time and experienced users.

The manual provides complete instructions for use of the system by the users. It is used as the primary reference during user training. It is the standard reference to which users turn during production operation of the system. This document is structured in such a way that its effective use does not depend on prior data processing experience on the part of the users.

The scope of the manual includes initiation of system operations, data entry and correction, production of reports, and system recovery procedures. A hardcopy manual complements the systems online menus, screens, and prompts.

Quick Reference Cards (QRCs) are also provided, as needed, for quick and easy assistance to the users. They provide easy access to information that users need often while they are online, such as codes or special commands. The QRCs are printed on bright, heavy card stock, in a "Z" fold fashion. The cards are easily maintainable and can be distributed with inexpensive plastic cases which users can attach to their terminals.

The System Maintenance Directory is designed to acquaint the maintenance programmers with the system processes, system programs, and files. It serves primarily as a road map to the entire online system. It identifies the components of the system and their interrelationships. The directory generally includes an introduction, system overview, processes, program names and files. Files include data base schemas and data sets. Supplementary information, as required, is included as an appendix.

Unlike the Program Maintenance Manuals of the past, we no longer include flow charts, program descriptions, or a detailed logic flow for each program. Information of related standard software packages that can be obtained easily from other sources are also not included.

Program-level documentation for each program is part of the source code and is updated whenever a program is modified.

The operational documentation, hardcopy or online, should be easy to use, and simple to maintain. Yet, it must provide the necessary information for the users and support personnel to run and maintain the system.

> TRADITIONALLY OPERATIONS DOCUMENTS ARE CREATED AS PAPER DOCUMENTS, MAINTAINED BY SUPPORT PROGRAMMERS OR TECHNICAL WRITERS, AND DISTRIBUTED BY A TECHNICAL REFERENCE LIBRARIAN.

For years, system user reference and program maintenance documentation have been created as paper documents during the operational documentation phase of the systems development life cycle. The programmer scribbles these documents (reluctantly when they find the time). They are typed, printed and distributed by the hundreds by the technical reference librarian.

Then the poor maintenance programmer has to try and update the documents (again reluctantly) and the cycle continues. If you are lucky enough to have a technical writer, you can be assured the grammar and spelling are correct and the i's are dotted. And don't forget those distribution lists have to be updated -- or are they?

Current documentation development practices are expensive, difficult to manage and maintain, and often lead to products of inconsistent quality. I'll ask my question again -- how many of you actually enjoy writing and maintaining hardcopy documentation? I think you are getting my message.

My history lesson is over. Let's talk about some tools and practices you can use to enjoy developing good documentation.

Here's how we can revolutionize our documentation procedures:

- o We need to use our computers as information gathering and distribution tools. Drop the paper and pencils.

- Write like you talk. Reach the point where you can write in a conversational style. Use words that are easy to understand and avoid jargon. Write as if the reader is:

  - Unprepared.

  - Insecure.

  - Confused.

  - Incompetent.

- We need a commitment to user-friendly communications: online documentation.

These are the areas we are going to discuss from here on in this presentation. First, this isn't a 1, 2, 3 "how to" presentation, but one to give you some ideas and make you think about them. Take these ideas back to your own organization and implement the ones that will work in your environment.

WE NEED TO USE OUR COMPUTERS AS INFORMATION GATHERING AND DISTRIBUTION TOOLS

For years, we have been providing the users with systems to gather, edit, manipulate, store and report information. Documentation is information. So why haven't we used these system capabilities? One obvious answer -- no shoes for the shoemaker's kids. We need to use our computers as information gathering and distribution tools for operational documentation.

We have our editors, HP Text Data Processor (TDP), word processing, graphics packages, and now desktop publishing. We have the computers, albeit mainframe, HPs and the PCS. And, we have communication packages like HPs Distributed Systems (DS). So let's start using these tools to develop our operational documentation. We still need some hardcopy documentation, but we need lots more online documentation. So start using some of your computer hardware and software to your advantage.

Why?

1. You will love it. This is your natural habitat. You feel warm and cuddly with it, and can develop documentation as you program. If you use a word processor, it probably features a spelling check, perhaps even editing features. Some shops have put together some menu interfaces to create documentation skeletons you can fill out at your terminals. And, there are commercial packages available to assist with documentation. By using some of the methods I'm about to discuss, you can easily write like you talk and create much of your documentation online.

And wouldn't it be easier for you if the inventories of your application programs were stored as online data sets? They would be easier to update than having them printed in a hardcopy System Maintenance Directory.

2. Documentation is easy to maintain when it's online. Whether updated by a programmer, a technical writer, or a word processing person; modifications can be made easier and quicker. Editor bars can be inserted, date of modification indicated and new pages added. Even a revised table of contents can be generated automatically, if needed. Desktop publishing puts the means for producing quality hardcopy operational documentation in the hands of the developers.

3. Most users like online documentation. For one of our new major systems, we developed both online documentation and a paper System Reference Manual (SRM). We found that the paper document was used infrequently. We have about 80 active users on this system, but only a dozen hardcopy SRMs are now available. Our users prefer using the online documentation and the Quick Reference Cards (QRCs). For other systems, we have database managers updating access tables and error message tables. The maintenance programmer no longer has to perform these clerical chores.

For one of our large, complicated budget systems, we developed five QRCs. The first card contains general information, i.e., title, system modules, points of contact, logon/logoff procedures, and command conventions. The other four cards represent each of the system modules. With these QRCs and online documentation, only a few of the users needed a hardcopy System Reference Manual.

4. If we develop documentation using the computer, we can use the computer to distribute technical documents. For example, our Departmental Accounting System, operating on the Hewlett-Packard, provides a method for storing all operational systems documentation as files, and transferring text from/to the host HP to the word processing equipment. The modified HP files are transmitted electronically to the ten accounting sites for internal printing and distribution.

Someday we hope to have online user documentation for this system, but it is always easier to develop it for new systems than to go into an existing system. Documentation experts estimate that it costs from 4 to 10 times as much to document software after it is written than to do it while a program is being developed.

5. Finally, there has to be a cost savings associated in using our computers as information gathering and distributing documentation. The savings are generated by providing online documentation at time of programming and eliminating much of the hardcopy printing and distribution. When these documents are updated online by the maintenance programmer and released to the production system, they are immediately available to the user. A major savings in costs and other resources.

It is virtually impossible to determine the price that we pay for faulty documentation. We may know what it costs to document something properly, but what does it cost to not document? Consider, for example, the costs of training and maintenance for a poorly documented system. It takes users longer to come up to speed, and how much longer does it take a maintenance programmer to fix a bug? Who knows?

I had one programmer develop all of his documentation on the HP3000 using TDP and an HP100 lap-top PC. Then he devised a way to transfer it to an IBM PC, which was connected to our NBI word processing machine. The word processing folks did some reformatting, clean up, and spelling checks. They printed a beautiful draft copy for review and modifications.

Others have done something similar using the IBM mainframe and word processing packages on a PC and transferred the document to our centralized word processor. My point is they enjoyed it. It worked, and the company saved money.

Something that can't be measured in dollars is the online documentation psychology. The programmers and users like it. Management is pleased. Good documentation is available and not a forgotten entity. We were electronically sending documentation for one system, but realized that no one was copying and distributing it. To our embarrassment, our own programmers hadn't even gotten a copy of the revisions. The online documentation psychology makes for high morale, updated information and cost savings. Try it -- you'll like it!

WRITE LIKE YOU TALK

We write it as if we are trying to write. And we shouldn't. We should write it as if we are speaking. We ought to talk it. Because in talking it, we automatically put some emotion into what we are saying. We eliminate the technical, archaic, flowery, meaningless, tongue-twisting wordiness. This comes with getting our pencil or terminal into gear before our brains have thought it all out. We get diarrhea of the fingers.

Now let's discuss some reader-based writing techniques. These will make it easy fcr you to develop good documentation -- and even enjoy it.

Have you ever done much writing? Probably not. Few of us ever feel an itch to write. The rest - 99% or more of us - were born non-writers. Writing has always been an unpleasant chore - something like a visit to the dentist or a car dealership! Most everyone has to do a certain amount of writing in their job.

Why don't we enjoy writing? The reasons are numerous. But one common reason is we are afraid our grammar, punctuation, usage and spelling are not up to snuff. Therefore, we are afraid to write -- we hate writing. But you don't need more grammar, punctuation or usage (spelling maybe) to develop good documentation.

Most of us need a basic change in attitude. Let's look at a new and different way of approaching the whole problem. Again, forget the correct grammar, good English, and mistakes to avoid. Reach the point where you can "talk on the computer" or paper, if you insist. Write like you talk!

We've already established the fact that you are comfortable using terminals and PCs. So why not continue to use these tools to develop documents? If you develop online documentation, you must use these tools. If you still need hardcopy documents and online narrative, establish guidelines and use the computer to write. Sorry, I meant to say "write like you talk."

The trick is to remind yourself to write in a conversational tone, and in informal surroundings. Try to imagine yourself talking to one reader in person. After all, your reader is alive, hopefully well, and likes talking in the everyday conversational mode. All the ones we work with certainly do.

When most people read, they have a voice inside them speak to them. Write assuming that your reader will read your documentation the same way. How many of you enjoy talking? (If necessary, be honest). Writing can be just as easy and fun as talking. Write as you talk. You'll be surprised at the quality of the finished document if you use a conversational rather than a literary style.

You needn't impress a user or maintenance programmer with big words, technical jargon, and a barnyard full of bull. Chances are they won't be impressed. So, write like you talk. Go over what you've written and then listen to it. If it doesn't sound like talk, change it.

Rudolf Flesch has written an excellent, short, easy-to-read book on "How To Say What You Mean in Plain English". In his book, he describes seven ground rules (specific style devices) that will make your writing look and sound like spoken English. By the way, these rules may apply to all writing, not just systems documentation.

I've modified these rules and included a few of my own which will assist you in developing documentation the easy way. Here they are:

1. Use active voice.

2. Use simple whole declarative sentences.

3. Use short words.

4. Use short sentences.

5. Limit paragraphs to only two to six sentences.

6. Use contractions like it's or doesn't.

7. Use direct questions.

8. Use the pronoun "you" as much as possible.

9. Use bullets.

Are you starting to get the drift of what I'm talking about? Let's look at some of the rules in a little more detail.

Use Active Voice

The active voice expresses action. It is something which the doer does. This example is from an ad in one of the Washington papers.

> "I will listen to you for $5.00 an hour. Call :::-::::."

Here's one from one of our user documents. "Function keys 5 (left) and 6 (right) scroll the columns to the left and right for you to view."

Be specific - eliminate all generalities.

> "The world is round."

See how the active voice "I will listen" and "for you to view" are action packed statements. No generalities in any of the above statements. Easy for you to write. Easy for the reader to understand. You've got to accent the conversational, eliminate the jargon, and don't mess with Mr.. In-between.

Use Simple Whole Declarative Sentences

Remember the old saying, "Tell it like it is?" The sentence declares itself. So easy to write, yet so powerful. Use it regularly.

You might remember back in the 60s when President Kennedy made a famous speech and talked about development of the space program. He said something like this. "Before this decade is out, we shall put a man on the moon." This is simple, but short. Everyone knew exactly what he said. A recent cartoon in the Washington Post updated this quote like this, "Within this decade we will send a man to the Soviet space center and bring him back."

Here's how a famous ad could have been written. "The man who is in possession of one should be asked." Fortunately, it was put in both the active voice and in a declarative sentence or it wouldn't have become famous. "Ask the man who owns one."

So tell it like it is! Again easy to write -- easy to understand. If you only knew this kind of writing is acceptable, wouldn't you use it? Sure you would.

Use Short Words

So why do people use long words in writing? There are many reasons, but who cares? Just don't use them. Long words and technical words come between the writer and the reader. If you want your reader to understand you, then use short words. Besides, you probably won't have to worry about spelling a short word. The user won't have to look it up in a dictionary.

Here's some do and don't use words for system documentation:

DO USE	DON'T USE
yes	affirmative
no	negative
help	assist
follow	comply
send	forward
ask for	request
end or stop	terminate

Use simple, short words whenever you can. Do you think you can argue with this technique? Probably not.

Use Short Sentences

The average person can only read so many words at a time before their eyes give up and take a short rest. Newspaper writers and editors try to keep the number of words in a sentence down to plus or minus 20. It's so much simpler to construct a short sentence than a long complicated one with semicolons, colons, parentheses, and etc....

All you need to do is make short sentences with an occasional comma and a period or question mark. You will love it and so will your user. Not much grammar to worry about. Mostly a subject and a verb. And yes, it is possible to put a preposition at the end. We use prepositions at the end when we talk. No one has had to go directly to jail - do not pass go - do not collect $200 for putting a preposition at the end.

Here's a good example of a poorly written long sentence from a real life document I received. A total of 46 words!

"In consideration of this implication, our first and fundamental recommendation concerning the future HQs supplied system (software) end user documentation is that it and the locally developed and maintained accounting system documentation each acknowledge and compliment the information provided by the other, both conceptually, and logistically."

Why not just write - "Our first recommendation is ::::::::::::::."? Shortening your sentences is one of the best and easiest ways to improve your writing style. Surely you can use the short sentence technique.

Keep Your Paragraphs Short

If using short words and short sentences make sense, why doesn't the same logic apply to paragraphs? It does.

Here's an excerpt from a document I received. A good example of how not to write a paragraph with only one sentence. By the way, this has 52 words in it.

"In other words, are sufficient Human and material resources available (commitment) to prepare and/or update the Documentation at the same time that particulars of its subject change, and does the method of communication of the Documentation material (logistics) support rapid and reliable dissemination to the targeted End Users of the Documentation."

Now don't ask me what that paragraph says. Because, I don't know. You understand what I am saying. Keep your words short. Keep your sentences short. Keep your paragraphs short. Easy to write. Easy to read. Any of us can write this way.

Use Contractions

There are fifty or so contractions that are commonly used in writing. I don't want to take the time to list them all. A few examples are: I'm, you're, he's, she'll, aren't, haven't, won't, don't, and let's. I've used a plenty of them in my presentation.

Don't get weird and invent your own like it'd and this'll. Stay with the accepted ones.

So, use contractions because that's what we use when we talk. You probably use them too.

Use Direct Questions

There's nothing like a direct question. Use them freely, especially in preparing online documentation. Do you want to continue? Do you want to edit your input? Do you need help? All the user has to do is type in a Y or N. Keep the questions short and direct. Let the user answer as simply as possible. There's nothing like a direct question to get some feedback to what you are saying.

Use Pronouns As Much As Possible

When you're talking to people, don't you use I, me, you, we and they? Sure you do. Why not use pronouns when you are documenting? I love the word "You" when you are explaining something to the user or maintenance programmer. Consider the following examples:

- o You do this.
- o You enter.
- o To help you.

How about using this techniques as it makes documentation personal and warm. You are talking directly to one person. Only one person - you.

See how easy it is for you to talk as you write?

Use Bullets

Although you don't talk using bullets, I like to use this technique in documenting for the following reasons:

- o It makes writing easier.
    - - Phrases, statements or sentences may be used.
    - - Little or no punctuation is used.
    - - Statements can be broken down to at least four levels.
    - - Not much grammar to worry about as you don't even need to write complete sentences.
    - - No numbering scheme is necessary; e.g., 1.a, 1.b, etc.....
- o It makes reading easier.
    - - The facts aren't all jammed together in a paragraph.
    - - It's easy to make notes or references to a line item.
    - - You can easily concentrate on one line at a time.

See how clean this looks and each line jumps out at the reader. If you need to go to lower levels, here's how we break it down:

o level one

- level two

oo level three

-- level four

This technique is easy for you to use, and it makes your documentation easy for the reader to read.

ONLINE DOCUMENTATION

We need a commitment to user-friendly communications: online documentation.

So much for tradition. Let's make it easy for us and our users. Let's start developing online operational documentation for the mainframes, the HPs and the microcomputers (PC) systems. The PC folks have been doing this for years. Why not strive for a paperless society?

Did you notice the statements on the income tax instructions about form reduction? What a laugh! And, their instructions are something else, aren't they? The instructions are user-nasty. The IRS hasn't really helped us this year. To illustrate, here's a cartoon from the Washington Post:

"Dear Tax Accountants:

I'd like to get your help in figuring out the new tax instructions and this year's tax forms. They've really got me stumped.

Sincerely, John Q. Jones IRS Agent Treasury Dept.... Washington, D.C."

Well, let's not be like the IRS. Go for online documentation. You can still have some hardcopy documentation or Quick Reference Cards available. Or even tell your users they can print a screen if they need to for a paper reference. Isn't this much easier than looking something up in a three-inch documentation manual? Sure it is.

We've established that tradition is out, and you can write like you talk. And, you can develop documentation in your own environment using the skills and tools you already have. Now we'll talk about some methods to provide online text. This subject could be a technical discussion all by itself, so I will only touch on it lightly.

Tutorials

You should provide the user with tutorial assistance screens. This provides first-time users and maintenance programmers with the basic information on "how to use the system". This is also a valuable tool for initial and ongoing training seminars. Tutorials are also good for describing screens and error messages. Remember to write like you talk.

Screen Design

Very briefly, design the screens to help the user. Design them to create a paper document if necessary. Provide user guidance and prompting assistance. Highlighted movable lightbar (block cursor) prompting is great. If you have the luxury of color monitors, take advantage of using color. Highlight errors and provide a brief description at the bottom of the screen.

Online menu screens provide prompts and help. They should provide the ability for the user to request more detailed online help. Develop screen guidelines to use for screen consistency.

Help Prompts

Context sensitive help is provided through alternate PF key selection. Offer help prompts on all level 1 screens. Briefly define the prompt at the bottom of the screen; for example, 1=help, 2=next, 3=previous, 4=create, 5=report, and 12=quit. Try and keep the PF keys consistent within a system whenever possible. For more user assistance, help overlay or level 2 screens may be initiated by pressing the PF keys designated on the screens.

System Maintenance Directory

The hardcopy segment of this programmer's document is truly a directory. It serves as a road map to the entire online system. Why shouldn't this document be online for the maintenance programmer? Sure it can. Store your inventory of programs and files online for easy updating by the programmers. All narrative can also be online for easy browsing and maintenance.

Internal Program Documentation

Every program developed must have comments within the program to document it. If you aren't doing this - please start immediately!

At the beginning of each program, there should be a description of the function performed by the program. Include all files used the by program, data sets accessed or modified, and all entry points. Also, all sections that perform major functions or contain complicated code should be explained by comments throughout the program.

A log of all changes to a program should be kept at the beginning of the program for future reference. This log must include the date of modification, change or problem request number, initials of programmer making change, and the reason for change.

The use of new 4GL languages heightens the importance of internal documentation. COBOL has been with us for a long time. Most programmers are experienced with it and consider COBOL reasonably straightforward and easy to follow. C language, for instance, is full of peculiarities and may be hard to follow if it isn't carefully documented.

For all of our mainframe CICS applications, we have developed standard front-end programs and screens. These include a system entry program with administrative routines; for example, date conversion. Other examples include a message file, a system owner file, and a tutorial broadcast file. Each of these are defined online for the programmer's use. The programmer can then concentrate on the specifics for the application system. We even have a hotline report to indicate user problems. The report contains such information as abends, error message numbers, and user terminal. The hotline programmer can give quick response in helping the user.

In summary, online text is more easily modified and maintained than paper documents, and may be used to create paper documents if needed. Online screen displays are easier to illustrate, and the text can provide the user with online assistance and reference capabilities. Tutorials lead first-time users through the basic job tasks by means of a series of exercises. And, you can develop this documentation as you program.

CONCLUSION

This technical session has provided you with some practical guidelines. I trust it has stimulated an awareness of how you can provide quality, easy-to-use operational system documentation.

Documentation methods are giving us new power to improve the state of our documentation. These techniques and tools will help to make it easier for us to create better documents, and of course provide better products - but, only when our attitude towards documentation changes. This includes us, our staffs, and our users.

I would like to challenge each of you to continue to develop innovative methods for producing quality documentation. Use methods that integrate online and hardcopy media in a cost-effective manner throughout your systems life cycles.

And even more important, from a technical perspective, why you can grow up to write online operational documentation. And, you can enjoy it!

The Use and Abuse of Non-hashing Keys in IMAGE

Fred White
Adager
Apartado 248
Antigua
Guatemala

BACKGROUND

The first set of specifications for IMAGE/3000 proposed that all master dataset primary address assignment be accomplished by hashing the key of each master entry and reducing the result modulo the capacity of the master dataset.

In late November of 1971, some reviewers suggested that application designers might want the application to have control of this primary address assignment and requested that we modify our specifications to provide such a capability.

Our initial response was to require the key field of such masters to be a 16-bit (or 32-bit) integer whose value would be treated as invalid if it was less than 1 or greater than the capacity.

If we had left it at that, we would have provided the user with master datasets using this simple direct access method.

This was aesthetically displeasing in that it restricted both the length of keys and the values of keys.

We eliminated the former restriction by allowing any key length while using only the low order 31 bits as input in calculating the primary address.

We eliminated the latter restriction by reducing this 31-bit value modulo the capacity N (with a zero result mapping into N).

The user also had to have some method of specifying to IMAGE, via the database schema, which type of primary address calculation to apply for each master dataset.

It seemed only natural that IMAGE should apply hashing to keys of data types U and X and that the "integer" data types I, J and K were perfect for use in the generalized direct access method of primary address assignment. It was less obvious which method to employ with data types P, Z and R.

Then one of us noticed that the HP 3000 internal representations of the mantissas for IMAGE data types I, J, K and R were all in binary format and that P and Z were not. On this rather weak basis we decided that all keys of non-binary format would employ hashing and that the others would not.

Thus, the hashing/no-hashing decision is implicit in the data type of the designated search field with hashing employed if the the data type is U, X, P, or Z and non-hashing for data types I, J, K and L.

SYNONYMS

Two or more key values are said to be synonyms if they are assigned the same primary address.

Whenever a new entry is assigned a primary address which matches that of one or more existing entries, IMAGE locates it at an alternate address close by its synonyms.

It attempts to place it in the block containing the entry at the primary address. If this block is filled, IMAGE serially searches subsequent blocks until it finds an unoccupied location to place the new entry.

IMAGE links all synonyms for a given primary address together into what is known as a synonym chain.

If there is no severe clustering (see below) and if the dataset is not almost full and if the blocking factor is large relative to the average chain length, most synonym chains will reside in a single disc block and thus have little impact on performance since they can all be made present in memory with a single disc read.

For hashing keys, the average synonym chain length can be kept small by:

1. using keys at least 10 (preferrably 12) characters long. (IMAGE's hashing algorithm does a better job with long keys than with short ones.)
2. using key values which are not excessively uniform in their content
3. using capacities 15 to 30 percent larger than the expected number of entries

Also, their negative impact on performance can be reduced by:

1. making the blocking factor large relative to the average synonym chain length.
2. not allowing the dataset to become nearly full.

For heavily accessed masters, try to have a blocking factor of at least 6 or 8. If your blocking factor is less than 6, consider replacing the manual master with an automatic master and a related detail containing the data portion of the original manual master.

CLUSTERING

It is entirely possible, particularly for non-hashing keys, that many of the records of a dataset will fill contiguous blocks in one or more portions of a dataset while other portions are empty, or nearly so. Such a phenomenom is referred to as clustering.

Clustering is harmless as long as there are no synonyms. Otherwise, clustering is typically dangerous, as we shall see.

THE USE OF NON-HASHING KEYS

An excellent use of the direct access method provided by non-hashing keys would arise if the key, for example, were DAY-OF-YEAR.

In this situation, the master dataset would only need a capacity of 366 (any higher would be wasted disc space).

The data item DAY-OF-YEAR could be defined as type I and the key values would be the positive integers 1, 2, . . ., 366.

As long as the application prevented other key values from occurring, no synonyms would ever arise, there would be no waste space, and IMAGE performance would be optimal.

Note also that the record with key value of 1 would be record number 1, the one with key value of 2 would be record number 2, and so forth. This "natural" ordering might be of some advantage to your application.

You may find other such situations, perhaps involving badge numbers, building numbers, or whatever, where you might want to employ integer keys (i.e., data types I1 or I2) in this manner. Usually, however, this will involve some wasted disc space. Only you can decide if the wasted space is too exorbitant for the benefits offered.

THE ABUSE OF NON-HASHING KEYS

1. The Clustering Pitfall

My first live encounter with a misuse of integer keys arose in 1978.

One Friday in 1978 I received a phone call from an insurance firm in the San Francisco Bay Area. I was told that their claims application was having serious performance problems and that, in an attempt to improve the situation, they had, on the previous Friday, performed a DBUNLOAD, changed some capacities and then started a DBLOAD which did not conclude until the early hours of Tuesday morning!

They were a $100,000,000-plus company which couldn't stand the on-line response they were getting and couldn't afford losing another Monday in another vain attempt to resolve their problems.

Investigation revealed that claims information was stored in two detail datasets with paths to a shared automatic master. The search fields for these three datasets was a double integer key whose values were all of the form YYNNNNN (shown in decimal) where YY was the two-digit repesentation of the year (beginning with 71) and where each year NNNNN took on the values 00001, 00002, etc. up to 30,000.

Although the application was built on IMAGE in late 1976, the earlier claims information (from 1971 thru 1976) was loaded to be available for current access. I do not recall the exact capacity of the master dataset but, for purposes of displaying the nature of the problem (especially the fact that it didn't surface until 1978) I will assume a capacity of 370,000.

Although the number of claims per year varied the illustration will also assume that each year had 30,000.

The first claim of 1971 was claim number 7100001 to which, using a capacity of 370,000, IMAGE would assign a primary address of 70,001. This is because 7,100,001 is congruent to 70,001 modulo 370,000.

The 30,000 claims of 1971 were thus assigned the successive addresses 70,001 through 100,000.

Similar calculations show that the claims for each year were stored in clusters of successive addresses as follows:

Year	Claim Numbers	Assigned addresses
1971	7100001-7125000	70,001-100,000
1972	7200001-7230000	170,001-200,000
1973	7300001-7330000	270,001-300,000
1974	7400001-7430000	1-30,000
1975	7500001-7530000	100,001-130,000
1976	7600001-7630000	200,001-230,000
1977	7700001-7730000	300,001-330,000

Note that no two records had the same assigned address and thus that there were no synonyms and that all DBPUTs, DBFINDs and keyed DBGETs were very fast indeed!

Along came 1978!!!

Unfortunately 7,800,001 is congruent to 70,001 so that the first DBPUT for 1978 creates the very first synonym of the dataset. It is, in fact, a synonym of claim 7100001.

DBPUT attempts to place this synonym in the block occupied by claim 7100001 but that block is full so DBPUT performs a serial search of the succeeding blocks to find an unused location. In this case, it searches the next 60,000 records before it finds an unused address at location 130,001! Even with a blocking factor of 50, this required 1200 additional disc reads making each DBPUT approximately 200 times as slow as those of all previous years!!

Note that the next claim of 1978 (with claim number 7800002) is congruent to 70,002 so is a synonym of 7100002 and also leads to a serial search which ends at location 130,002! Thus each successive DBPUT results in a search of 60,000 records 59,999 of which it had inspected during the preceding DBPUT!!

Clustering had claimed another victim!! The designer of this system had unknowingly laid a trap which would snap at a mathematically predictable time, in this case 1978. After struggling with this problem for months, the user escaped the clustering pitfall by converting to "hashed keys" (in both the database and the application modules); a very expensive conversion!

Note that the problem was not a synonym problem in the sense that synonym chains were long nor was it a "fullness" problem since the master dataset was less than 57% full when disaster struck.

The problem was due to the fact that the records were severely clustered when the first synonym occurred and DBPUT's space searching algorithm is efficient only in the absence of severe clustering.

Note that the performance of DBFIND and DBGET was excellent.

A similar, more modest pitfall would have been encountered if, in the above example, the claim numbers had been of the form NNNNNYY with the same capacity of 370000. In this case, the performance of DBPUTs, DBFINDs and keyed DBGETs would all degrade over time but would never reach the disastrous level of the DBPUTs of the example. In this case, the degradation would arise due to the length of synonym chains and due to local clustering.

Note that this modest pitfall can be eliminated simply by changing the capacity, for example, to 370010.

Note however that this problem would still arise if the capacity were merely changed, for example, to 370001.

2. The Synonym Pitfall

An even worse case would arise if the designer elected to use a key whose data type was R4 and whose key values were greater than zero and less than 10 million.

To understand why, one must be knowledgeable about the format of 64-bit reals as represented on the HP 3000 family of computers.

The leading bit is the sign bit, the next 9 bits are the exponent, and the remaining 54 bits are the mantissa excluding the leading bit.

As a consequence, the floating point format of all integers less than 8,388,609 (2**23+1) is such that the low order 31 bits are all zeroes. Therefore ALL entries would be in a single synonym chain having the dataset capacity as its primary address!

In adding a new entry, DBPUT would have to traverse the entire synonym chain to ensure that the key value of the new entry was not a duplicate before adding it to the chain. This would have a negative impact on performance proportional to the number of entries and inversely proportional to the blocking factor.

Also, each DBFIND or mode 7 DBGET would, on average, be forced to traverse half of the chain to locate the desired entry!

I hope that you will NEVER use fields of data type R as key fields.

SUMMARY

Remember that, in electing to use non-hashing keys, the designer has taken the responsibilty for primary address assignment out of the hands of IMAGE and placed it in the hands of the application.

This should be done only if:

- some benefit will be derived by their use
- the application has absolute control over key value assignments
- the values so assigned, together with the assigned dataset capacity, assure the designer that the application will never encounter the clustering or synonym pitfalls

FOOTNOTE

Avid readers of IMAGE articles might be surprised at the absence of any reference to primary numbers as capacities for master data sets.

The reason for this is that I consider any argument for or against their use as, at best, an academic exercise in futility and, at worst, a "red herring." Application designers and database administrators can realize far greater performance improvements by dealing with other, more significant, issues such as those addressed in this paper.

Using a Task Manager to Improve User Productivity

Barry Polhemus
ETC Corp.
284 Raritan Center Pkwy.
Edison, NJ 08818-7808

Introduction.

The HP3000 is reasonably 'programmer friendly' when it comes to application development. Particularly if using TRANSACT/FASRTAN, it is easy in a prototyping shop such as ours for many applications to be developed rapidly as company needs arise. Most of our applications are transaction based and the user environment is highly interactive. The problem is, however, that as the company grows, the number of transactions also grows. As the needs grow, so do the number of applications. Soon, not having CPU horsepower to burn, the HP3000 could not keep up with our system needs. It became necessary to make some adjustments in the way work was being done on the system. There were many steps in our ongoing solution including spreading applications across multiple Series 70 machines. What I will present here is one of the steps taken to make our system (and users) more productive by decreasing interactive user time.

Many of the functions performed in applications can be separated into foreground and background tasks. By background task, I don't mean a batch process which could be handled by a periodic job stream, but rather a function which must be performed in a short time frame. These functions or tasks obviously require user initiation but do not require a user to sit in front of a terminal until the task completes. If the user can put together necessary information to perform a given task in the form of a work request, and simply submit that request to a background process to have the work carried out, the user becomes more productive. The time spent waiting for a CPU intensive task to complete can now be spent doing other work.

Drawbacks of a Simple Solution.

The design of a system to function in this manner could be built around MPE message files, but there are distinct disadvantages to relying exclusively on message files. First, let's look at this simple approach. There will be a submit process which puts together a work request and writes it to a message file. There will also be a background process to read the message file and carry out the requested task. The work request will be lost if the background process reads the request from the message file but is aborted before completing its task. If you have a means of determining what request was lost, you can requeue it, but there may be problems if the application requires the order of requests in the queue to be maintained. To avoid losing requests, we could first do a non-destructive read followed by a destructive read after the request has been processed. This technique will work for a single background process, but not if you want multiple background processes to distribute the load for the same type of work request. There is no way to coordinate which process is working on which request.

Another diffculty will arise when you want to see what requests are pending for the background process. You can't just read the message file (eg. FCOPY) since that will wipe out your work requests. We can use COPY access to look at the message file which does non-destructive reads, but this requires exclusive access to the file. This means the background task which reads the file can't be running at the same time. Then there's the problem of starting and stopping the background process. If it's running in job mode, how do you let it know that you want it to stop without simply aborting it? If you write a request indicating self termination, how do you get it past the other pending requests in the message file? Reordering the requests in the message file would be difficult at best while the process is running. Actually, in addition to a termination request, the ability to reorder requests might be might come in handy if we needed to move a priority request to the front of the queue.

For a single application there could be a single background job to process requests, but when our number of applications increases to ten or twelve, we would get twelve separate jobs being fed by twelve separate message files each with their own set of maintenance difficulties as described above. As you can see, the number of messy details in our simple solution is increasing rapidly. Let's consider an alternate, more practical approach.

The Real Solution.

Even though message files have their drawbacks, they can be useful if used with discretion. Let's say there will be a submit process which will format a work request and write it to a message file. There will also be a background process to perform the intended task, but now it will be a son process running under a father/monitor process in a single job. The monitor process will act as a work request buffer between the submitting process and the process which does the work. It will pick up the incoming requests from the message file and store them internally. Even though there can be many submitting processes, there will be only one receiving process. The delay between writing a request and having it read is now minimal. We don't need to rely on the message file to hold requests until they are processed and we don't have to do any tricks to get the request from the message file while worrying about whether or not the work request will complete successfully.

The monitor will automatically maintain the order of the incoming requests as well as keep track, since there can be multiple son processes, of which requests belong to which son processes. Son processes can each perform a different function and/or they can be duplicate sons used to distribute processing of a given function. There will be a separate queue for each son process and now that we no longer have the message file restrictions, managing the queues is now a much simpler task. We can now list the individual queues, reorder requests within a given queue, and move requests between queues. Of course, the latter may not make sense unless the source and destination queues handle the same type of request.

Queue management also involves starting and stopping individual queues. We need to selectively change a given queue's status without bringing down the whole monitor system. To accomplish these queue management functions, we need to send commands to the monitor from the outside world. Outside here refers to the inter-session environment. MAIL intrinsics suffice for father/son (intra-session) communication, but we need to send commands from an interactive session to a background process running in job mode. So we must go back to our trusty message file only now we don't have to wait for all preceding requests to be processed before our command gets through.

Since input to the monitor process may be an incoming request or some sort of command, we need some minimal structure to the incoming message. Each must contain a command identifier followed by data such as process number, request number, or whatever might be pertinent to that command. Incoming requests will be regarded as commands meaning pick up the request buffer and store it. As for requests, however, a little more flexibility incorporated into request routing will go a long way later. The need is simply to associate an incoming request with a particular son process. Good style suggests that we avoid things like hard-coding destination son process names into our submit programs since if we change our son process name, we also have to change our submit program. Also, in cases where there are multiple son programs to distribute load, we need a mechanism to submit requests such that our submit process doesn't care how many son processes might be available to perform the requested work. Even more important, we don't want to require our users to follow the load and explicitly submit to a given son process in the group.

The 'Chain' Mechanism.

We can achieve the flexibility we desire by using a method which I will call chaining. The mechanism will work as follows. Submit processes will put an identifier called a chain id in the request. There will be an external reference (chain file) which will relate a given chain id (request) to a specific son process number. The monitor will deal with its son processes by number. There is another external reference (configuration file) used by the monitor which contains program file names and provides an association between program file and process number. This way, the chain file as well as submit program are independent of physical file names. As the monitor receives a request, it will look up its chain id and queue it to the appropriate son process. The chain file may indicate a single son process number or a list. In the latter case, the monitor can chose among the list of potential recipient sons based on which are active and how many requests are pending for each that is active. Thus load balancing among active duplicate son processes is automatic. Since we can start/stop specific son processes, we can regulate the throughput for a given request function. For example, if we have three son processes to handle a given request type and all are active, we can have requests distributed automatically between all three. At times when the load is lower, we can shut down (stop) one of the three so that requests will be distributed only among the two active sons. Also, since we can move requests between sons, any pending requests for the process we shut down can be moved to one of the remaining active processes.

Perhaps the most significant feature of the chain mechanism is also the reason for the name 'chain'. In the chain file, along with each chain id there can be a second chain id called the 'next chain id'. When a request is completed, the next chain id, if there is one, will become a chain id on the same request as it goes back to the monitor. This means that requests can be 'chained' from one son process to another automatically using the same physical requests. Thus it is possible for a son process to insert additional data into the request before it gets passed on to the next son. We now have the freedom to modularize background processing in any way we chose. Requests may follow different paths through the same group of son processes. For example, say we have three son processes through which a given request will follow through in sequence. We also have a similar request type which needs only the first and third sons. By simply supplying the appropriate chain id on the original request, we can direct it through either path via the chain file. Suppose we want to insert a new module in a given chain. All we need do is modify the chain sequence in the chain file and set up the new process in the monitor. We don't need to change any of the submit or associated son processes.

Communicating with the Outside World.

So far, I have only discussed the communication of information from the outside world to the monitor system (monitor process and associated sons). We also need a means of getting messages back to the outside world as to what's going on inside. There are two types of information we want to see. First, general information regarding the status of the system such as which son processes are active, how many requests are pending for each, the processing order and content of the pending requests, which requests are currently being processed, etc. Since all of this information can be kept in files dynamically updated by the monitor, all we need is a utility program which understands the structure of these files and can then display status information about the monitor system. We don't want any synchronization problems, so we won't allow the utility program to make changes to these files. Should we want to effect some change to the system, such as shut down a particular son process, the utility program will send a command to the father process and the father will actually make the change. The utility program can then be used to verify that the did actually took place. The change won't necessarily be instantaneous since we're dealing with communication between two separate processes (actually, three in the example of shutting down a son).

The second type of information the monitor must provide are messages from the various son proceses regarding errors encountered while performing intended functions. That is, if the request didn't complete successfully, we want to know that it failed and probably why it failed. If son processes simply write error messages to $STDLIST the entire monitor process would have to be shut to check for errors. A more practical method is to log error messages to a file. This file could then be reviewed without disturbing the monitor process. Since there is usually no need to keep the error messages indefinitely, a circular file is a suitable choice. The simplest way to handle error logging is to have the father process do the actual writing to the logging file. The son processes can send a command containing the error message to the monitor which will then log it to the error file including a time and date stamp and process id. There can be a separate program to read the error log file and display error messages without affecting the monitor system.

Implementation.

Now that I have laid the groundwork for the monitor system, I will give some details of our implementation and describe more of the features of such a system. Though we are basically a TRANSACT shop, the monitor program (named Monitor/3000) was written in SPL for practical reasons. The son processes, as well as the submit processes can be written in any language. I will discuss these later. The monitor functions to link the submit process to the son process(es) which eventually carry out the given task by passing the work request from one to the other. Only the submit and son processes need understand the content of the request. The actual request does include some header information, however, on the request as received by the monitor such as a request indicator to identify it as a request as well as the chain id. The rest of the request is blindly passed along to the appropriate son.

There are several files used by the father process for storing requests, maintaining queuing information, maintaining status information, as well as the chain and configuration files mentioned earlier. The configuration file contains not only the list of program files that represent the various son processes but also contains the file names of the other internal files used by the monitor system. This way, all of this configuration information can be obtained by the monitor program via a generic file equation without hard-coding any file names into the monitor program. We can now set up multiple monitor systems on a given HP3000 by simply setting different file equations. The associated utility program previously mentioned works in the same manner. There is only one physical utility program which gains access to the various monitor systems via a different file equation before running the program.

Mission Control - the Utility Program.

The control of the monitor system really lies in the utility program. It, too, is coded in SPL for practical purposes. By practical, I am referring mainly relative to TRANSACT, given what the program needs to do. The utility program gains access to all of the monitor system files via the configuration file back-referenced through a file equation. The utility program can display the overall state of a given monitor system including number of son processes, the current status of each, and the number of pending requests for each. The requests' content can be explicitly listed in octal and/or ascii formats. A given request can be prioritized by moving it to the front of the queue for its destination son process. As mentioned before, requests can be moved from one son to another (provided they can service the same type of request as per the configuration file). Requests can also simply be purged.

The utility program functions not only to manage requests but also the state of individual son processes. There are three main states in which a son process can be. First, there is Down, which means that son has no associated physical process (no PIN). There is Inactive, which means that son process has been created, but it is not activated. A son cannot process requests in either of these states. Finally, there is Wait which means the son has been activated and is waiting for a request to process. Once there is a pending request, the status will be Active, implying that a request is currently being processed. The contents of that request can be listed even while it is active. There is also a special case when that occurs when a son is started up with requests pending. This Busy status means that the monitor process has sent a request to the son but the son has not yet received it because it is busy going through its start up procedure.

Utility program commands exist to change a given son between the three states. There is CR (create) to go from down to inactive, and SU (start up) to go from inactive to active. These correspond to MPE Intrinsics CREATE and ACTIVATE. There is SD (shut down) which moves a son from wait/active to inactive. The currently processed request, if any, is allowed to complete. During this time while the request is completing, the status will appear as Shut Pending for that son. As it completes, the son is suspended. There is also KI (kill) which removes the son immediately from the system. If there is a request being processed it is left in the queue.

Before continuing, let me digress a bit and discuss queue integrity. The first concern is in not losing requests. The monitor system I am describing meets that requirement. Since it stores the requests internally, the requests remain intact until they are either completed as determined by the monitor itself via notification from the son which processed it, or an explicit purge request is received. Whenever a request is being processed and does not complete normally due to the son process being killed, the entire monitor process being shut down or even aborted, or even a system failure, that request is left in the queue and will be dealt back to the appropriate son when the system is restarted. (Of course, this is under 'semi-normal' circumstances, there isn't much that can be done for disc crashes, for example.) The other concern with queue integrity is maintenance of chronological order of requests. In most applications this is not a necessity but in others it can be vital that requests be processed in the order submitted even if not processed right away. Again with queue integrity in mind, our monitor design includes periodic (whenever a change takes place) posting of queuing infomation to disc. Thus the individual queues remain intact even if the entire monitor system is aborted.

There are a few remaining utility program commands which I will mention. BR (break) can be used to abort processing of the current request and have the son continue with the next request. GO and SH are the commands which start and stop the entire monitor system. There are also some commands not for use by the average user such as those to initialize (IN) all the queues (wiping out all requests for all sons) and a renew (RN) command which rebuilds all the queues from scratch but destroys chronological order. VR (verify) causes the monitor process to do an internal verification of its queuing buffers and lists. These commands are shielded from the typical user by a security code. Another protected command allows the priority of individual sons or the father itself to be changed among CS, DS or linear queues. There are some realistic checks built in here, however. You can't put a son in the linear queue at a priority higher than the father or at a priority higher than the the low end (highest priority) of the CS queue.

The remaining commands are harmless. There is a pair of commands which function like OUT=LP and OUT=TERM commands in QUERY. Output from the listing or status commands can be sent to a file or printer. The ID command will display version id and some configuration parameters. There are some semi-arbitrary limits on the monitor system such as the maximum number of sons that can be included in a given monitor system as well as the maximum number of requests that can be stored internally. In our system, we have set the maximum sons at 15 and the maximum requests at 2000. When a son process aborts (or is shut down), the pending requests will continue to accumulate until it is restarted. If enough requests back up, the monitor will no longer accept requests until room is made available by either purging requests or processing them. In the meantime, requests simply accumulate in the message file.

When the internal request limit has been reached, the monitor periodically displays a message on the console indicating a request overflow has occurred. Display of this message can be turned off or on via a utility command. Once you realize that an overflow has occurred, there is usually no need to continue the console messages. In some applications, 2000 requests may be much too many to take advantage of the overflow warning system. The number of requests that will be accepted by the monitor is configurable up to the maximum allowed, so if 2000 requests constitute a month of work, it is probably better to set the internal limit at more like 200 instead.

The Workers - Son Processes.

Enough said about the utility program. Now I'll discuss some aspects of developing/converting applications to run under the monitor, namely submit programs and son programs. The submit program has a simple task. It only has to collect any information necessary to perform the task in question, put it in a format that will also be understood by the destination son process, and write it to the request message file. The request has some header information (such as the chain id) followed by up to 480 bytes of data. In some of our applications, the request consists of only a file name but in others it may contain a fairly wide IMAGE record right in the request. Since both our submit programs and son programs are written in TRANSACT, it is a common practice to put the request buffer definition in an include file for use by both submit and son programs. This insures that no discrepancies exist in the request contents and tends to make programs more understandable because the same variable names are used.

The son programs need to carry out the intended task based only on information from the request. The physical process of request handling, ie. getting requests from the monitor (father) and letting the monitor know when the task has been completed, is provided in the form of a monitor interface. For our TRANSACT applications, there is a file which is to be included at the beginning of the program which does all of this dirty work. The interface code will do a perform to a predesignated label for each request it receives. It also performs to another label for initial item listing and one for any initialization code. The son program then only needs to include the interface file and provide the appropriate labels and need not be concerned with details of request passing. Once code is developed to perform a function, making it run as a son process is quite simple. This same style has been extended to other languages as well. We have one son application written in SPL and we have done some testing in PASCAL. In these cases, the interface is in the form of procedures placed in an SL or RL such that the main body of the program does nothing more than call the interface procedure. The son program will have one main procedure (other than 'main' body) for processing a request whose calling address is passed to the interface procedure so that it can be called from there. Three other procedures can be included, one for initialization upon startup, one for processing upon restart after a shut down, and one for processing prior to a shut down. The latter of these procedures is only used in case of a normal shut down command as described above. For a normal shut down, the father process tells the son to suspend when it has finished processing its current request, if any, and the son can perform any shut down functions prior to suspending. If the son is killed, no shut down processing can be performed.

The design of a monitor process (submit/son pair) is basically separating the information needed to perform a task from the task itself. A typical application will include both an interactive section to specify what has to be done, and a section to do the work. In cases where the task is simple, creation of a monitor process may not be practical. However, a task which is CPU intensive relative to the interactive front end is a prime candidate. Another situation might be when the tasks are very simple but numerous and can be queued sequentially as a batch. Let me give an example to differentiate. Suppose there is a chain in an IMAGE data set and we need to perform some action for each entry in the chain. The submit program can queue just the key value in the request such that the son reads the chain and processes accordingly. The alternative is to have the submit program read the chain and send requests for each entry. The latter approach involves a lot more request traffic but for some applications this trade off is necessary. In either case, the user is supplying only the key.

One more comment about developing son processes concerns error handling. Different coding styles usually involve different degrees of error handling. Monitor son processes must be made as robust as possible so that if anything goes wrong during processing of a given request, short of major catastrophe such as a corrupt data base, the error will be handled gracefully and the son process will continue with subsequent requests. In the above example where the son process accepts a key value to read in an IMAGE data set, and the key value is invalid, the son process should just log an error message and move on to the next requestrather than abort. Error logging is as simple as putting pertinent information in a buffer and calling a procedure (or perform for TRANSACT) supplied as part the monitor interface. The more son processes abort on their own, the more attention must be paid to son process status and management. It is certainly more desirable for the sons stay running without paying attention to them, especially if there is a high volume of requests being processed. If a high volume son aborts, the incoming requests will back up quicjly and if not discovered soon enough, the backlog may be difficult to recover from. This is an example of why one might configure the total requests held internally by the monitor to less than the maximum so that if a son aborts and the queue backs up, the console warnings give an earlier indication that a problem exists.

Real-Life Examples.

I will now give an example of some of the ways we use our monitor system. We have a chemical analytical lab which generates data on a variety of intruments, predominantly HP1000 based mass spectrometers. Data files containing analysis results are transferred from the HP1000 to the HP3000 and subsequently a request is written to a monitor message file which contains only a file name. A son process receives this request and will load the file into a transitory review data base. Once the file is loaded sucessfully, the file is purged. The loading process looks up some pertinent information from another data base and stuffs it back into the request. The request is then chained to a second son which takes the added information from the request and performs some other cross referencing functions. This processing is all automatic.

The data is then reviewed in the temporary data base with interactive processes. There are batch oriented calculations done by a different son process which are also reviewed subsequently via an interactive process. When review is complete, data organized into groups which represent the final reports, and sent to another pair of data bases which will hold the data indefinitely. The problem is that these data bases are on another HP3000. The solution lies in a another son process that handles generic file transfer via a remote I I session as well as simultaneous (part of the same request) writing of a supplied buffer to a message file on the remote system. The remote message file is, as you might guess, an input request file for a monitor running on the remote system. The request to the transfer son process contains a local file name, remote file name, remote message file name, and buffer to write to the remote message file along with some options such as overwriting the remote file and purging the local file after transfer. The program is designed to handle file transfer only, message transfer only, or both.

The user flags a series of groups in a given batch of results that are ready to be transferred and then queues the batch name to a son process which will find the flagged groups, put each group into a file, and queue the files to the transfer process process described above. That son process transfers the file to the remote system and writes a record to a remote message file which effectively queues a son process on the remote system to pick up the file and put the data away into the appropriate data base. These remote requests get chained to a second son process which print reports from the data in the data base. Since there is a significant volume of these reports queued directly from the remote system, we have two copies of the son process which produces the reports to avoid building a backlog. However, due to some improvements made to our report program, a single copy has been able to keep up with the load so that the dynamic load balancing trick mentioned earlier isn't really necessary. A third son process in the chain can take the data for the report and put it into our electronic mail system where our clients can dial in and view their data before the hard copy reports are mailed.

Another of our applications for monitor processing involves data base synchronization bewteen two HP3000 systems. We have some master data sets (as in master/slave, not IMAGE master) on one system with mirror image, read only slave copies on a second system. Any time a change is made to master side, the changes are queued to the transfer process described above in the form of a file or for cases of a single record update, the whole IMAGE record is put in the request. The request that ultimately reaches the son process on the slave (remote) system is encoded as to type of update in terms of file, single record, add, update, delete, and what else is in the remainder of the request. The single record updates occur in very high volume but are processed rapidly including transfer as the transfer process not only remains logged on to the remote system as long as it's active but also leaves the remote message file open as long as the message file writes are to the same message file. We saw to it that this was the case to avoid a file opens (for the message file) on a per request basis, let alone remote logins. Since much of the synchronization involves client order changes, maintaining chronological order of the changes is critical as well as just not losing the changes. In other words, queue integrity is important even across systems.

Conclusion.

We have gotten a lot of mileage out of the monitor concept. We even have a monitor system on the HP1000 mass spec data systems mentioned above for processing raw data from the instruments, reporting the processed data, and archiving the raw data to tape automatically. As a matter of fact, our HP3000 monitor concept grew out of the HP1000 version. The physical mechanisms are as different as RTE is from MPE but the basic functionality is the same. The HP1000 monitor is an integral part of the Aquarius software package developed by ETC which is now comes standard with HP's RTE mass spec systems.

Even though the design I've described satisfies our needs, further extentions are certainly possible. A simple extention might involve capturing statistics such as the volume of requests and even processing time. It might then be possible to build in some 'smarts' to the monitor program to evaluate request load and automatically startup/shutdown son processes based on the load. Another extention might be to build in some time delay before the submitted request can be processed or, in effect, be able to time schedule certain requests.

We feel the monitor systems have been a tremendous success in making our computers do what they do best without making the computer users wait thereby increasing their productively. If the users had only computer tasks to perform we would probably be in trouble, but even with our great degree of computer automation, a number of paper tasks still remain, and our users now have more time to devote to these tasks.

MAXIMIZING PERFORMANCE FOR IMAGE DATA BASES

Kathy S. McKittrick
Dynamic Information Systems Corporation
910 15th Street - Suite 640
Denver, Colorado 80202

What exactly is the value of the data that is contained in the data bases on your system? How can you measure that value and what components contribute to the equation?

For years we data processing professionals have been busy developing systems to capture and disseminate information. We do this based on user requirements and requests, often times without understanding or examining how those systems will effect our company's success.

Why is it important that we understand this? Put more specifically, why is it important that a programmer understand the value of a report that he or she is writing a program for? There are two reasons. First, the programmer stands a much better chance of fulfilling the user's requirement if he/she understands its purpose. The communication between a user and a programmer is often challenging and an understanding of the business problem being solved enhances that communication.

Secondly, the programmer's job satisfaction will be greatly enhanced by understanding the use of his/her finished product. If a programmer is simply juggling bits and bytes around, the job becomes like that of an assembly line worker who bolts on door handles, but never gets to see the finished product. We all enjoy our jobs better when we can see the tangible fruits of our labors.

So the first step is understanding the business problem(s) being solved. The next step is quantifying the value of solving that problem.

Let's agree on one basic, simple truth: that users benefit from the information they GET OUT of the system, not the information that they put into the system. "Wait a minute!" you might say, "You can't get any information out of the system until its been entered into the system!" True enough! But entering information into the system is not the goal. It's the necessary evil. You can enter customer information, orders, invoices, and receivings all day long but unless you have a good mechanism for retrieving that information in meaningful formats, all that entry is useless. Most computer systems contain vast amounts of data; much of which is difficult or impossible for users to get at. And how ironic! If users can't get to it, it has no value!

Let's talk about some of the types of information that are "locked" inside computer systems and data bases. Then we'll talk about why its so difficult to get at and some of the strategies that we can employ to do a better job of making information available.

INFORMATION THAT IS "LOCKED" INSIDE YOUR SYSTEM

During this discussion, be thinking about how our hypothetical company (ACME Distributing Company) may have some similarities to your own company, at least in-so-far as applications go. ACME Distributing uses an HP3000 series 68 for processing orders, processing shipments, invoicing, accounts receivable and accounts payable. There are fifty order entry operators on the system during business hours and if things are going well they are entering orders throughout the day. There are four terminals in shipping so that orders can be updated with a "shipped" status as soon as they've left the loading dock. There are five receivables clerks and five payables clerks using the system fairly consistantly during the day to enter payments, request checks, and do collections.

There are also ten executive and mid-level managers who have on-line access to the system. A couple of years ago the MIS department purchased a report writer so these management users could easily (meaning without a programmer), extract information from the system. Prior to using the report writer, management had received reports from the system, however these often included too much detail or a summary of all information. Typically, management wants to see only the exceptional information.

About six months ago, the MIS Director had a meeting with the executive staff to discuss system performance. After many complaints from users on the system being too slow, an investigation was done to discover the cause of the bottle neck. The MIS Director reported that two things had to be done to improve performance:

1. Reports run by management often required serial reads and these reports would now be run only at night.

2. The series 68 machine must be upgraded to a series 70.

It turns out that the management reports were already taking from 20 minutes to two hours to run and managers impatient for faster results, were disappointed to hear they were the ones being knocked off of the system. But they also understood the wisdom of servicing customers as quickly and efficiently as possible, so they didn't complain too much.

They now had to wait a full day to get answers to their questions; questions that sometimes affected what inventory was purchased or moved from one location to another. They soon found that this also affected customer service. The management information so critical to doing business on an hour-by-hour basis was "locked" inside their computer system and unavailable to those who would benefit from it.

This is a common story in our industry. High priced decision makers are prohibited from getting timely information upon which to base their decisions. The priority is given to ENTERING the information when the prime benefit of entering the information is BEING ABLE TO RETRIEVE IT.

Anytime you say to a user, "I'm sorry, you can't run that during the day", you may be impacting your company's bottom line or their ability to do business.

The Sales Manager at ACME, for example, heard one day from a sales representative that a large order for blue pens would be placed later in the week by an important customer. The Sales Rep stressed that it was important that ACME be able to deliver the pens the next day. There wasn't enough stock in the local warehouse so the Sales Rep wanted to know if the Sales Manager could have inventory shipped in from other locations. This required the Sales Manager to run a report which looked at blue pen inventory in all twenty area locations, subtract inventory that had been committed during sales that day, and predict how much could be shipped without impacting the other locations. This report was DISALLOWED during the day.

So, the Sales Manager set it up to run that night. Unfortunately, he entered the part number incorrectly and when he came in the next morning he discovered that nothing came out on the report. Even though it had run for two hours.

Back to the drawing board. He resubmitted the job to run that night. Now he'd have his answer on Wednesday. (He was more careful about entering the part number this time).

Wednesday morning. The important customer's order comes through. The Sales Rep is on the phone to the Sales Manager. The report has come through but just a little too late. It turns out that inventory has been committed at the alternate locations and can't be shipped to the important customer. The Sales Manager has already called the factory and they can drop-ship the pens to the customer by Friday afternoon. Not good enough. The Sales Manager has a decision to make. Either divert the inventory in stock to the important customer and make many smaller customers angry or deliver to the smaller customers as promised and risk losing the important account.

And all because ACME was trying to save on system performance. If the Sales Manager could have run the report on Monday morning and again on Monday afternoon after his initial error, the manufacturer could have delivered in time.

When situations like this occur it becomes clear that we must find a way to provide timely information and there is always a solution. It becomes a question of "cost versus benefit" not "possible versus impossible".

WHY IS IMPORTANT INFORMATION LOCKED IN???

We've talked about system performance as a culprit in making critical information unavailable. But what are the underlying reasons for the poor performance?

There are many. We'll talk about three key areas in this paper but understand that in complex multi-user, multi-application systems, there are no simple answers. Each individual component of a system must be examined as-well-as taken as a part of a whole.

The three areas we'll focus on here are:

- Data Base Design
- Volume of Data
- Available Indexing/Retrieval Methods

Data Base Design

The design of a data base that is being used to provide information to users on a regular basis can have an enormous impact on performance and therefore, the users' ability to retrieve information. For example, if we need to obtain information from fifteen different data sets for a particular report, we're going to be expending far more disc I/Os than if we retrieved information from only one data set. And in a reporting or inquiry application disc I/O is generally the most significant factor in the performance equation.

The single biggest reason for poor data base design is that we try to use a single data base for too many purposes. We cannot for example, optimize a design for transaction

DESIGN FOR THROUGHPUT

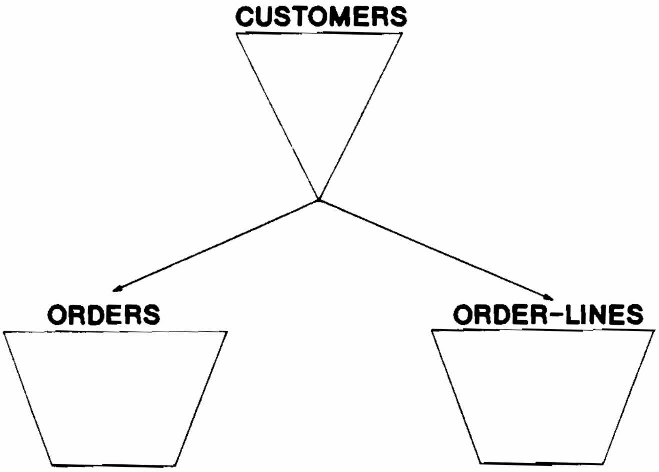


FIGURE 1

throughput and data retrieval both. By trying to use one design to solve both problems we wind up with a design that is not optimized for either task.

In a sales application for example, if we were designing to optimize throughput with no thought to retrieval requirements, we might have a design similar to the one you see in Figure 1. We have a customer master containing name and address information, an order detail containing information such as the date of the order, the total dollar amount of the order, bill-to and ship-to information, and an order line detail, with one record for each line on the order. This data set would contain the inventory item number, unit price and quantity information. Notice in this design I have included only one path to each detail data set; the ACCOUNT-NUMBER of the CUSTOMERS master links to each detail set. Remember, I've designed this data base in order to optimize throughput (adding records), and as a result I've minimized the I/O overhead when adding records by limiting the number of paths.

It is important to note that each path on an IMAGE detail data set requires an average of four I/Os each time a record is added or deleted.(1) By minimizing the number of paths on a set, you reduce update overhead.

So adding records to the ORDERS and ORDER-LINES data sets will be extremely efficient.

Now let's look at a second data base design. This one will be optimized for retrieval purposes and will include only the ORDERS data set. The ORDERS data set contains the following fields:

ACCOUNT#	(the customer's account number)
DATE-ORDERED	(the date the order was placed)
SALES-REP	(the Sales Representative's initials)
REGION	(the sales region)
STATUS	(the status of the order, ie., shipped, back ordered, invoiced, etc.)
ORDER#	(a sequentially assigned sales order number)
ORDER-TOTAL	(total dollar amount of the order)
STATUS-YRMO	(a concatenation of the status field and the year and month of the order)

We'll further assume that our users need to select records based on DATE-ORDERED, SALES-REP, REGION, STATUS, and STATUS-YRMO. Given this scenerio, this "Informational" data base would probably be designed as shown in Figure 2. Remember, we're not concerned here with update overhead, just retrieval capabilities. So we've provided our users with a path for each field that will be used as selection criteria.

[1] Robert M. Green, F. Alfredo Rego, Fred White, David J. Greer, Dennis L. Heidner, *The IMAGE/3000 Handbook*, (Wordward 1984).

DESIGN FOR RETRIEVAL

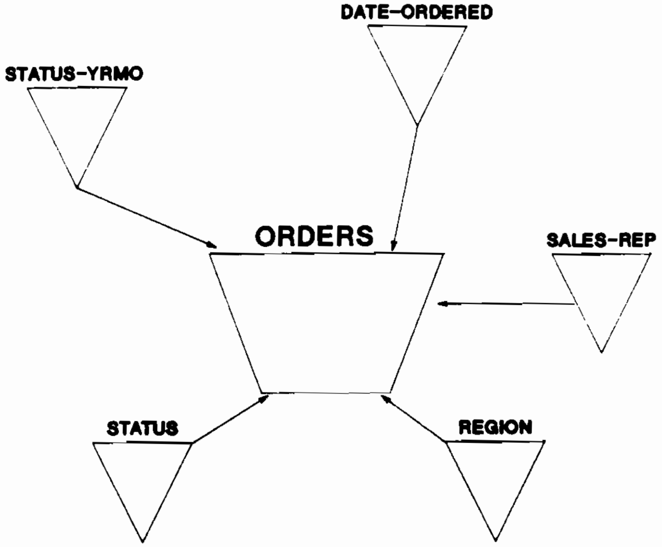


FIGURE 2

THE COMPROMISE

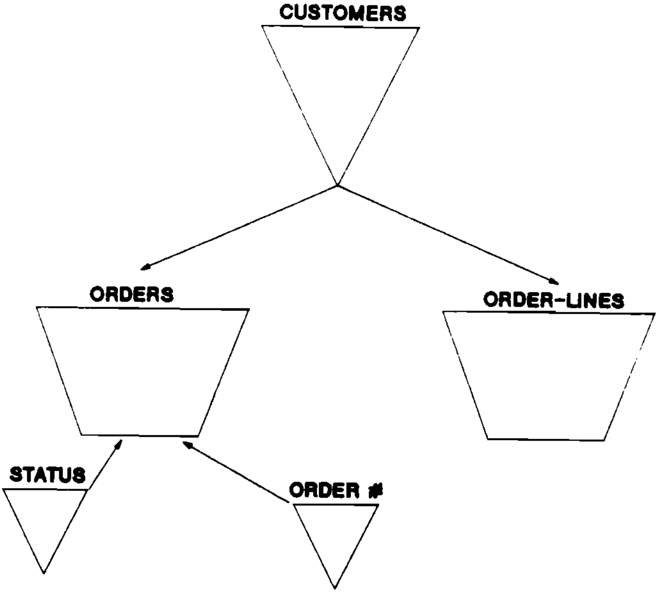


FIGURE 3

It is easy to see that these two data base designs are very different due to the fact that they are being used for two different purposes. But this is not the way that we usually design systems. Typically we come up with a data base design that is a compromise between our retrieval and update requirements. In our order entry data base example, it would look something like the design in Figure 3. We've provided our users with the bare minimum in terms of paths in order to minimize the update overhead. This means that due to many retrievals the ORDERS data set will have to be serially read. We may have to limit our users access to the information.

In addition, as we add orders, we're going to be incurring more overhead due to the additional paths that we decided are essential. The compromise data base is less efficient for updating AND retrieving information.

To compare the efficiency of a compromise data base versus a "split" data base approach we'll run through a possible scenerio. Refer to Figures 1, 2, and 3 for design information. To do this we'll make some basic assumptions:

1. 600 orders are entered into the system on a daily basis.

2. An average order includes one ORDERS entry and ten ORDER-LINE entries.

3. There are 250,000 entries in the ORDERS data set.

4. The ORDERS entries are 256 bytes in length and are blocked at five records per block.

5. There are 20,000 shipped orders (STATUS of "SH").

6. There are 2,000 orders that were entered in March of 1987.

7. There are 200 orders that were entered in March that are in a shipped status.

8. A report is run daily selecting orders that have shipped in the current month.

First, we'll run through the numbers for the compromise data base. In order to calculate the I/O required to enter the orders, we'll use the following equation:

```
        600 x (3 + (4 x 3)) =  9,000 I/Os  (ORDERS data set)
(10 x 600) x (3 + (4 x 1)) = 42,000 I/Os  (ORDER-LINE data set)
                      TOTAL  51,000 I/Os to enter orders
```

Remember that our formula for I/Os to add detail data set records is 3 + (4 x #paths). One ORDERS record and ten ORDER-LINE records are used per order.

Now, to provide the report (on all orders shipped in March), we need to use the STATUS path of the ORDERS data set. Since STATUS is not likely to be the primary path for this set, we assume that to read each record in this chain requires one I/O, and performing the DBFIND to find the chain head for the "SH" (shipped) path costs one I/O.

DBFIND for chain head	1
DBGETs to read chain	20,000
TOTAL	20,001

In summary, to enter orders on our "compromise" data base costs us 51,000 I/Os per day and to run the report costs us 20,001 I/Os.

Entry	51,000	I/Os
Retrieval	20,001	I/Os
TOTAL	71,001	I/Os for daily processing

Now let's look at what it takes to accomplish these same tasks using our split data base approach. In this case, we have only one path on the ORDERS data set, so the calculation for adding orders would be:

600 x (3 + (4 x 1)) = 4,200 I/Os (ORDERS data set)

(10 x 600) x (3 + (4 x 1)) = 42,000 I/Os (ORDER-LINE data set)

TOTAL 46,200 I/Os to enter orders

To update the data base designed for retrievals (we'll talk about how to do this later) the calculation would be:

600 x(3+(4 x 5)) = 13,800

To run the report from our data base designed for retrievals, we can use the STATUS-YRMO path, and read only the 200 records that qualify for our selection criteria. Adding the one I/O for the DBFIND, this gives us:

DBFIND for chain head	1
DBGETs to read chain	200
TOTAL	201

The total number of I/Os for daily processing on the split data base approach is:

Entry	46,200	I/Os
Update	13,800	I/Os
Retrieval	201	I/Os
TOTAL	60,201	I/Os for daily processing

And how does that compare with the I/Os it took to process on the compromise data base?

Compromise data base	71,001	I/Os
Split data bases	60,201	I/Os
	10,800	I/Os DIFFERENCE!!!!!

By splitting the data bases we reduce our I/O overhead by 15%! We will increase the amount of disc space used, but disc space is a one time cost whereas the cost of slower processing occurs over and over again. Just imagine improving your own system performance by 15%!

Let's look at what the disc space differences might be for the sample data bases. First the compromise data base. For simplicity sake, let's assume that each of our three main data sets (CUSTOMERS, ORDERS and ORDER-LINES) contain media entries that are 256 bytes (or one sector) in length. Let's also assume that we have 10,000 customers, 250,000 orders, and 2,500,000 order lines.

For our auto masters (ORDER# and STATUS) we need to calculate based on the size of each field plus the chain pointers times the number of different values likely to be contained in each set. Let's assume that there are ten possible status codes and that STATUS is a two-byte field. In the case of ORDER#, let's assume an eight-byte field. Since there will be a unique ORDER# for each ORDERS record there will be 250,000 records in this auto master.

STATUS	(2 + 12) * 10	=	140 bytes
ORDER#	(8 + 12) * 250,000	=	5,000,000 bytes
CUSTOMERS	256 * 10,000	=	2,560,000 bytes
ORDERS	256 * 250,000	=	64,000,000 bytes
ORDER-LINES	256 * 2,500,000	=	640,000,000 bytes
			711,560,140 bytes or 2,779,532 sectors

Now let's compare that with our split data base approach. For the data base designed for throughput, the disc space required would be:

CUSTOMERS	256 * 10,000	=	2,560,000 bytes
ORDERS	256 * 250,000	=	64,000,000 bytes
ORDER-LINES	256 * 2,500,000	=	640,000,000 bytes
			706,560,000 bytes or 2,760,000 sectors

The disc space required for the data base designed for retrieval would be:

ORDERS	256 * 250,000	=	64,000,000 bytes
STATUS	(2 + 12) * 10	=	140 bytes
DATE-ORDERED	(6 + 12) * 220	=	3,960 bytes
SALES-REP	(4 + 12) * 100	=	1,600 bytes
REGION	(4 + 12) * 10	=	160 bytes
STATUS-YRMO	(6 + 12) * 120	=	2,160 bytes
			64,008,020 bytes or 250,032 sectors

(NOTE: We are making the following assumptions for the number of records in each of the automatic masters:

STATUS - we established earlier we would assume 10 status codes

DATE-ORDERED- assuming here one years worth of data;
approximately 220 work days per year.

SALES-REP - we assume 100 Sales Reps

REGION - we assume 10 regions

STATUS-YRMO - given 10 possible status codes and 12 months, there are 120 possibilities here.

Furthermore, w: add 12 bytes to the length of each record to account for the chain pointers and chain count in the master sets).

For both the throughput and retrieval data bases:

```
          2,760,000 sectors
        +   250,032 sectors

TOTAL     3,010,032 sectors
             versus
          2,779,532 sectors (for the compromise data base)
```

In conclusion, by using more disc space and designing separate data bases for separate purposes, we are providing the business with more benefit from the computer system. The cost benefit when measured in terms of resources looks like this:

COST	BENEFIT
230,500 sectors disc space	15% throughput improvement

There's one issue that we have yet to discuss regarding this split data base strategy. That is keeping the information in the second data base up to date. There are several approaches that you can take.

There are several products on the market which are designed to keep two or more data bases in sync. (Silhouette and BackChat are two that I know of.) In some cases these products allow you to mirror only selected data sets, and at configurable intervals.

If you cannot use a separate machine for informational data bases consider running a nightly process to move records from the "operational" or throughput data base. This approach will save your on-line users from the overhead of updating the informational data base.

Finally, make sure that your operational and informational data bases reside on separate disc drives. If at all possible, providing a separate machine for informational systems is the most effective solution of all. This way there is no impact on the operational users even when updating the informational data base during on-line hours.

Volume of Data

Sometimes the shear volume of data that we must extract information from is a limiting factor, even when the operational and informational data is separate. In the case of the sales order information, it is not unlikely or unreasonable in some businesses to keep three year's of sales information available for management reports.

So, if we keep three year's worth of information in our informational data base, and we need to report on total sales dollars for a particular region in a particular time frame, what are we looking at in terms of reporting time? Let's take a closer look.

Remember that we had 250,000 ORDERS in our data base that contained one year's worth of information. Based on that figure let's assume that three years worth of information would be three times that figure or 750,000 records.

Now we need to select on both the REGION (one of the paths defined in our informational data base illustrated in Figure 2) and the DATE-ORDERED. Let's say, hypothetically, that we're looking for orders in Region 7 during March of 1987.

Based on the fact that there are ten sales regions in ACME Distributing Company, let's assume that 10% of the records (or 75,000) would be orders for Region 7.

Let's further assume that since we have three years worth of history, that approximately 21,000 sales occurred each month. If one-tenth of those total sales occurred in Region 7, then 2100 records would ultimately qualify for our report selection. But how many records do we have to read in order to get to those 2100 records of interest?

Refer again to Figure 2, our data base designed for retrieval. You can see that the only IMAGE path available to us is the "REGION" path. We can't use the DATE-ORDERED path because we're selecting on a RANGE of dates, not specific dates. IMAGE will only allow us to use full key values when doing a chained read.

We can therefore assume that we're probably going to do 75,000 I/Os in order to select the 2100 records of interest. Why will we get no benefit from the blocking factor? Because unless the records are sorted in sequence by region before they are added to this data base, (a scenerio that is highly unlikely), it is unlikely that any of the records for Region 7 will wind up in the same block. Refer to Figure 4. You can see how records for region 7 are spread throughout the detail data set. Since one I/O is performed per block (not taking disc caching into account) and we only get one record of interest per block, we will do 75,000 I/Os to read 75,000 records.

Figure 5 shows the advantage we would realize if we were to make REGION our primary path and reload this detail set in primary path sequence. Because our blocking factor is 5 we would do one-fifth the I/Os after a reload. Products that can perform highspeed single set reloads are Adager, Dbmgr and Dbgeneral.

But is it practical to make REGION our primary path? Only if it is the path that we will most often use to retrieve information. That is the whole purpose of specifying a primary path; to allow for more efficient retrievals. Unfortunately, REGION is unlikely to be a candidate for primary path.

So, in fact, it looks like we'll do approximately 75,000 I/Os to obtain the information we need.

Unless we use some alternative method for retrieving the data. We have two choices:

- Large blocked reads (called MR NOBUF)
- Alternative indexing methods

Why have I not included MPE caching as an alternative? Because MPE caching uses the RANDOM FETCH QUANTUM when doing serial reads of IMAGE data sets, not the sequential fetch quantum. The whys and wherefores of this are beyond the scope of this paper but you can test this yourself as I did. Take a data set within an IMAGE data base and perform a serial read in a batch job (a QUERY "FIND ALL" command will do). Then write all the records in the set out to a flat file. Using the same blocking factor as in the IMAGE data set. (Be sure and set the MPE record size large enough to account for the MEDIA record length, so that you're comparing apples to apples.) Then do a serial read on the MPE file in a batch job. Compare the time it takes for each and you'll find that the read of the MPE file is much faster. This, of course, assumes that your sequential fetch quantum is set fairly high and your random fetch quantum fairly low (as recommended by HP).

So...back to our two alternatives; large blocked reads and alternative indexing methods. The method that you choose here will depend largely on how many records are being selected from the whole and whether or not you need to select from more than one data set.

The way large blocked reads work (these are also known as MR NOBUF or multi-record no buffering) is to read many thousands of bytes of information per physical disc I/O. This method can significantly reduce the time an I/O intensive job takes (such as our report) by reducing the number of disc reads required. In the case of our report, to read the REGION chain took 75,000 I/O's. Now let's look at what a serial read using an MR NOBUF tool such as SUPRTOOL by Robelle would take.

The default buffer size for SUPRTOOL is 14,366 words, or 28,732 bytes. Given that our ORDERS records are 128 words (256 bytes) in length, we can read 112 records per disc I/O. Given that we have 750,000 records in the ORDERS data set, this would mean that we can accomplish our task using 6,697 disc I/Os -- a savings of more than 68,000 I/Os over our chained read! (See Figure 6).

$$\frac{\text{21,000 record pointers for "8703"}}{\text{64 record pointers per I/O}} = \text{329 I/Os}$$

$$\frac{\text{75,000 record pointers for "07"}}{\text{64 record pointers per I/O}} = \text{1,172 I/Os}$$

1,501 I/Os

(to retrieve the records using the relative record numbers) 2,100 I/Os

3,601 I/Os

FIGURE 8

The second possibility we'll look at is alternative indexing techniques. A method available for multi-field selection on IMAGE data bases is OMNIDEX by DISC. Using this method to index the REGION field and the first four characters of the DATE-ORDERED field (which contain year and month) we can select the same 2,100 records using only 3,601 I/Os! Let me explain.

The way this indexing technique works, in simple terms, is to maintain an index of values and their associated record pointers. With IMAGE detail data sets the relative record number is used as the internal record pointer. (In the case of master data sets, the IMAGE search item value is used as the internal record pointer).

Selection criteria (or indexed values) can be words within fields or specific portions of fields, such as year and month from the date field.

When selecting records by two or more selection criteria, OMNIDEX simply reads each set of record pointers into stack and finds the intersection of those two sets. Figure 7 illustrates what a subset of our sales history report selection might look like. This Figure illustrates what the selection of the 2,100 entries for our report entails. There are several record pointers which appear in both lists. These constitute the intersection.

Now it turns out that each physical index record contains up to 64 record pointers. What this means is that 64 record pointers are read per disc I/O, if there is no benefit realized from blocking. (These index sets can be reloaded so that up to seven index records or 448 record pointers are retrieved per I/O). Figure 8 shows the calculations used to arrive at our figure of 3,601 disc I/Os using Omnidex indexing methods to retrieve records for this report.

From this example, you can begin to see how the method used to improve performance is dependant upon the number of records that are ultimately used. Using sophisticated indexing techniques was by far the best way to perform this retrieval. In fact, half the number of I/Os were used as compared to the number used for an MR NOBUF serial read.

But let's look at another example that shows a different result. If instead of looking for one month's sales for Region 7, we were looking for one year's sales, the numbers would look different for the indexed retrieval.

We stated earlier that the 750,000 ORDERS records represented three year's worth of sales. Let's assume, then, that in 1987 there were a total of 250,000 orders, or one third of the total. If we can again assume that since Region 7 is one of ten regions, that 10% of those 250,000 orders in 1987 will be in Region 7, then 25,000 total orders will be selected for our report.

Looking back at Figure 6 at the calculations for an MR NOBUF read, we see that the retrieval will take 6,697 I/Os. These numbers do not change when we select a different number of records because we're not dealing with record pointers.

However, Figure 9 shows that our indexed retrieval is no longer our best choice. When we qualify this large a number of records, large blocked reads are our best choice.

CONCLUSION

The bottom line then is that once we understand and buy into the value of providing certain timely information to our users, there is no one answer or "black box" that we can plug into our application in order to optimize performance. There are a few basic rules, however that you can follow that will help guide you in your quest for providing information quickly:

1. Provide Management Reporting Systems using data bases designed specifically for retrieval purposes. Don't try to cut corners on disc space at the expense of timeliness of information.

2. Don't assume that using an IMAGE chain is always more efficient than a serial read. Depending on the number of entries in the chain and the blocking factor of the data set, even a standard IMAGE serial read may be more efficient.

3. Use tools such as Adager to reload detail data sets where you often do retrievals, particularly where chains are long and/or blocking factors high.

4. For on-line applications, ad-hoc reports and exception reporting, use sophisticated indexing techniques such as Omnidex to maximize performance.

5. For applications that extract large volumes of data, use a MR NOBUF tool such as Suprtool to minimize I/Os.

6. A combination of these techniques implemented intelligently can provide users with an extremely valuable informational system.

$$\frac{\text{250,000 record pointers for "87"}}{\text{64 record pointers per I/O}} = \text{3,907 I/Os}$$

$$\frac{\text{75,000 record pointers for "07"}}{\text{64 record pointers per I/O}} = \text{1,172 I/Os}$$

$$\underline{\hphantom{\text{5,079 I/Os}}}$$

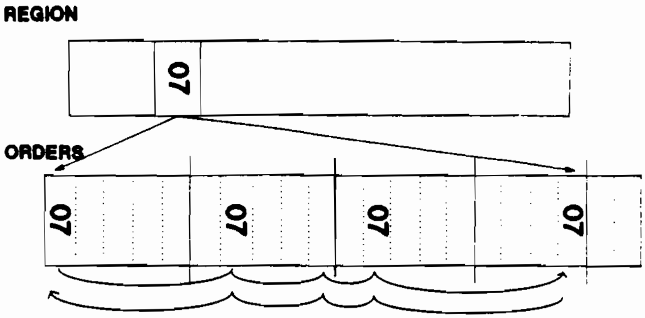
$$\text{5,079 I/Os}$$

(to retrieve the records using the relative record numbers) 25,000 I/Os

$$\underline{\hphantom{\text{30,079 I/Os}}}$$

$$\text{30,079 I/Os}$$

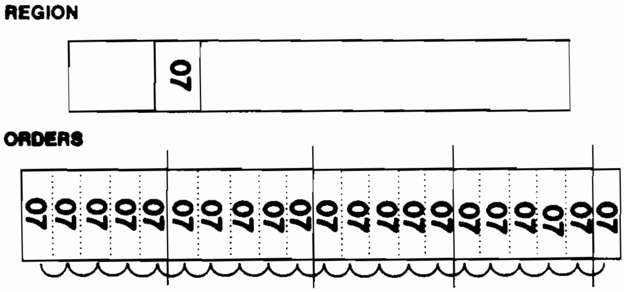
FIGURE 9



75,000 I/Os

FIGURE 4

AFTER SORTING



15,000 I/Os

FIGURE 5

CHAINED READ

75,000 RECORDS WITH REGION = "07"

75,000 I/Os

MR NOBUF READ

$$\frac{\text{28,732 BYTES/IO}}{\text{256 BYTES/RECORD}} = \text{112 RECORDS / I/O}$$

$$\frac{\text{750,000 RECORDS}}{\text{112 RECORDS/IO}} = \text{6,697 I/Os}$$

FIGURE 6

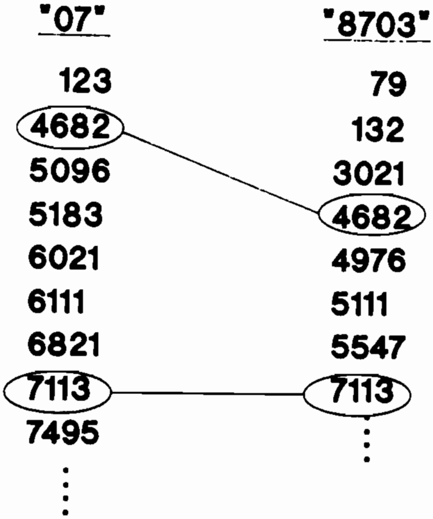

"07"
123
4682
5096
5183
6021
6111
6821
7113
7495
"8703"
79
132
3021
4682
4976
5111
5547
7113

FIGURE 7

MPE/XL PROGRAMMING

by Eugene Volokh
VESOFT, Inc.
1135 S. Beverly Dr.
Los Angeles, CA 90035

ABSTRACT

In 1983, I wrote a paper called "MPE PROGRAMMING" (presented at the INTEREX Montreal conference), which showed how you could do some remarkable things with MPE alone, without the aid of a custom-written program. MPE Programming was the art of writing system programs entirely in the "language" of CI commands (possibly with some help from standard, HP-supplied utilities).

The main advantages of MPE Programming were ease of writing and ease of maintenance. The idea was that a couple of dozen MPE commands in a job stream were easier to deal with than a custom-made SPL or COBOL program, especially since when you write a program, you'll have to always keep track not just of the job stream, but also the program's source and object files. UNIX, incidentally, has a very powerful "Command Interpreter Programming" facility (such programs are called "shell scripts"); UNIX users often write very many shell scripts to do things that would otherwise require some rather cumbersome C or PASCAL system programs.

Unfortunately, MPE/V (and earlier MPE versions) were not really designed for any sort of sophisticated MPE programming. Many of the tricks I showed in my original paper bordered, I must admit, on the perverse. For instance, to find out if you're in job mode or session mode (without writing a program that calls WHO), I suggested that you execute the :RESUME command.

Why the :RESUME command, of all things? Well (almost by accident), the :RESUME command returns one error condition if done in a job and another if done in a session (but not in break). We could then completely ignore the actual function of the :RESUME command, and look only at its "side effect" -- the value of the CIERROR JCW, which told us whether we were in a job or sesion.

Similarly, to see if a file existed, we'd do a :LISTF ;$NULL of it. This was not because we wanted to see information about this file (if we did, we wouldn't put on the ;$NULL) -- rather, we wanted to see if the :LISTF succeeded or failed. If it failed with a CIERROR 907, this meant that the file didn't exist -- if it succeeded, the file did exist.

MPE/XL was intended to make many of these things a lot simpler to do -- instead of weird, indirect techniques, mechanisms would be provided for easily getting environment information (your logon mode, etc.), file information (does a file exist?), and so on. Seemingly using UNIX as a prototype (in spirit if not always in detail), MPE/XL sought to make MPE Programming a straightforward proposition.

To a large extent, HP succeeded -- MPE/XL has a number of new commands and features that let you do much more powerful things from the Command Interpreter. In some ways, though, some of the features seem at first glance to be more powerful than they really are, and quite a few things that you'd like to do remain tantalizingly out of your reach.

In the process of converting my MPEX/3000 and SECURITY/3000 products to MPE/XL -- and in the process of implementing most of the MPE/XL user interface features in the MPE/V version of MPEX (and in SECURITY/3000's STREAMX module), usable by "classic HP 3000" users -- I learned a good deal about the new MPE/XL features, their strengths and their weaknesses. This paper will try to objectively discuss both; to show you how to use the strengths to their utmost and how to work around some of the weaknesses.

THE NEW FEATURES OF MPE/XL

What exactly are the new MPE programming-related features of MPE/XL? There are several:

* First of all, MPE/XL supports VARIABLES. Think of them as JCWs that can have string values as well as integer values. (Actually, they can have boolean and 32-bit integer values, too.) E.g.

```
:SETVAR FNAME "FOO.DATA.PROD"
```

* MPE/XL PREDEFINES some variables to values such as your user name, your account name, your capabilities, etc. For instance,

```
:SHOWVAR @
HPACCOUNT = VESOFT
HPDATEF = TUE, FEB  9, 1988
HPGROUP = DEV
HPINPRI = 8
HPINTERACTIVE = TRUE
HPJOBCOUNT = 2
HPJOBLIMIT = 2
HPJOBFENCE = 7
HPJOBNAME = EUGENE
HPJOBNUM = 268
HPJOBTYPE = S
HPLDEVIN = 20
...
```

(Don't you wish you'd had this all along???)

* MPE/XL lets you SUBSTITUTE the values of variables (and even EXPRESSIONS involving the variables) into MPE commands -- just as you could always substitute the values of UDC parameters. For example,

```
:SETVAR FNAME "FOO.DATA.PROD"
:PURGE !FNAME
```

is equivalent to

```
:PURGE FOO.DATA.PROD
```

Then you could also say

```
:BUILD !FNAME;DISC=![100*NUMUSERS+25];REC=-64,,F,ASCII
```

it will build a new FOO.DATA.PROD file with room for 100*NUMUSERS+25 records (presumably NUMUSERS is an integer variable previously set with a :SETVAR).

- As shown in the above example, MPE/XL lets you use EXPRESSIONS in variable substitution, in the :SETVAR command, in the :IF command, and in the new :WHILE and :CALC commands:

```
:SETVAR EXPECTEDFLIMIT 100*NUMUSERS+25
:SETVAR FNAME "S"+MODULENAME+".PUB.SYS"
:SETVAR MODULENAME STR(FNAME,2,POS(".",FNAME)-2)
:IF HPACCOUNT<>"SYS" THEN
:IF POS("SM",HPUSERCAPF)=0 THEN  << user doesn't have SM >>
```

As you can see, the expressions can involve either numbers or strings, and a number of useful operators have been defined, such as:

```
   + to concatenate strings;
   STR to extract substrings;
   POS to find the position of one string in another;
   UPS to upshift a string;
```

and many others.

- Perhaps the most useful of the defined operators is FINFO, which takes a filename and an option number and returns a piece of information about that file:

```
   FINFO(filename,0)  = TRUE if file exists, FALSE if it doe
   FINFO(filename,1)  = string with fully-qualified filename
   FINFO(filename,4)  = string containing file's creator
   FINFO(filename,8)  = file's creation date, formatted stri
   FINFO(filename,-8) = file's creation date, integer format
   FINFO(fi'ename,9)  = file's string filecode (e.g. "EDTCT"
   FINFO(filename,-9) = file's integer filecode (e.g. 1052)
   and much more.
```

For example, to check if a file exists, you can say

```
:IF FINFO('MYFILE',0) THEN
```

To check if a file's EOF is within 10% of its FLIMIT, you might enter

```
:IF FINFO('MYFILE',19)>=FINFO('MYFILE',12)*9/10 THEN
```

FINFO mode 19 gets you the EOF; FINFO mode 12 gets you the FLIMIT. (The mode numbers are taken from the FLABELINFO intrinsic -- one of the weaknesses of FINFO is that you have to remember these silly item numbers.)

* Commands have been added to OUTPUT and INPUT data:

```
:ECHO NOW WE'LL ASK YOU FOR A FILENAME.
:INPUT FNAME; PROMPT="Please enter the filename: "
:ECHO FNAME = !FNAME, FLIMIT = ![FINFO(FNAME,12)]
```

The :INPUT command can even have a timeout (wait for no more than X seconds) option.

* In addition to MPE/V control structures like :IF, :ELSE, and :ENDIF, MPE/XL implements the :WHILE / :ENDWHILE construct, e.g.

```
:SETJCW I = 295
:WHILE I < 314
:  ABORTJOB #J!I
:  SETJCW I = I+1
:ENDWHILE
```

* Instead of setting up UDCs, you can set up COMMAND FILES. If you want to define a command called S that does a :SHOWJOB, you can build a file called S.PUB.SYS that contains the lines:

```
PARM WHAT=" "
SHOWJOB JOB=@!WHAT
```

Now, whenever you type

```
:S J
```

(for example), MPE/XL will execute the file S.PUB.SYS passing "J" to it as a parameter. Same as a UDC, but no need to :SETCATALOG.

* Actually, whenever you type a command (like S in the example above) that isn't a normal MPE command, MPE/XL doesn't just check for it in PUB.SYS. It instead looks at the variable (remember those?) called HPPATH, and tries to find the file in the groups listed in the variable.

By default, HPPATH is set to

```
!HPGROUP,PUB,PUB.SYS
```

This means "first look in !HPGROUP (i.e. your group), then in the PUB group (of your own account), and then in PUB.SYS". You can change HPPATH to tell MPE/XL to look in UTIL.SYS, PUB.VESOFT, PUB.TELESUP, or what have you.

* In addition to letting you execute command files by just entering their names, you can also run a program just by entering its name (IMPLIED RUN). If you say

```
:SPOOK5
```

MPE/XL will search the groups specified in HPPATH -- if the first file it finds is SPOOK5.PUB.SYS (a program file), it'll run it just as if you'd said

```
:RUN SPOOK5.PUB.SYS
```

Similarly, to run a program in your own group, you can just say

```
:MYPROG
```

and MPE/XL will automatically supply the :RUN (remember, MPE/XL will look in HPPATH to determine which groups it should search -- by default, your group is one of them). If you say

```
:MYPROG "BANANA",5
```

it'll run MYPROG with INFO="BANANA" and PARM=5 (other :RUN command parameters are not available).

* Finally, a few odds and ends:

- The :CALC command works as a general-purpose integer and string calculator.

- Users can now redefine their own prompt by setting the HPPROMPT variable.

- :SETCATALOG lets you add a new UDC file (or remove one) without retyping the names of all the other UDC files (which is cumbersome and risks accidentally unsetting an important file).

- You can :REDO not just the last command, but one of the last 20 commands (or even more than 20 if you so choose). This is actually a very powerful tool -- I'm only including it in "odds and ends" because it's not directly relevant to MPE/XL programming.

These are the features -- what are the benefits?

THINGS THAT ARE NOW EASY TO DO

#1. ENVIRONMENT VARIABLES

One example in my original "MPE Programming" paper involved a UDC finding out whether it's being executed in a session or in a job. This might, for instance, be a logon UDC that you use to set your function keys -- it outputs a whole bunch of escape sequences, which you want to see when you're online, but which will only garble your printout if printed in a job.

In MPE/V, if you recall, checking job/session mode was done this way:

```
SOFTKEYSINIT   << the logon UDC name >>
OPTION LOGON
SETJCW CIERROR=0
CONTINUE
RESUME
IF CIERROR<>978 THEN
  << initialize the softkeys >>
ENDIF
```

Very straightforward, isn't it? The :RESUME command, of course, is not used for :RESUMEing at all; rather, we count on it to generate an error condition -- error 978 if in batch, but a different error (warning 1686) if online.

MPE/XL makes this laughably simple:

```
SOFTKEYSINIT
OPTION LOGON
IF HPINTERACTIVE=1 THEN
  << initialize the softkeys >>
ENDIF
```

Essentially, MPE/XL automatically presets some JCWs to interesting values -- HPINTERACTIVE, HPLDEVIN (your terminal number), HPUSER (your logon user id), etc. This process actually started in MPE/V with the HPYEAR, HPMONTH, HPDATE, HPDAY, HPHOUR, and HPMINUTE JCWs, but MPE/XL has added a lot of new and useful ones.

Some more practical applications are readily apparent and others (the best kind) aren't. For instance, a really nice typing-saver is:

```
:NEWUSER JACK;CAP=!HPUSERCAPF
```

"HPUSERCAPF" stands for "USER CAPabilities, Formatted". It's a STRING variable that indicates which capabilities you currently have, e.g. "AM,AL,GL,ND,SF,PH,DS,IA,BA". The "!" before the "HPUSERCAPF" works much as it would before a UDC parameter -- it tells MPE to substitute in the VALUE of the HPUSERCAPF variable in place of its name.

Thus, the command might end up being:

```
:NEWUSER JACK;CAP=AM,AL,GL,ND,SF,PH,DS,IA,BA
```

You didn't have to type in all of those capabilities -- the !HPUSERCAPF automatically put in all the ones you have.

You might even say

```
:NEWUSER JACK;CAP=![HPUSERCAPF-"AM,"]
```

Saying ![xxx] tells MPE: "Evaluate the expression xxx and substitute in its result". Subtracting two strings in MPE/XL removes the first occurence of the second string from the first -- thus, the :NEWUSER command will become

```
:NEWUSER JACK;CAP=AL,GL,ND,SF,PH,DS,IA,BA
```

(since "AM,AL,GL,ND,SF,PH,DS,IA,BA"-"AM," is "AL,GL,...,BA").

Another nice example is:

```
:FILE SYSLIST=BK!HPYEAR!HPMONTH!HPDATE,NEW;DEV=DISC;SAVE
:STORE @.@.@; *T
```

This will do a system backup and send the listing to a disc file IDENTIFIED BY THE BACKUP DATE.

Thus, you can keep many of your backup listings online (so you could easily tell which tape set and reel number a file was on); each one will be stored in its own file.
For instance, on 20 November 1988, the above commands will be executed as:

```
   :FILE SYSLIST=BK881120,NEW;DEV=DISC;SAVE
   :STORE @.@.@; *T
```

Unfortunately, it's not quite this simple. (Almost, but not quite.) What if we do the :FILE SYSLIST= on the 9th of April? Then, we'd get

```
 :FILE SYSLIST=BK8849;...
```

-- not quite what we want. We'd like the month and day to be zero-padded, so that the file names will be more comprehensible and a :LISTF will show them in the right order (i.e. not show BK8849 after BK88410 and BK881231). How can we do this? Well, how about

```
 :FILE SYSLIST=BK![10000*HPYEAR+100*HPMONTH+HPDATE];...
```

Instead of substituting the month and the day in directly, we calculate the value 10000*HPYEAR+100*HPMONTH+HPDATE. Since this is arithmetic, not textual substitution, "zero-padding" will occur -- the 9th of April of 1988 will yield 880409. Then, we textually substitute the resulting value into the :FILE equation:

```
 :FILE SYSLIST=BK880409;...
```

Even the additional power of MPE/XL doesn't remove the need for a little ingenuity.

Finally, one more useful little UDC:

```
 HIPRI !JOBNUM
 ALTJOB #J!JOBNUM;INPRI=14
 SETVAR OLDJOBLIMIT HPJOBLIMIT
 LIMIT ![HPJOBCOUNT+1]
 LIMIT !OLDJOBLIMIT
 DELETEVAR OLDJOBLIMIT
```

Three guesses as to what this does? Give up? Well, you :STREAM a job and find it at the bottom of the WAIT queue; you want it to execute, but you don't want to let any of the other WAITing jobs through.

This UDC:

* Alters the job to input priority 14 (the highest priority possible).

* Saves the old job limit (indicated by the built-in variable HPJOBLIMIT) in an MPE/XL variable (OLDJOBLIMIT).

* Sets the job limit to HPJOBCOUNT -- the number of currently executing jobs -- plus 1, thus letting the topmost WAITing job (the one you just :ALTJOBd) through.

* Sets the job limit back to what it was before.

* Just for cleanliness, deletes the OLDJOBLIMIT variable.

Voila! The one problem I can see is that the UDC expects only a job NUMBER, not the leading "#J" -- if a user types

```
HIPRI #J123
```

then the very first line will be

```
ALTJOB #J#J123;INPRI=14
```

-- MPE won't like this much. We'd like to let the user type either

```
HIPRI 123
```

or

```
HIPRI #J123
```

whichever he prefers.

The solution is again fairly simple, taking advantage of MPE/XL's provisions for strings and for string operators:

```
HIPRI !JOBNUM
IF UPS(LFT("!JOBNUM",2))="#J" THEN
  ALTJOB !JOBNUM;INPRI=14
ELSE
  ALTJOB #J!JOBNUM;INPRI=14
ENDIF
SETVAR OLDJOBLIMIT HPJOBLIMIT
LIMIT ![HPJOBCOUNT+1]
LIMIT !OLDJOBLIMIT
DELETEVAR OLDJOBLIMIT
```

The key here is the :IF expression -- it extracts the leftmost 2 characters of the string containing JOBNUM (LFT("!JOBNUM",2)), upshifts them (UPS(LFT("!JOBNUM",2))), and then compares them against "#J". If the characters are equal to "#J", then we just do an :ALTJOB !JOBNUM; if the characters are something else (presumably the start of the job number), then we insert a #J in front of them.

#2. FILE INFORMATION

One of the most valuable new features of the MPE/XL CI is the ability to obtain FILE INFORMATION. Remember the old MPE trick of finding out if a file exists or not?

```
SETJCW CIERROR=0
CONTINUE
LISTF MYFILE;$NULL
IF CIERROR=907 THEN
  << file doesn't exist >>
ELSE
  << file exists >>
ENDIF
```

Again, what we're doing here is executing a command (:LISTF) not for its main purpose, but rather for a side effect -- if we give :LISTF a file that doesn't exist, it'll set the CIERROR JCW to 907; if the file exists, CIERROR will remain 0.

MPE/XL is much more straightforward:

```
IF FINFO('MYFILE',0) THEN
  << file exists >>
ELSE
  << file doesn't exist >>
ENDIF
```

The FINFO function returns information about the file whose name is passed as the first parameter. The second parameter tells FINFO which information is to be gotten; 0 means a TRUE/FALSE flag indicating whether or not the file exists. Other values ask for other things, such as file code, EOF, FLIMIT, etc.

Applications for this abound. For instance, your job stream might check to see if a file has 100 or more free records:

```
:IF FINFO('DATAFILE',19) > FINFO('DATAFILE',12)-100 THEN
:  TELLOP File DATAFILE is &
:         ![FINFO('DATAFILE',19)*100/FINFO('DATAFILE',12)]% full!
:ELSE
...
```

FINFO(xxx,19) returns xxx's EOF; FINFO(xxx,12) returns xxx's FLIMIT; if EOF > FLIMIT-100, we send a message to the operator indicating how full the file is (again, the wonders of expression substitution).

Another application is that we can now :BUILD files that are the right size (rather than choose some number and hope that the file won't overflow) --

```
:BUILD NEWFILE;DISC=![FINFO('DFILE1',19)+FINFO('DFILE2',19)+100]
```

This builds NEWFILE to be large enough to fit all of DFILE1, all of DFILE2, and 100 more records on top of that. Unfortunately, note that we still can't figure out, say, the number of entries in an IMAGE database (which you might very well want to use in calculating a file limit) -- we're still restricted to the rather limited set of features that HP in its wisdom chose to provide to us.

There are, in fact, two pretty serious problems with FINFO:

* For one, there are still a number of things that FINFO just doesn't provide. To name a few:

  - The NUMBER OF SECTORS in a file. I found myself wanting to write a command file that compared the number of sectors a file occupied before and after a certain operation, but there was no way of getting this information.

  - The file's LAST ACCESS DATE/TIME and LAST RESTORE DATE/TIME (FINFO gives us the creation date and the last modify date, but not the last access date or the last restore date).

  - The file's security information -- :RELEASEd/:SECUREd flag, security matrix, etc. It would be quite nice, for instance, to check the access you're allowed to a file before running a program that might abort quite bizzarely if it isn't given the access it wants.

- Whether or not the file is currently IN USE (and if it is, in what mode).

- The NUMBER OF EXTENTS in a file, the number of user labels, and others.

In fact, if you look at the FINFO option numbers, you'll find that they're pretty much a subset of the option numbers of the FLABELINFO intrinsic, which also lets you obtain file information. Why a subset? Why not just implement all the FLABELINFO options (though even that would still leave some options out).

All the file attributes -- certainly all those listable with :LISTF ,2 and MPE/XL's new :LISTF ,3 -- should be easily obtainable from the CI.

* Perhaps more important than the omitted functions is the fact that

ALL THE FINFO OPTIONS ARE "MAGIC NUMBERS".

When you saw the command

```
:IF FINFO('DATAFILE',19) > FINFO('DATAFILE',12)-100 THEN
```

was it clear to you what FINFO(xxx,19) and FINFO(xxx,12) did? If HP is going to implement file access functions, why not have an FFLIMIT('DATAFILE'), an FEOF('DATAFILE'), an FFILECODE('DATAFILE') and so on? Or, if you want a single function, why not let the user say

```
FINFO('DATAFILE','FLIMIT')
```

or

```
FINFO('DATAFILE','EOF')
```

Sure, it would take a little bit of extra time to parse, but think of the advantages in clarity.

Of course, you can remedy this problem yourself by setting up (probably in a logon UDC) variables or JCWs that are set to the the appropriate FINFO values, e.g.

```
SETVAR FIFILECODE 9
SETVAR FIFLIMIT 12
SETVAR FIEOF 19
...
```

You'd probably have to set either 14 or 18 of these variables, and then you could say

```
:IF FINFO('DATAFILE',FIEOF)>FINFO('DATAFILE',FIFLIMIT)-100 THEN
```

Unfortunately, you and I both know most people won't do this -- they'll use the "magic numbers" and let you try to figure out what's going on.

Even if you set up all the variables and use them consistently, you'll lose one of the greatest advantages of command files: their stand-alone nature. Your "MPE programs" will now rely on your logon UDC and its SETVARs -- if it gets deleted, they'll stop working. If you want to copy your job stream or other MPE program onto some other machine, you'll have to be sure that the other machine has the same logon UDCs. The point is that HP shouldn't have made you (or let you) use "magic numbers" in the first place.

This might seem like looking a gift horse in the mouth -- for fifteen years, we had nothing, and now, when they give us something, we want more. However, it seems almost a shame that HP, having made the CI so much more powerful, didn't implement such reasonable and useful features.

#3. INPUT AND OUTPUT

A major shortcoming of MPE/V was the absence of any general output command. Why, to output a simple message, you had to have a UDC like

```
DISPLAY !STUFF
OPTION LIST
COMMENT !STUFF
```

The OPTION LIST would cause the UDC body -- in this case COMMENT followed by the DISPLAY parameters -- to be output; to output any message, you'd say

```
DISPLAY "HI THERE!"
```

Unfortunately, this would display not HI THERE!, but rather

```
COMMENT HI THERE!
```

To avoid the output of the "COMMENT ", you had to output special escape sequences to backspace the cursor and clear the line -- of course, this wouldn't work on a printing terminal. All this bother just to display some text!

MPE/XL does things the right way -- it simply has an MPE command to do the job. Just say

```
ECHO HI THERE!
```

and that's it. The only thing I can complain about is the command name -- ECHO's pretty unintuitive. UNIX, of course, calls its command ECHO (along with calling its PURGE command RM and its text search command GREP), and MPE/XL borrowed the name. I'd rather HP called it DISPLAY or TYPE or OUTPUT or something like that, but it's hardly a big deal.

Of course, outputting variables and expressions can be easily done with the ECHO command -- just use the !xxx and ![xxx] syntaxes:

```
ECHO YOU'RE SIGNED ON AS !HPUSER.!HPACCOUNT, X = ![UPS(X)]
```

The only trick you need to know here is this: how do you output a string with leading blanks? Like all MPE commands, all blanks between the command name and the first parameter are skipped, so

```
ECHO HI THERE!
```

and

```
ECHO                 HI THERE!
```

produce exactly the same output -- "HI THERE!" with no leading blanks. Stumped? Just say

```
CALC "    HI THERE!"
```

The CALC command takes an expression parameter (in this case, just a string constant), evaluates it, and outputs the result. Since the parameters start at the quote, all the blanks between the quote and the HI are NOT ignored, and are output. (Be careful, though, of using the CALC command for general output purposes -- it works quite well for strings and booleans, but for integers it outputs more than just the integer's value.)

In addition to the :ECHO command for output, MPE/XL also has an input command, fortunately called :INPUT. For instance, you might have a UDC that says:

```
MOVE !FROMFILE, !TOFILE
SETJCW CIERROR=0
IF FINFO("!TOFILE",0) THEN
  COMMENT Target file already exists!
  INPUT PROMPT="OK to purge !TOFILE? "; NAME=PURGEFLAG
  IF UPS(STR(PURGEFLAG,1,1))="Y" THEN
    PURGE !TOFILE
  ENDIF
ENDIF
RENAME !FROMFILE, !TOFILE
```

If TOFILE already exists, the UDC will ask the user if it's OK to purge it. UPS(STR(PURGEFLAG,1,1)) merely means "the upshifted first character of PURGEFLAG" -- this way, Y, YES, and YOYO will all be accepted as a YES answer.

Actually, there's one pretty big temptation with the :INPUT command that should be resisted. You should think twice (or more) before using the :INPUT command to prompt for UDC (or command file) PARAMETERS. For instance, a UDC such as

```
MOVEP
INPUT PROMPT="From file? "; NAME=FROMFILE
INPUT PROMPT="To file? "; NAME=TOFILE
SETJCW CIERROR=0
IF FINFO("!TOFILE",0) THEN
  COMMENT Target file already exists!
  INPUT PROMPT="OK to purge !TOFILE? "; NAME=PURGEFLAG
  IF UPS(STR(PURGEFLAG,1,1))="Y" THEN
    PURGE !TOFILE
  ENDIF
ENDIF
RENAME !FROMFILE, !TOFILE
```

may not be a very good idea at all. Unlike the parameterized UDC we showed above, this one can only be conveniently used directly from the CI. Say that you want to write another UDC that runs a program and renames one of its output files (LISTFILE) into LISTFILE.ARCHIVE. With the parameterized MOVE UDC, we could say:

```
...
RUN MYPROG
MOVE LISTFILE, LISTFILE.ARCHIVE
...
```

and then have the MOVE UDC prompt the user if LISTFILE.ARCHIVE already exists. The unparameterized MOVEP UDC can't be used here at all, since it always prompts the user for the input and output files, which in this case are fixed and should not be prompted for.

In other words, this is the same reason why the best third-generation language procedures take their input values as parameters rather than prompt for them -- a parameterized procedure is much more reusable than a prompting one.

One very interesting use of the :INPUT command, though, might be in cases such as this:

```
MOVE !FROMFILE=" ", !TOFILE=" "
IF "!FROMFILE"=" " THEN
  INPUT PROMPT="From file? "; NAME=VARFROMFILE
ELSE
  SETVAR VARFROMFILE "!FROMFILE"
ENDIF
IF "!TOFILE"=" " THEN
  INPUT PROMPT="To file? "; NAME=VARTOFILE
ELSE
  SETVAR VARTOFILE "!TOFILE"
ENDIF
SETJCW CIERROR=0
IF FINFO("!VARTOFILE",0) THEN
  COMMENT Target file already exists!
  INPUT PROMPT="OK to purge !VARTOFILE? "; NAME=PURGEFLAG
  IF UPS(STR(PURGEFLAG,1,1))="Y" THEN
    PURGE !VARTOFILE
  ENDIF
ENDIF
RENAME !VARFROMFILE, !VARTOFILE
```

This UDC can accept its input either from its parameters or from the terminal. If it's used from within another UDC or by a knowledgeable user, it can be passed parameters -- if a novice user is using it, he can just type

```
:MOVE
```

and be prompted for all the input (for instance, if he's unfamiliar with what parameters the UDC takes). Actually, this may not be so useful for a simple UDC like this, but a really complicated UDC with many parameters can be made much more convenient with "dual-mode" processing like this.

There are plenty of other uses for the :INPUT command -- menus, error processing ("Abort UDC or continue? "), etc. There are also a lot of rather devious, non-obvious uses for it, too (more about those later). The only thing that bears keeping in mind is that :INPUTs should not entirely take the place of parameter passing.

#4. :WHILE LOOPS

No programming language is really complete without some sort of looping capability. In MPE/V, you could sometimes make do with the pseudo-looping capabilities of EDITOR/3000 (for things like taking the output of one program and translating it into input for another) and the ability of :STREAMs to stream other jobs. For instance, one thing that we at VESOFT used to make multiple production tapes was a tape-making job stream that at the end streamed itself, thus forming a sort of "infinite loop". (This was before we implemented :WHILE and other MPE/XL functions in our STREAMX Version 2.0, which makes things much easier.)

In one respect, MPE/XL's :WHILE command gives you all the looping that you need (any loop, including the FOR x:=y TO z and the REPEAT ... UNTIL constructs, can be emulated with a :WHILE); however, as we'll discuss later, it falls tantalizingly short in some areas.

First the good news:

```
SETVAR JOBNUM 138
WHILE JOBNUM<=174 DO
  ABORTJOB #J!JOBNUM
  SETVAR JOBNUM JOBNUM+1
ENDWHILE
```

This is an example of how the :WHILE loop can iterate through a set of integers. This simply aborts a whole range of jobs, from #J138 to #J174. (Seems useless? Try submitting fifty jobs in one shot -- all of them with the same silly error! I did this the day before I wrote the paper; the :WHILE loop sure came in handy.) Similar things can be done in some other cases -- for instance, you can use this to purge LOG####.PUB.SYS system log files IF you know the starting and ending log file numbers (unless you're willing to start at LOG0001 and work your way up).

Another example, taken roughly from Jeff Vance and John Korondy's excellent paper "DESIGN FEATURES OF THE MPE XL USER INTERFACE" (INTEREX Las Vegas 1987 Proceedings):

```
PRT F1, F2="", F3="", F4="", F5="", F6=""
COMMENT Prints F1, F2, F3, F4, F5, and F6 to the line printer
FILE OUT;DEV=LP
SETVAR I 1
SETVAR F7 ""            << to terminate the loop >>
WHILE '!"F!I"' <> ''
```

```
  IF FINFO('!"F!I"',0) THEN
    ECHO PRINTING !"F!I"
    PRINT !"F!I",*OUT
  ELSE
    ECHO ERROR: !F"!I" NOT FOUND, SKIPPED.
  ENDIF
  SETVAR I I+1
ENDWHILE
```

The WHILE loop here iterates through the 6 UDC parameters, making it unnecessary to repeat its contents once for each one. The construct !"F!I" is actually rather interesting. If I is 3, it gets translated into !"F3", which in turn gets replaced by the value of the F3 parameter.

Another example might be checking a parameter to make sure that it's, say, entirely alphabetic (in preparation for passing it to some program that will abort strangely and unpleasantly if there are any non-alphabetic characters in it):

```
SETVAR I 1
WHILE I<=LEN(PARM) AND UPS(STR(PARM,I,1))>="A" AND &
                       UPS(STR(PARM,I,1))<="Z" DO
  SETVAR I I+1
ENDWHILE
IF I>LEN(PARM) THEN
  COMMENT Hit the end of the string without finding a non-alpha
  RUN MYPROG;INFO="!PARM"
ELSE
  ECHO Error! Non-alphabetic character found:
  CALC "!PARM"
  SETVAR BLANKS ""
  SETVAR J 1
  WHILE J<I
    SETVAR BLANKS BLANKS+" "
    SETVAR J J+1
  ENDWHILE
  CALC BLANKS+"^"
ENDIF
```

Note the little "bell-and-whistle" -- if there's a non-alphabetic character, we use a :WHILE loop to concatenate together several blanks and an "^", so the output looks like:

```
Error! Non-alphabetic character found:
FOOBAR.XYZZY
      ^
```

Many parsing operations can actually be done more simply with the POS function (which finds the first occurence of one string in another); however, some complicated operations (such as the ones we just showed) may require :WHILE loops.

Finally, one other place where :WHILE should find a lot of use is the :INPUT command:

```
INPUT PROMPT="OK to proceed (Y/N)? "; NAME=ANSWER
WHILE UPS(ANSWER)<>"Y" AND UPS(ANSWER)<>"YES" AND &
      UPS(ANSWER)<>"N" AND UPS(ANSWER)<>"NO" DO
  ECHO Error: Expected YES or NO.
  INPUT PROMPT="OK to proceed (Y/N)? "; NAME=ANSWER
ENDWHILE
```

Most good UDCs and command files that use :INPUT should have some sort of input error checking, and this kind of :WHILE loop is a convenient way of doing it.

With all this power, what's there to complain about? After all, with an :IF and a :WHILE any language is theoretically complete -- any algorithm can be implemented.

Well, not quite. Control structures can get you only as far as the data access primitives are able to take you. Take some of the iterative operations that you'd REALLY want to implement:

* WHILE there are files in a fileset, DO something to them.

* WHILE there are jobs left, ABORT them (in preparation for a backup).

* WHILE there are records in a fileset, DO some processing on them -- perhaps write some of the records into another file, or pass them as input to some other program.

You can't do any of this (straightforwardly) because MPE/XL doesn't provide you any functions to read files, to handle filesets, to find all jobs, etc. You'd like to be able to say:

```
:WHILE FRECORD('MYFILE',RECNUM)<>''
...
:ENDWHILE
```

where FRECORD would return you a particular record of the specified file; unfortunately, no FRECORD primitive exists. The :WHILE command is only as powerful as the conditions you can specify; unfortunately, at the moment, this seems mostly limited to numeric iteration and to checking command success/failure.

Another thing you'd like to be able to do with :WHILE is to repeat a particular command every given number of seconds or minutes -- for instance, to have a job stream wait until a particular file is built or becomes accessible. Unless you're willing to spend lots of CPU time in the loop, you need to have some way of pausing for a given amount of time, e.g.

```
:WHILE NOT FINFO('MYFILE',0) DO
:   PAUSE 600    << 600 seconds >>
:ENDWHILE
```

Unfortunately, there is no :PAUSE command or PAUSE function provided by MPE/XL (although as we'll see shortly, there are some tricks you could do...).

#5. COMMAND FILES

Command files were implemented more for convenience than for additional power; however, they can be convenient indeed.

Simply put, a command file is a replacement for a UDC. If you want to implement a new command called DBSC to run DBSCHEMA, you used to have to write a UDC:

```
DBSC !TEXT="$STDIN", !LIST="$STDLIST"
FILE DBSTEXT=!TEXT
FILE DBLIST=!LIST
RUN DBSCHEMA.PUB.SYS;PARM=3
```

You'd add this UDC to your system UDC file, :SETCATALOG the file, and presto! you have a new command.

In MPE/XL, you could use a command file to do the same thing. You could build a file called DBSC.PUB.SYS that contains the text:

```
PARM !TEXT="$STDIN", !LIST="$STDLIST"
FILE DBSTEXT=!TEXT
FILE DBLIST=!LIST
RUN DBSCHEMA.PUB.SYS;PARM=3
```

Then, the very presence of the DBSC.PUB.SYS file will implement the new command -- no need to :SETCATALOG it. You can just say

```
DBSC MYSCHEMA, *LP
```

and MPE will check to see if DBSC.PUB.SYS exists, find that it does, and execute it much like it would have a :SETCATALOGed UDC.

Why is this so nice? Well, remember all the nonsense you had to go through to change a :SETCATALOGed UDC file? You had to build a new file with a different name, :SETCATALOG it in the old one's place, and even then it wouldn't take effect for another session until it logged off and logged back on! Most people ended up having several versions of the system UDC file, since you couldn't purge the old file until everybody who had been using it was logged off.

With command files, simply build the file, and there you have it. No need to worry about whether the UDC file is in use (unless the command is actually being executed at that very moment, it won't be in use); no need to choose a new name for the file; no need to remember to re-specify all the other UDC files on the :SETCATALOG.

In fact, the MPE/XL compiler commands are actually implemented this way -- :PASXL, for instance, is just a command file (PASXL.PUB.SYS) that sets up several file equations and runs PASCALXL.PUB.SYS (the actual compiler program file -- you still need programs for something!).

Whenever I give an example in this paper that involves UDCs, chances are very good that it will work with command files, too (in fact, you'd probably want to do it with command files). I only use UDCs in the examples to keep things as familiar as possible.

You could also implement account-wide commands by just putting the command files into your PUB group, and group-wide commands by putting them into your own groups. In fact, MPE/XL has a special variable called HPPATH that indicates where it is to search for command files; by default, HPPATH is set to "!HPGROUP,PUB,PUB.SYS", i.e. "search your group (!HPGROUP) first, then the PUB group, then the PUB.SYS group". You could actually change it to something else, e.g.

```
:SETVAR HPPATH "!HPGROUP,PUB,PUB.VESOFT,CMD.UTIL,PUB.SYS"
```

In fact, it's probably a good idea to keep your own command files not in PUB.SYS (where they'll just get lost among all the other files) but in a special group, say CMD.UTIL. This way, a simple

```
:LISTF @.CMD.UTIL
```

will show you all the system-wide command files that you've set up. Of course, you'll have to have a system-wide logon UDC that sets up the HPPATH variable to include CMD.UTIL.

A similar feature of MPE/XL is "implied run". Just entering a program file name will AUTOMATICALLY cause that program to be run; e.g.

```
:DBUTIL
```

will automatically do a

```
:RUN DBUTIL.PUB.SYS
```

WITHOUT your having to have a UDC or a command file for this purpose. You can also specify a parameter, which gets passed as the ;INFO= string to the program being run:

```
:MYPROG FOO
:PROG2 "TESTING ONE TWO THREE"
```

and also a second parameter, which gets passed as the ;PARM=:

```
:MYPROG ,10
:MYPROG FOOBAR,5
```

(Other parameters -- ;LIB=, ;STDIN=, ;STDLIST=, etc. can not be passed; you have to do a real :RUN for that.) Also note that MPE/XL looks for the program file in exactly the same places in which it looks for a command file: all those groups listed in the HPPATH variable.

These features are all very convenient, and can save you a good deal of effort and some typing. There is, however, one problem with both command files and implied :RUNs (and also UDCs) that limits their usefulness:

- THERE'S NO WAY FOR PASSING THE *ENTIRE REMAINDER OF THE COMMAND LINE* TO EITHER A COMMAND FILE, AN IMPLIED :RUN, OR A UDC.

For example, say that I want to implement a new command called :CHGUSER that executes my own CHGUSER.PUB.SYS command file. I want it to look much like MPE's :NEWUSER and :ALTUSER -- I'd like to let people say

```
:CHGUSER XYZZY;CAP=-BA,+DS,+PM;PASS=$RANDOM
```

The CHGUSER.PUB.SYS command file could then take the entire remainder of the line as a single parameter, and then perhaps pass it to some program that would process it.

Unfortunately, this simply can't be done! Since the parameter list includes ";"s, ","s, and "="s, MPE/XL views them as delimiters (it would view blanks as delimiters, too); there's no way of specifying in the command file that delimiter checking is to be turned OFF, and that the entire remainder of the command is to be passed as one parameter. Of course, you could require the user to enclose the parameter in quotes, but you'd rather not do that. (If you're thinking that declaring CAP=, PASS=, etc. as keywords to the command file will work, it won't -- look at the ","s in the CAP= parameter.)

In fact, MPE's own :FCOPY command couldn't be implemented as an auto-RUN or as a command file for this very reason -- each :FCOPY command always includes delimiters, and that won't work. I can see why HP doesn't like delimiters in an implied :RUN (so that the ;PARM= value can be specified as well as the ;INFO=), but why not have some sort of option for command files? Personally, I'd rather be able to pass the entire remainder of the command as one parameter than be able to specify a ;PARM= value.

In fact, UNIX does have a way of treating the parameter list (of either a program or a command file) as either a sequence of individual parameters or as one single string; UNIX programmers frequently use this feature. Again, this may be looking a gift horse in the mouth, but it would have been so easy for HP to implement something like this.

TRICKS

We've pretty much covered all the things you can do straightforwardly with MPE/XL. Of course, if this was all I had to say, I'd never have written this paper. People who know me know that I NEVER do things straightforwardly...

MPE/V had the (small) set of things you can do easily and the far larger set of things you could do if you really stood the system on its head. Similarly, MPE/XL has the larger set of things you can do easily, and the bigger still number of things you can do with a little bit of trickery. This is where the fun begins.

#1. PAUSING FOR X SECONDS

At a certain point in your job stream, a particular file may be in use. You don't want this to abort the job -- rather, you want the job to suspend until the file is no longer in use.

A first attempt at this might be:

```
WHILE FINFO('MYFILE',fileisinuseflag) DO
  PAUSE one minute
ENDWHILE
```

While the file is in use (surely there must be an FINFO option for this!), pause for a minute, and then check again. This shouldn't be too much of a load on the system (though without the :PAUSE this would be a heavy CPU hog indeed!).

Of course, you face two problems. First of all, there is no FINFO option to check to see if the file is in use or not. (OK, everybody, submit those SRs!) Old MPE programming hands, however, shouldn't despair:

```
FILE CHECKER=MYFILE;ACC=OUTKEEP;SAVE
SETJCW CIERROR=0
CONTINUE
PURGE *CHECKER
WHILE CIERROR<>0 DO
  PAUSE one minute
  SETJCW CIERROR=0
  CONTINUE
  PURGE *CHECKER
ENDWHILE
```

See what we're doing? The :FILE equation tells the file system to open the file with ;ACC=OUTKEEP (so the data won't get deleted) and close it with disposition ;SAVE (so the file itself won't get purged) -- the :PURGE command will thus not purge the file at all, but just try to open it with exclusive option. As long as the :PURGE is failing, we know that the file is in use (unless, of course, it doesn't exist or we're getting a security violation).

We do this check once before the :WHILE loop; then, if CIERROR<>0, we pause for a minute, do the check again, and keep going until the check succeeds.

The only problem that remains is, of course, that MPE/XL has no :PAUSE command -- without it, the entire exercise is academic.

What can we do? Well, one solution is to write a program. Call it PAUSE.PUB.SYS -- it'll take a ;PARM= value, convert it to a real number, and call the PAUSE intrinsic. Then, any of your command files could say

```
:RUN PAUSE.PUB.SYS;PARM=60
```

or just use the implied :RUN, as in

```
:PAUSE ,60
```

I don't like this. I don't like it for several reasons:

* The program, though not by any means difficult, is not trivial to write. If you know SPL, it's only a few lines; what if you only know COBOL? You can't even call the PAUSE intrinsic from COBOL (at least from COBOL '74), since COBOL can't handle real numbers (which the PAUSE intrinsic expects).

  From FORTRAN, you could call PAUSE, but you also need to call the GETINFO intrinsic (quick! do you know it's parameter sequence?). What if you had to write a program that checked to see if the file was in use? You'd have to call FOPEN, figure out the right foptions and aoptions bits (%1 and %100, if you're curious), and then use an intrinsic to set a JCW appropriately.

* Once you write it, you have to keep track of it. You put its object code into PAUSE.PUB.SYS -- where do you keep the source code? What if you lose it? Will you write documentation for it, or add a HELP option?

* Finally, the more external programs you use, the less self-contained the job stream will be. What if you move the job to one of your machines? You'll have to move the PAUSE program, too, and probably its source code and documentation, just to be safe.

  For vendors like VESOFT, the problem becomes even greater -- our installation job stream has to be able to run on a system where NONE of our software currently exists. We can't rely on your PAUSE.PUB.SYS or what have you.

You might agree with me or you might not. It's quite possible that the only problem with an external program file is that it somehow affects some silly esthetic sense of mine

-- that my mind is too twisted to appreciate a simple, straightforward solution. In any event, here's my answer to the problem:

```
:BUILD MSGFILE;TEMP;MSG
:FILE MSGFILE,OLDTEMP
:RUN FCOPY.PUB.SYS;STDIN=*MSGFILE;INFO=":INPUT DUMMY;WAIT=60"
```

Nice, eh? I build a temporary message file called MSGFILE, and then I run FCOPY with ;STDIN= redirected to it. Then, I tell FCOPY to execute an :INPUT command, telling it to WAIT for 60 seconds for input! (Of course, the only reason I use FCOPY here is to have it execute the MPE/XL command ":INPUT DUMMY;WAIT=60" -- FCOPY's convenient for this because we can pass the command to it as an INFO= string.)

Of course, the input will never come, since MSGFILE is empty; and, I must admit that the :INPUT ;WAIT= parameter was almost certainly intended to wait for TERMINAL input. However, it also works perfectly well when the input is coming from a $STDIN file that was redirected to a message file. When the 60 seconds are up, the :INPUT command will terminate and return control to FCOPY, which will then return back to the CI.

Now, our job stream is complete:

```
:BUILD MSGFILE;TEMP;MSG
:FILE MSGFILE,OLDTEMP
:FILE CHECKER=MYFILE;ACC=OUTKEEP;SAVE
:SETJCW CIERROR=0
:CONTINUE
:PURGE *CHECKER
:WHILE CIERROR<>0 DO
:  RUN FCOPY.PUB.SYS;STDIN=*MSGFILE;INFO=":INPUT DUMMY;WAIT=60"
:  SETJCW CIERROR=0
:  CONTINUE
:  PURGE *CHECKER
:ENDWHILE
```

Complete, of course, except for the many :COMMENTs that I'm sure that you, as a conscientious programmer, will certainly include...

Some may say that only a computer freak can think that the above solution is simpler than just running a program that loops doing FOPENs and PAUSEs.

They may be right.

#2. READING A FILE

The :REPORT command nicely shows you all the disc space used by each account on the system (actually, on MPE/XL 1.0 the disc space :REPORTed is sometimes erroneous, but I'm sure that'll be fixed soon). Unfortunately, it doesn't show you the total disc space used in the entire system, which is a useful piece of information. For instance, you might want to subtract the free and the used disc space counts from the total space on your discs, thus finding out how much lost space there is.

The :REPORT command can send its output to a file, which is good. But what can you do to read the file?

Well, let's start at the beginning. First, let's do a :REPORT into a disc file:

```
:FILE REPOUT;REC=-80,16,F,ASCII;NOCCTL;TEMP
:CONTINUE
:REPORT XXXXXXXX.@,*REPOUT
```

What's the XXXXXXXX.@ for? The :REPORT command usually outputs information on accounts and on groups; in our case, we don't want to have any group information at all. By specifying a group that we know doesn't exist in any account (I hope that you don't have a group called XXXXXXXX) we can make MPE output only the account information and no group information. It'll also print an error (NONEXISTENT GROUP), but that's OK.

Now, we have a temporary file called REPOUT, which contains two header lines and one line for each account. We'd like to extract the number of sectors used from each account line and add everything up. This is where the real trickery comes in.

One thing we might do is use EDITOR. The principle here is that we'll take the :REPORT listing, which looks like

```
ADMIN           15502       **     1046       **      8372      **
CUST             3062       **        0       **         0      **
DEV              7080       **       18       **         8      **
 ...
```

and "massage" it into a sequence of MPE/XL commands:

```
:SETVAR TOTALSPACE TOTALSPACE+    15502
:SETVAR TOTALSPACE TOTALSPACE+     3062
:SETVAR TOTALSPACE TOTALSPACE+     7080
...
```

We can then execute all these commands, and TOTALSPACE will be the total used disc space count.

Doing this is simple (?):

```
:PURGE REPOUT,TEMP
:FILE REPOUT;REC=-80,16,F,ASCII;NOCCTL;TEMP
:CONTINUE
:REPORT XXXXXXXX.@,*REPOUT
:SETVAR TOTALSPACE 0
:EDITOR
/TEXT REPOUT
/DELETE 1/2                << delete the header lines >>
/CHANGE 23/72,"",ALL       << delete everything right of the count >>
/CHANGE 1/8,":SETVAR TOTALSPACE TOTALSPACE+"  << delete the left >>
<< now, each line looks like: >>
<< :SETVAR TOTALSPACE TOTALSPACE+    15502 >>
/KEEP REPUSE,UNN
/USE REPUSE                << execute the :SETVARs >>
/EXIT
```

Now, the TOTALSPACE variable is set to the total disc space!

This is very much like what we did in pre-MPE/XL "MPE PROGRAMMING" -- we used EDITOR as a means of taking a program's or a command's output and making it another program's (in this case, also EDITOR's) input. In fact, UNIX's "sed" editor is very frequently used for this purpose by UNIX programmers (although it's much more adapted to this than EDITOR/3000 is).

The trouble with this solution is that it's inherently limited to plain textual substitution. What if we wanted to sum the disc space of all accounts that used more than 20,000 sectors? EDITOR has no command that can easily check the value of a particular field in a line. What we'd really like to do is use all the power of MPE/XL's :WHILE loop and expressions to process the :REPORT listing one line at a time.

As I mentioned before, MPE/XL unfortunately has no "get a record from a file" function. However, not all is lost.

Let's set up two command files. One (TOTSPACE) will look like this:

```
FILE REPOUT;REC=-80,16,F,ASCII;NOCCTL;TEMP
SETVAR OLDMSGFENCE HPMSGFENCE
SETVAR HPMSGFENCE 2
PURGE REPOUT,TEMP
CONTINUE
REPORT XXXXXXXX.@,*REPOUT
SETVAR HPMSGFENCE OLDMSGFENCE
FILE REPOUT,OLDTEMP
CONTINUE
RUN CI.PUB.SYS;PARM=3;INFO="TOTSPAC2";STDIN=*REPOUT;STDLIST=$NULL
ECHO TOTAL USED DISC SPACE = !TOTALSPACE
```

There are two new things here. One is

```
SETVAR OLDMSGFENCE HPMSGFENCE
SETVAR HPMSGFENCE 2
CONTINUE
REPORT XXXXXXXX.@,*REPOUT
SETVAR HPMSGFENCE OLDMSGFENCE
```

What's all this HPMSGFENCE stuff? Well, remember that the REPORT XXXXXXXX.@,*REPOUT command will almost certainly output an error message (NONEXISTENT GROUP). This is to be expected, and we don't want the user to have to see this.

So, we set the HPMSGFENCE variable to 2, indicating that error message are not to be displayed (setting it to 1 would inhibit display of warnings, but still print errors). However, since we want to reset HPMSGFENCE to its old value later, we save the old value of HPMSGFENCE, set the value to 1, do the command, and then reset the old value.

Personally, I think that this is a bit more effort than required. In MPEX, I simply added a new command called %NOMSG; saying

```
%NOMSG REPORT XXXXXXXX.@,*REPOUT
```

makes MPEX execute the :REPORT command without printing any messages. Similarly, HP could have had a :NOMSG command (for suppressing errors and warnings) and a :NOWARN command (for suppressing only warnings). This would have saved all the bother of the saving of the old HPMSGFENCE, setting it, and resetting it. In fact, to be really clean, I should even do a

```
:DELETEVAR OLDMSGFENCE
```

after doing the :SETVAR HPMSGFENCE OLDMSGFENCE.

In any case, the HPMSGFENCE solution is better than no solution at all -- in MPE/V, the warning message would always be displayed, and users might get quite confused by it.

The only other little trick (in this command file) is

```
RUN CI.PUB.SYS;PARM=3;INFO="TOTSPAC2";STDIN=*REPOUT;STDLIST=$NULL
```

What on earth does this mean?

In MPE/XL, the CI is not some special piece of code kept in the system SL. Rather, it's a normal program file called CI.PUB.SYS -- when a job or a session starts up, the system creates a new CI.PUB.SYS process on the job/session's behalf. However, CI.PUB.SYS is also :RUNable just like any other program; you can run it interactively by saying

```
:RUN CI.PUB.SYS
```

or just

```
:CI
```

Alternatively, you can run it and tell it to execute exactly one command:

```
:RUN CI.PUB.SYS;PARM=3;INFO="command to be executed"
```

(;PARM=3 tells the CI not to display the :WELCOME message and to only process the ;INFO= command, rather than prompt for more commands -- other ;PARM= values do different things.)

In our case, we're running CI.PUB.SYS with ;INFO="TOTSPAC2" (telling it to execute our TOTSPAC2 command file), and with ;STDIN= redirected to our :REPORT command output file. We redirect ;STDLIST= to $NULL, since the CI will otherwise echo its ;INFO= command -- ":TOTSPAC2" -- before executing it.

Now we can see what TOTSPAC2 contains:

```
:INPUT DUMMY            << to skip the first header line >>
:INPUT DUMMY            << to skip the second header line >>
:SETVAR TOTALSPACE 0
:SETVAR HPMSGFENCE 2    << to ignore any error messages >>
:WHILE TRUE DO          << loop until we get an error >>
:  INPUT REPORTLINE     << get a :REPORT detail line >>
   << extract the disc space -- 15 columns starting with >>
```

```
    << column 9 -- and add it to TOTALSPACE >>
 :  SETVAR TOTALSPACE TOTALSPACE + ![STR(REPORTLINE,9,15)]
 :ENDWHILE
```

See the trick? CI.PUB.SYS's ;STDIN= is redirected to a disc file, so all :INPUT commands will read from that disc file. For each line we read in, we extract the account disc space (STR(REPORTLINE,9,15)), and do a

```
 :SETVAR TOTALSPACE TOTALSPACE + extracted_account_disc_space
```

When we run out of input lines, the :INPUT command will get an EOF condition, and the command file will stop executing. TOTALSPACE is now set to the total disc space.

Both the EDITOR and the two-command-files solution can be used online, though both require two files (the first approach would require a disc file that contains all the required EDITOR commands). In a job, the EDITOR approach can be completely self-contained, since the EDITOR commands can just be put into the job stream; the second approach can also be self-contained if you create the TOTSPAC2 command file within the job (by using EDITOR or FCOPY).

Finally, one more variation on the same theme:

```
FILE REPOUT;REC=-248,,V,ASCII;NOCCTL;MSG;TEMP
SETVAR OLDMSGFENCE HPMSGFENCE
SETVAR HPMSGFENCE 2
CONTINUE
PURGE REPOUT,TEMP
REPORT XXXXXXXX.@,*REPOUT
SETVAR HPMSGFENCE OLDMSGFENCE
FILE REPOUT,OLDTEMP
CONTINUE
RUN CI.PUB.SYS;PARM=3;INFO="INPUT DUMMY";STDIN=*REPOUT;STDLIST=$NULL
RUN CI.PUB.SYS;PARM=3;INFO="INPUT DUMMY";STDIN=*REPOUT;STDLIST=$NULL
SETVAR TOTALSPACE 0
WHILE FINFO('*REPOUT',19)>0 DO
  RUN CI.PUB.SYS;PARM=3;INFO="INPUT REPORTLINE";STDIN=*REPOUT;&
                                                 STDLIST=$NULL
  SETVAR TOTALSPACE TOTALSPACE + ![STR(REPORTLINE,9,15)]
ENDWHILE
ECHO TOTAL USED DISC SPACE = !TOTALSPACE
```

Intuitively obvious, eh?

- The :REPORT command output is sent to a MESSAGE FILE.

- To read a line from the file, we say

  ```
  RUN CI.PUB.SYS;PARM=3;INFO="INPUT REPORTLINE";STDIN=*REPOUT;&
                                                STDLIST=$NULL
  ```

  This essentially tells the CI to read into REPORTLINE the first record from *REPOUT -- since it's a message file, the record will be read and deleted; the next read will read the next record.

- We loop while FINFO('*REPOUT',19) -- REPOUT's end of file -- is greater than 0. When the file is emptied out, we stop.

This is entirely self-contained, and in some respects more versatile (we can, for instance, prompt the user for input in the middle of the :WHILE loop, since our $STDIN is not redirected). The output-to-a-message-file and run-the-CI-to-get-each-record constructs are essentially a poor man's FREAD function. On the other hand, this approach runs CI.PUB.SYS once for each file -- even on a Spectrum this'll take some time!

One other glitch: each one of those :RUNs prints out one of those pesky "END OF PROGRAM" messages. In MPE/XL, you can actually avoid them -- as long as you use an implied :RUN rather than an explicit :RUN command. We can't use an implied :RUN because we need to redirect the STDIN and STDLIST. This is another good argument for using the two-command-file solution, which does only one :RUN and thus prints out only one END OF PROGRAM message.

#3. A PSCREEN COMMAND FILE

One of the most useful contributed programs for the HP 3000 is PSCREEN. (If you've been living in Katmandu for the past ten years, you might not know that it prints the contents of your screen to the line printer.) It works by outputting an ESCAPE-"d" sequence to the terminal, which causes almost any HP terminal to send back (as input) the contents of the current line on the screen. PSCREEN sends one ESCAPE-"d" for each line, picks up the output transmitted by the terminal, and prints it to the line printer.

Now, PSCREEN is already up and running, so there's really no reason to implement it as a command file; however, it's quite interesting to try it, both as an example of the power of MPE/XL and of the trickery you need to resort to in order to work around some restrictions on that power.

The process of reading the data from the terminal is actually quite straightforward:

```
CALC CHR(27)+'H'
WHILE there are more lines on the screen DO
  INPUT CURRENTLINE;PROMPT=![CHR(27)+"d"]
ENDWHILE
```

CHR(27) means a character with the ascii value 27 -- the escape character. "![CHR(27)+'d']" is the string ESCAPE-d, which when sent to the terminal (by the ;PROMPT=) will cause the terminal to input (into CURRENTLINE) the current line on the screen. The CALC command outputs ESCAPE-H (home up) to send the cursor to the top of the screen.

(Actually, it turns out that we can't just display the home up sequence in the :CALC since :CALC will then output a carriage return and line feed, and we'll skip the first line on the screen; instead, we have to incorporate the ESCAPE-H into the first :INPUT command prompt.)

The only twist here (one that the "real" PSCREEN has to deal with, too) is finding out how many lines there are on the screen. If we send an ESCAPE-d after we've already read the last data line, the terminal will just send us a blank line, and will be happy to do this forever.

There are two ways of solving this problem. One is to output (at the very beginning) some sort of "marker" to the terminal, e.g. "*** PSCREEN END OF MEMORY ***"; then, we can keep INPUTing until we get this marker line, at which point we know we're done. (We should also then erase the tag line so that subsequent PSCREENs won't run into it.)

Another solution is to ask the terminal itself. If we say

```
INPUT PROMPT="![CHR(27)+'F'+CHR(27)+'a']";NAME=CURSORPOS
```

then the terminal will be sent an ESCAPE-F (HOME DOWN, i.e. go to the end of memory) and an ESCAPE-a. The ESCAPE-a will ask it to transmit information on the current cursor position, in the format "!&a888c999R", where the "!" is an escape character, the "888" is the column number, and the "999" is the row number. This string will be input into the variable CURSORPOS. Then, the value of the expression

```
![STR(CURSORPOS,8,3)]
```

will be the row number of the bottom of the screen.

The old PSCREEN uses the first approach (write a marker), probably because it's more resilient; I suspect that some old terminal over some strange datacomm connection can't handle the ESCAPE-a sequence right.

In any event, reading the data from the screen isn't that hard. The question is: how can we output it to the printer?

As we showed in our previous discussion, it's quite hard to read data from a file into a variable. It's harder still to output the data from a variable to a file.

The solution lies in running CI.PUB.SYS with ;STDLIST= redirected, thus letting the :ECHO command output to a file rather than to the terminal. (This is much like doing file input by running CI.PUB.SYS with ;STDIN= redirected.) Here's what the full PSCREEN script actually looks like:

```
SETVAR PSCREENTERM "*** PSCREEN MARKER ***"
ECHO !PSCREENTERM
SETVAR PSCREENLINE 0
INPUT PSCREEN!PSCREENLINE;PROMPT="![CHR(27)+'H'+CHR(27)+'d']"
WHILE PSCREEN!PSCREENLINE <> PSCREENTERM DO
  SETVAR PSCREENLINE PSCREENLINE+1
  INPUT PSCREEN!PSCREENLINE;PROMPT="![CHR(27)+'d']"
ENDWHILE
CALC CHR(27)+"A"+CHR(27)+"K"    << clear the PSCREEN MARKER line >>
FILE PSCROUT;DEV=LP
RUN CI.PUB.SYS;PARM=3;INFO="PSCREENX";STDLIST=*PSCROUT
RESET PSCROUT
DELETEVAR PSCREEN@
```

Note that we're reading all the lines into variables called PSCREEN0, PSCREEN1, PSCREEN2, PSCREEN3, etc. These variables will then be read by the PSCREENX command file, which looks like this:

```
SETVAR PSCREENI 0
WHILE PSCREENI<PSCREENLINE DO
  ECHO ![PSCREEN!PSCREENI]
  SETVAR PSCREENI PSCREENI+1
ENDWHILE
```

There it is, in all its glory! Again, the PSCREEN program works just fine -- probably even better than these command files -- but this is just an example of the kind of things you can do.

One little glitch you'll run into with these command files is that the first line of every printout will read ":PSCREENX". That's because CI.PUB.SYS will echo its ;INFO= command to the ;STDLIST= file. For PSCREEN, this should be fairly harmless; however, what if you simply want to write the contents of a variable to a disc file without the echoing getting in the way?

The solution is this:

```
PURGE TEMPOUT,TEMP
BUILD TEMPOUT;NOCCTL;REC=-508,,V,ASCII;TEMP
FILE TEMPOUT,OLDTEMP;SHR;GMULTI;ACC=APPEND
RUN CI.PUB.SYS;INFO="ECHO !MYVAR";STDLIST=*TEMPOUT
FILE TEMPOUT,OLDTEMP
FILE DISCFILE;ACC=APPEND
PRINT *TEMPOUT;OUT=*DISCFILE;START=3
```

We run the CI and tell it to echo the variable MYVAR to a temporary file called TEMPOUT. Then we do a :PRINT command (a new feature of MPE/XL) that appends to DISCFILE the contents of TEMPOUT starting with record #3. Record #1 is CI.PUB.SYS's echo of the ":" prompt; record #2 is its echo of the "ECHO !MYVAR" command; record #3 is the actual contents MYVAR variable.

What a bother, and relatively slow, too (that's why we ran the CI only once in the PSCREEN script). A built-in MPE/XL FWRITE function would have been so much simpler...

#4. EXPRESSIONS AND PROGRAMS

One of the most interesting possibilities of the MPE/XL command interface has nothing to do with command files (or UDCs or job streams) at all. I've never seen it implemented before, so it might have a good deal of practical problems; however, I think that it has a lot of potential for power.

Consider a program that prints the contents of one of your specially-formatted data files. If it were a database, you could use QUERY, with its fairly sophisticated selection conditions -- you could specify exactly what records you want to select.

However, if you're writing a special custom-made program, how can you let the user specify the records to be selected? There are 1,000 records in the file (17 pages at 60 lines per page), and the user only wants a few of them. If you don't put in some sort of selection condition, the user won't be happy; if you put in the ability to select on one particular field, I'll bet you that the user will start asking for selection on another field. What about ANDs? ORs? Arithmetic expressions (SALARY<>BASERATE+BONUSRATE)? Soon they'll be asking for you to write your own expression parser!

What you really want is a GENERALIZED EXPRESSION PARSER, usable by any subsystem that wants to have user-specified selection conditions (and user-specified output formats). You could tell it about the variables that you have defined -- e.g., define one variable for each field in the file, plus some other variables for some calculated values that the user may find handy. Then, you tell it to evaluate a user-supplied expression.

Think of all the various programs that could use this!

* V/3000 could have used this for the input field validity checks (rather than having its own parser);

* QUERY could have used this for the >FIND command (rather than having its own parser, which, incidentally, can't handle parenthesized expressions);

* MPE/V could have used it for the :IF command logical expressions;

* LISTLOG could have used it to let you select log records;

* QUERY could have used it to output expression values in >REPORTs (rather than have that silly assembly-language-style register mechanism);

* EDITOR or FCOPY could have implemented a smart string search mechanism (find all line that contain "ABC" OR "DEF").

HP could have saved itself man-years of extra effort, while at the same time standardizing those expression evaluators that exist AND implementing expression evaluation in a lot of places that need it! Not to mention the uses that you and I could put it to!

The point here is that with MPE/XL you can -- in a way -- do this yourself. Take that file-reader-and-printer program of yours and prompt the user for a selection condition. Then, for each file record, use the HPCIPUTVAR intrinsic (or pass the COMMAND intrinsic a :SETVAR command) to set AN MPE/XL VARIABLE FOR EACH FIELD IN THE RECORD. Now, do a

```
:SETVAR SELECTIONRESULT expression_input_by_the_user
```

Finally, do an HPCIGETVAR to get the value of the SELECTIONRESULT variable; if it's TRUE, the record should be selected -- if it's FALSE, rejected.

In other words, you're using the :SETVAR commands expression handling to do the work for you. You set MPE/XL variables for all the fields in your record, and the user can then use those variables inside the selection condition. The condition can use all the MPE/XL functions -- =, <>, <, >, +, -, STR, POS, UPS, etc.; it can reference integer, string, or boolean variables. A sample run of the program might be:

```
:RUN SELFILE

SELFILE Version 1.5 -- this program prints selected records from
the PS010 KSAM file; please enter your selection condition:

   >UPS(STATUS)<>"XX" AND WORK_HOURS*HOURLY_SALARY>=10000
```

Meantime, the program is doing:

```
FOR each record from PS010 DO
  BEGIN
  :SETVAR STATUS value_of_status_field_file
  :SETVAR NAME value_of_name_field
  :SETVAR WORK_HOURS value_of_work_hours_field
  :SETVAR HOURLY_SALARY value_of_hourly_salary_field
  :SETVAR DEPARTMENT value_of_department_field
  ...
  :SETVAR SELECTIONRESULT &
          UPS(STATUS)<>"XX" AND WORK_HOURS*HOURLY_SALARY>=10000
  IF value of SELECTIONRESULT variable = TRUE THEN
    output the record;
  END;
```

(The :SETVAR commands in the pseudo-code should probably be calls to the HPCIPUTVAR intrinsic.)

There are several non-trivial problems with this approach:

- You're restricted to INTEGER, STRING, and BOOLEAN variables -- no dates, reals, etc.

- You're restricted to those functions that MPE/XL provides, which are rather limited (though fairly powerful).

- Most importantly, all those intrinsic calls will take some time! If you're reading through a 100,000 record file, you might encounter some serious performance problems.

As I said, to the best of my knowledge nobody's ever implemented this sort of facility -- for all I know, it may just not be practically feasible. However, I suspect that for quick-and-dirty query programs (and also input checking, output formatting, etc.) where performance is not a major consideration, it can be very powerful. You can use it to give a lot of control to the user, with very little programming effort on your own part.

CONCLUSION

The MPE/XL user interface is much more powerful and much more convenient than the "classic MPE" interface. (I didn't even mention some features, like multi-line :REDO, which are convenient indeed.) It lets you easily do many things that used to require a lot of effort; however, some key features are unfortunately missing.

Fortunately, with a little bit of ingenuity, even the apparently "impossible" can be achieved -- I'd be happy if all this paper did was let you know that there are possibilities to MPE/XL beyond those that are apparent at first glance. We HP programmers did some pretty amazing things with the limited capabilities that classic MPE offered us -- with MPE/XL, we should be able to write some very powerful stuff.

One thing that the new MPE/XL features should do is whet the appetites of all the poor people who still have to stick with MPE/V (or, heaven forbid, MPE/IV!) for some time in the future. After seeing all those wonderful things on the new machines, how can we bear to live with the old stuff?

There is actually a product out now (called Chameleon, from Taurus Software, Inc.) that implements MPE/XL functionality on MPE/V; VESOFT's own MPEX/3000 Version 2.0 release, tentatively scheduled for a June 1988 release, should do the same (in addition, of course, to all the other stuff that MPEX has always done -- fileset handling, %ALTFILE, new %LISTF modes, hook, etc.). MPEX Version 1.6 has, for the past year, already implemented the multi-line :REDO feature, both in MPEX, and in other programs, such as EDITOR, QUERY, TDP, QEDIT, etc.

VESOFT's STREAMX also implements many MPE/XL-like features (including variables, :WHILE loops, expressions, etc.) for job stream submission, an area unfortunately neglected by HP. Personally, I think that variable input, expression evaluation, input checking, etc. are even more useful at job stream SUBMISSION time than they are in session mode and at job stream execution time.

Finally, there are several other papers available about MPE/XL, all of which I can recommend highly. Jeff Vance & John Korondy of HP had the "Design Features of the MPE/XL User Interface" paper in the 1987 INTEREX Las Vegas proceedings; David T. Elward published the "Winning with MPE/XL" paper in the October and November 1988 HP Chronicles. Also, the MPE/XL Commands Manual actually has a lot of useful documentation on command files (including some very interesting MPE/XL Programming examples!) -- I've seen several versions, and it seems that the most recent ones have the most information. And, of course, the recently released "Beyond RISC!" book is an indispensable tool for anybody who deals or will be dealing with Precision Architecture machines.

Thanks to Rob Apgood of Strategic Systems, Inc. and Gavin Scott of American Data Industries for their input on this paper; thanks especially to Gavin for letting me test out all the examples on the computer in the two hours between the time I finished writing it and the time I had to Federal Express it up to BARUG.

Finally, any errors in this paper are NOT the fault of the author, but were rather caused by cosmic rays hitting the disc drives and modifying the data...

The role of Data Dictionaries in Application Development, with an Emphasis on System Dictionary.

Raymond Ouellette
Infocentre Corporation
3100 Cote Vertu
St. Laurent, Quebec
Canada H4R 2J8

The idea behind a data dictionary is easy to understand. You create a description of the data available on your computer system and store it on the computer itself. In this way, programmers (or programs) can find out exactly what information is supposed to be on the machine and where and how to find it. This results in less error caused by confusion over the format of the data and makes it possible for end users to create reports without prior knowledge of the structure of the data.

Unfortunately the realities of implementing a working data dictionary environment are much more complex than the basic idea would suggest. It is therefore advisable to understand the general principles of data dictionaries before closely looking at a particular product.

This discussion of data dictionaries is in three parts.

Firstly there is an analysis of the benefits possible when a dictionary is available and the type of information which needs to be stored in order to achieve these benefits.

This is followed by a description of the problems which must be considered before implementing a data dictionary in a realistic environment.

Finally the ways in which System Dictionary can address these subjects in the HP3000 environment are discussed.

Audience level of my abstract: 3+ years.

This paper would best fit in track 3.

Introduction

Data dictionaries provide a means by which we can manage information. Dictionaries are simply a tool that facilitate the management of data, and the conversion of data into corporate information.

An effective implementation of a data dictionary can help manage this critical corporate resource. An ineffective implementation will hinder more than it will help.

Implementing a data dictionary is a major undertaking. A great deal of analysis, design and planning is required to set standards and procedures regarding the role and use of the dictionary. A number of technical and operational issues must be addressed, such as: dictionary maintenance, security, version control, and deciding on the number of physical dictionaries to be implemented.

Thc strategy chosen may be different for each of several uses of the data dictionary. The dictionary may be called upon to serve different roles in application development, end user computing, in conjunction with purchased application packages as opposed to in-house systems.

Using Hewlett-Packard's *System Dictionary* as a point of reference, let's concentrate on application development, and examine an approach to effective dictionary implementation.

1.1 Objective of Data Dictionaries

Data dictionaries are generally used for a combination of the following functions.

1) Centralized documentation of the data and programs on a computer.

2) Storing physical file specifications and record layouts so that programs always address the data correctly.

3) Storing logical attributes of the data which can be used to generate programs automatically.

4) Storing descriptions of the data on a computer which enable end users to generate reports.

1.1.1 Documentation

In the old days, all good system analysts used to fill in special forms which defined the record layouts for all of the files in the systems they were designing. Each programmer was given a copy of any layouts which affected his/her programs so that there could be no chance for confusion. Of course, in practice, the analyst would change the layouts quite regularly with the result that each programmer would end up with a different version of the file specifications.

The idea of data dictionaries emerged as a method of using the computer to ensure that documentation is always up to date.

Data dictionaries are not, however, the only possible solution to the problem. One of the most useful features of modern database systems is that they usually include some form of self-documentation of their structure. For example, all the vital information concerning an IMAGE database can be found by running QUERY and using a simple command. To some extent this has delayed the necessity to introduce data dictionaries since storing information in a dictionary is rather pointless if the same information is easily available to everyone anyway.

However, it is unlikely that any database system will ever be able to contain enough information about itself to make a dictionary totally redundant. An important function of a data dictionary is to keep a record of which programs process the various databases and data elements on the system. This information is particularly vital for finding which programs will be affected by changes to a database but cannot be obtained directly without checking every program on the system indivually.

Whereas the definitions of individual entities in a system could theoretically always be stored within the entity itself, the only way to effectively document the relationships between the elements is in a dictionary. Unfortunately, this sort of information is very difficult to maintain correctly since it is not easy to ensure that all of the relationships are actually included in the dictionary. If there were some automatic way of verifying that the dictionary were complete, there would be no need for the dictionary in the first place!

1.1.2 Storing physical file specifications

The most immediate benefits of a data dictionary almost always come from the ability of programs to extract physical data attributes directly from the dictionary.

This fact was realised long ago when it became popular to establish COBOL COPY libraries containing the record layouts of the files on a system. Whenever a COBOL program was compiled the relevant layouts were extracted from the library by the compiler. This freed the programmer from worrying about the hand-written record layouts mentioned above.

The system worked very well but was obviously limited by the restriction to a single language and by having to recompile all programs every time the library changed. Actually compiling the programs was no real problem but remembering which programs needed to be compiled was much more difficult.

With a proper data dictionary it is possible to extract file specifications from the dictionary every time the program is run so that any changes in the dictionary are always reflected automatically without altering or recompiling programs. To anyone who has used COBOL with copy libraries on a large system this sounds marvelous but as we shall see run-time access to a dictionary introduces all sorts of other problems.

It is worth noting that database systems such as IMAGE have to some extent reduced the benefits of extracting physical attributes from a dictionary since the same information is readily available from the database itself. In fact, it would probably be true to suggest that a program or compiler should never obtain physical data attributes from a dictionary. In the ideal world a database should, at the very minimum, hold a complete description of its physical structure and all programs should use this description to get at the data.

1.1.3 Logical Data Definitions

As well as storing a description of how the data is physically held on a computer we can also include the logical attributes of the data such as standard heading text or edit masks for data elements in the dictionary.

It is very important to understand the difference between "physical" and "logical" attributes.

As an example, consider the description of an IMAGE item in a dictionary. The physical specification defines the length and type of the item. These are physical facts which can never be sensibly contradicted by any program. The logical attributes, however, are not absolute and can often be altered for an individual program without obtaining invalid results.

In some cases the logical attributes are merely suggestions for programmers who need not take any notice if they do not wish to. In other cases, the attributes may represent "standards" which a program must follow unless there are good reasons for not doing so.

Just as the physical attributes of a database can be included in the database itself, there is no reason why some of the logical attributes could not also be included. But whereas the physical attributes of a database are finite, the possible logical attributes are limitless and for this reason it is very difficult to devise database systems which can fully cope with logical attributes.

Physical attribute definitions can be safely extracted from a dictionary automatically during program development without necessarily informing the programmer what is happening, however for logical attributes, the programmer must be given the opportunity to ignore the attributes if he/she wishes.

In the context of application processing, three methods for extracting logical attributes from a dictionary are possible:

1) Run Time. The logical attributes are freshly calculated every time a program is run.

2) Compile Time. Alterations to logical attributes in the dictionary will take effect only when a program is recompiled.

3) Development Time. The logical attributes are copied into the programmers code automatically when the code is initially developed. Alterations to attributes are not reflected in existing programs unless the programs are actually altered.

In fact, it is not likely that we would wish for logical attributes to be decided by a production program at run time. The effects of changing the edit mask for an item might be disastrous on a report where the print positions had been carefully counted by the original programmer.

The same sort of problems also arise when attributes are extracted automatically at compile time. Programmers are forced to test their programs everytime they compile them even if they haven't actually changed the code.

1.1.4 Describing Data for End Users

Most application systems offer the user some helpful information about how to operate the system and what data presented on screens or in reports represents.

A user trying to work with a report generator is not usually so lucky. The report generator itself will of course be able to explain how to create reports but will not know anything about the data being inspected or what it means.

Users can only look in the dictionary to try to find out what there is in the databases available to them. It is quite possible that the physical and logical attributes described in the dictionary, together with the documentation intended for the computer department, will be enough to get the intelligent user started. However, the user needs information about the data which is of no use to the computer staff and will not exist in the dictionary unless it is put there specifically for this purpose.

1.2 Problems Associated With Data Dictionaries

Having looked at the uses of data dictionaries we now concentrate on the difficulties which inevitably arise when attempting to actually take advantage of the benefits.

These problems are described under the following headings.

1) Standardisation

2) Security

3) Conflicts between Applications

4) Version Control

5) Administration

6) Prototyping

7) System Performance/Reliability

8) Proliferation of Dictionaries

1.2.1 Standardisation

It is extremely unlikely that there will ever be a single standard for data dictionaries even for one particular computer. Even if a such a standard were to emerge, there would be features missing which someone would need to use.

Suppliers of 4GLs often provide their own data dictionary for use with their product and are reluctant to base their language around any possible "standard" dictionary which does not really fit their needs anyway.

The result is that there are often several varieties of dictionary on the same machine all containing the same information but in a different format. Often the dictionaries deteriorate from their role as a centre for data definition into files which have to be maintained simply because 4GLs will not run if they are not present.

1.2.2 Security

A data dictionary contains some extremely important information and a lot of people are going to be legitimately inspecting and altering its contents.

The result of someone accidentally changing something when he shouldn't can be extremely traumatic, especially if programs are accessing data definitions at run time.

A dictionary therefore requires a security system which permits people to access what is available to them but can stop anyone going where they are not allowed. In fact, the security requirements for a data dictionary are much more intricate than we would expect from a typical application system.

1.2.3 Conflicts Between Databases

On large computer installations where several different databases are present, it can be difficult to represent all the databases conveniently in a single dictionary.

It is possible that each database uses a different name for what is essentially the same thing or uses the same name for completely unrelated entities. Somebody has to take the time to sort out the conflicts and however this is achieved, the resulting dictionary is likely to be very confusing.

This problem is most likely to occur on sites which are trying to establish a dictionary after many years of working without one. If a dictionary is used from the start the need to avoid conflicts between databases can be turned into an advantage.

1.2.4 Versions

It would be nice if it were always possible to establish a dictionary which never changed once it had been set up. Unfortunately, computer systems usually develop even after they are "live" and the dictionaries must also change.

When a change is required in the dictionary it is necessary to create a new version or copy of the dictionary so that the amendments can be tested while the old version is still in use. Once the new version is tested and the relevant program alterations have been implemented and tested, the old dictionary can be replaced by the new copy at the same time as the new versions of the programs are moved into production.

Problems start when several unrelated amendments to a system are being implemented at the same time and each programmer makes his own version of the dictionary to test his changes. We have to be sure to control these test versions of the dictionary very carefully and ensure that when a test version becomes the production version, it actually includes all of the changes which may have been implemented since the copy was originally taken.

A possible solution to this is to only allow one test version to exist at any given time. However, this leads to a situation where there is always something in the test dictionary which is not actually working yet and so the dictionary can never go live. In order to actually get the dictionary into production new developments must be suspended until current work is completed.

1.2.5 Administration

Any data dictionary needs an administrator to keep control of the contents of the dictionary.

Without such control, a dictionary is liable to degenerate to the lowest state which is capable of supporting the 4GLs which use it. It will probably become cluttered with definitions which are not actually used but cannot be removed for fear that they are.

The administrators main job is to ensure that the contents of the dictionary are complete and correct. Given the diversity of information in the dictionary and the range of people who will use it, this is not a simple job and usually requires a dedicated (and expensive) individual.

1.2.6 Prototyping

Contrary to accepted opinion, using a data dictionary makes system prototyping very difficult if the designer has to enter definitions into the dictionary before he can get anything working. The modern approach to prototyping which involves the programmer and end user working together is strongly inhibited when the user cannot request immediate alterations and additions to data specifications.

For successful prototyping with a dictionary, we need a development system which can make changes to the dictionary immediately during the design process or can temporarily function independently of the dictionary during the prototyping phase.

Both these methods deviate from the standard approach of most 4GL systems which regard the data dictionary as a relatively static predefined source of information.

1.2.7 System Performance/Reliability

As we have seen there are an enormous number of potential users for a data dictionary ranging from production batch programs attempting to obtain the attributes of a file to end users trying to generate their own reports. At the same time we also require that the dictionary should have a sophisticated security system and be accessible in a friendly interactive fashion.

However good the software, there are bound to be problems with the speed and reliability of such a complicated system.

Since everyone on the computer is theoretically connected to the dictionary, there will be a bottleneck as all of the various programs atttempt to extract information. Worse still, a failure in the dictionary will bring the whole computer to a halt.

Careful consideration must also be given when taking backups of the dictionary or bringing a new version into use. These will be operations which may require every single user on the machine to stop working and, on larger sites, this may not be feasible.

Obviously, these problems can be reduced by ensuring that the data attributes are extracted at compile time so that production systems do not access the dictionary. Even so, program development and end user reporting systems will still be vulnerable to weaknesses in the dictionary software.

1.2.8 Prollferation of Dictionaries

A simple solution to the many of the inherent drawbacks of data dictionaries is to create several dictionaries rather than a single master dictionary.

Instead of keeping a centralized dictionary it may be better to provide a separate dictionary for each application on the computer. A special dictionary for the end user report generator would also normally be appropriate in this case.

Some control over the creation of dictionaries must be maintained because if things get out of hand, it is likely that there will be a separate dictionary for each program and each user on the system. Programmers may even keep dozens of versions of different dictionaries on tapes in their desks!

Once this happens, the basic advantages of centralized documentation are lost. There is no longer a single place which gives the correct definition of the data and although the individual dictionaries may be useful for the functions that they support the need for a "master" dictionary will inevitably arise.

2 SYSTEM DICTIONARY

System Dictionary is much more than a simple data dictionary since it is intended to be used for many purposes other than the storage of data definitions. We shall, however, concentrate on its role as a data dictionary in this paper.

2.1 The System Dictionary Database

System Dictionary is basically a database designed specifically to store information about a computer system. As its name suggests, System Dictionary is intended to be a central database and it is not expected that there will be vast numbers of dictionaries on a single machine. It may be that the proliferation effect will eventually overtake the initial aims but it is evident that many of the features of the dictionary are intended to prevent this happening.

For readers familiar with IMAGE, a brief comparison of an IMAGE database with System Dictionary is a good introduction.

The System dictionary contains 'entities' and 'relationships' which are very roughly equivalent to the master and detail records in an IMAGE database. The entities and relationships have 'attributes' which are like the fields of an IMAGE dataset.

The key to an IMAGE Master set may be any length or type, but access to an entity type in the System Dictionary is always achieved through a a 32 byte 'key'.

The names of relationship types in the dictionary are always formed from the names of the entity types which they relate. This is equivalent to suggesting that detail sets in an IMAGE database should contain the names of the master sets to which they are chained. For example, a detail set representing an order line on an invoice would be called something like "ORDER contains STOCK-ITEM" if it were transformed into an equivalent relationship type in a System Dictionary. This is more descriptive than "ORDER-LINE" (or even worse "ORDER-DETAIL") which is the name usually selected for this purpose.

Finally System Dictionary allows variable length attributes to be assigned to entities. Variable length fields are not supported by IMAGE and most modern relational databases but this capability is vital for databases which are to be used as a dictionary.

In summary the following expressions are roughly equivalent :

IMAGE	SYSTEM DICTIONARY
Master Dataset	Entity Type
Master Record	Entity
Detail Dataset	Relationship Type
Detail Record	Relationship
Field	Attribute

This new terminology may seem irritating but for once there is a good reason for introducing new jargon. Remember that the dictionary will be used to store data about data. We will have entity types called "RECORD" and "IMAGE-DATASET" etc. and we can at least avoid some confusion by using new words.

2.2 Features of System Dictionary

In order to make the System Dictionary database suitable for use as a dictionary several special features have been included which would not normally be expected in a more conventional database system.

2.2.1 Extensibilty

Unlike IMAGE, there is no schema for a System Dictionary. It is fully flexible so that entity types and attributes can be added or altered at any time. When a new dictionary is created (using the utility SDINIT) it always contains a standard set of entity and relationship types called the core set. You can then customize the dictionary for your own needs by adding new entity and relationship types or changing the attributes of existing types.

The motivation for providing this exstensibilty in the dictionary is to overcome the problem of standardisation which forces software suppliers to produce their own dictionaries. Since the System Dictionary can be customized, it is not necessary to rely on the original specifications of the core set and software suppliers can add new features to the standard dictionary if they wish.

In reality, there is still great pressure on everyone to conform to the accepted standards. It takes some courage to invent a completely new entity type especially if it is likely that several other people will do exactly the same thing but use a different name.

2.2.2 Programmatic Access

A command driven utility program is automatically supplied, to serve as a user interface to the dictionary. This interface can be augmented or replaced with user written programs that access the dictionary programmatically via a set of documented intrinsics. In this way, special purpose user interfaces or utility programs may be written with the capability to access the full set of dictionary functions. This enables dictionary users to create their own customized interfaces to the dictionary.

2.2.3 Security and Scopes

Any user or program which opens a dictionary must specify a SCOPE and the relevant password before access is granted. The scope name is like a user name which everyone has to provide when opening the dictionary in the same way that as everyone has to give a name before logging on to MPE. Entities and relations are always accessible to the scope which created them but can be secured against read or modify access by other scopes.

Note that security works at the entity level. This is equivalent to being able to secure individual records in an IMAGE database.

2.2.4 Domains

A single system dictionary may be split into several domains which are effectively "logical" dictionaries within the same physical dictionary. This is intended primarily for situations where many applications run on the same computer but have very little else in common. Each domain behaves as an individual dictionary and functions independently of the other domains which reside in the same physical dictionary.

The aim is to avoid needless conflict while still keeping all the domains under the control of the same dictionary.

All dictionaries initially contain a single domain called the common domain and new domains are created as required by the dictionary administrator.

2.2.5 Versions

System Dictionary recognises the need for different versions to exist at the same time and permits creation of many versions of each domain within a single dictionary. The versions are labelled "test", "production", or "archive" and the software prevents you from updating production or archive versions.

This method of creating versions within the same physical dictionary ensures that programmers cannot simply use COPY to create new versions at will but the administrator must still overcome the inherent problems of version control described earlier. System Dictionary allows several test versions to co-exist and he must always ensure that when a test version becomes a production version it includes any updates which have taken effect since it was originally created.

2.2.6 Aliases

An alias is an alternate name for an entity or relationship which is to be used instead of the actual name in a particular situation. The most common type of alias is the IMAGE-ALIAS for an entity which will be the name to be used when the entity appears in an IMAGE database.

Typically it is preferable not to use aliases but there are circumstances especially when databases have already been designed with no consideration for System Dictionary, when it is appropriate.

For example, we may have a database where item names have been prefixed for some reason so the item name for customer number is A100-001-CUS. This is not really a suitable name for the element in the dictionary and it would be better to call the element CUSTOMER-NUMBER and include the actual item name as the IMAGE-ALIAS.

2.2.7 Synonyms

A synonym is an alternate name for an entity which is used for a completely different purpose to the aliases. If the actual name of an entity is long, it is nice to be able to supply a short name which may be used by anyone who accesses the entity regularly.

If an element were called "INDIVIDUAL-CUSTOMER-NUMBER", the administrator might supply "ICN" as a synonym to save typing the full name. Any number of synonyms may be given to a single entity.

2.2.8 Internal and External Names

Every entity type and entity in the System Dictionary has an internal and an external name. Normally these names are the same but it is possible to alter the external names of entities or entity types.

When a program opens the System Dictionary, it specifies whether it will use the external or internal names to access the dictionary. By using internal names, the program can ensure that the names in the core set have not been changed. This mode is intended for standard software which will operate on many different sites.

Programs developed for a particular dictionary can use the local external names which will be known to the users of that particular dictionary only.

2.3 Representing IMAGE and MPE Files

Although the System Dictionary is totally flexible, the core set of entity and relationship types imposes standards on how the structure of IMAGE databases and MPE files should be represented.

The following list shows the most important entity and relationship types from the core set which are used for this purpose.

Entities:

ELEMENT
RECORD
IMAGE-DATASET
KSAMFILE
FILE
IMAGE-DATABASE

Relationships:

RECORD contains ELEMENT
IMAGE-DATASET contains RECORD
KSAMFILE contains RECORD
FILE contains RECORD
IMAGE-DATABASE contains IMAGE-DATASET
IMAGE-DATASET key ELEMENT
KSAMFILE key ELEMENT
IMAGE-DATASET IMAGE-DATABASE chains

2.3.1 Representing an Image Database

Each item in an IMAGE database corresponds to an entity with type ELEMENT in the System Dictionary. Although there is no single attribute which defines the IMAGE item type, the attributes COUNT, ELEMENT-TYPE and BYTE-LENGTH can be combined to form the IMAGE type. Other attributes of the elements such as DISPLAY-LENGTH, DECIMALS, EDIT-MASK etc. enhance the item specification beyond what is included in the IMAGE schema.

The datasets are represented by the entity type IMAGE-DATASET which has an attribute called IMAGE-DATASET-TYPE to specify whether the set is a master or a detail. It may seem reasonable to expect a relationship type called 'IMAGE-DATASET contains ELEMENT' to be used to assign elements to the datasets. In fact, we need to create another entity of type RECORD and the elements are associated with this entity using the relationship type 'RECORD contains ELEMENT'. The start position of each element within the record is indicated by an attribute of this relationship. The dataset is then linked to the record by establishing a relationship of type 'IMAGE-DATASET contains RECORD' between the dataset and the record.

The database itself is represented by an entity called IMAGE-DATABASE and the datasets are assigned to the database using the relationship type IMAGE-DATABASE contains IMAGE-DATASET.

Finally we need to indicate the keys and chains in the database. For master sets the relationship type 'IMAGE-DATASET key ELEMENT' is used to define one element which is the key to the dataset. The chains to a detail set are represented by a complicated relationship which links five entities and specifies the dataset, search element, sort element, master dataset and database involved in the chain.

There are other entity and relationship types in the core set concerning security classes and the devices used to store datasets but these are not described here.

You will notice that the method for representing a database is fairly flexible in that it permits us to assign datasets to more than one database and to assign records to several datasets. This could be useful in situations where two databases contain copies of the same dataset or where two datasets in a database have the same fields. It remains to be seen whether it will become common to take advantage of this flexibility or whether people will prefer to create a separate entity for each dataset.

2.4 Maintaining the System Dictionary

Several tools are available for maintaining the contents of the system dictionary.

Firstly there is a utility called SDMAIN which is a command driven tool for directly accessing the dictionary. There are commands for adding, amending or deleting entities and relationships as well as facilities for achieving administrative functions such as customisation or creating new versions and domains.

The big disadvantage with SDMAIN is that it is cumbersome to use and extremely unfriendly. It achieves for System Dictionary what QUERY does for IMAGE.

Another method of setting up a dictionary involves running a standard utility program called SDDBD which loads the format of an existing IMAGE database into a dictionary. The resulting definition only includes information which can be extracted directly from the database so logical attributes such as standard headings or edit masks are not loaded.

3.0 Utilizing the Dictionary In Application Development

Having identified the objectives of data dictionary use, and some of the common pitfalls lets examine the role a data dictionary can serve in an application development environment, using System Dictionary as a point of reference. When integrating application development with a data dictionary, we should be careful to capitalize on the strengths of the dictionary while avoiding its weaknesses, such that we use the tool effectively. In this context two features of System Dictionary are particularily intruiging: programmatic access, and extensiblity.

Programmatic access means there is a set of intrinsics making it possible for a program to open a dictionary, read information from it, update information and so on. This may not be of practical importance to many HP3000 shops, however it presents an opportunity to software suppliers to interface *application development* software with the data dictionary.

This opportunity is made even more attractive by the capability to extend the structure of the dictionary. System Dictionary can be customized to fit the needs of any application development effort. It does not have to be language, application, or vendor specific. The possibility of having one centralized system wide dictionary becomes a feasible reality.

To picture this opportunity envision your application development tools (text editor, flow charts and diagrams, COBOL generators, COPYLIBS etc.) being replaced by sophisticated application generator software. This software becomes the analysts' workbench, and it is actively involved in the definition of all application system processing; ie: data entry and inquiry screens, reports, batch processing, command procedures, security, and menus. The software also develops and stores the underlying file structures (IMAGE, KSAM, MPE), and generates source code for the application as a natural result of the development effort.

Application generator software represents the state of the art in fourth generation development tools. Along with tremendous increases in system development productivity, it also brings changes to the roles of application developers, end users, and the data dictionary.

With the application generator automating the detailed programming tasks, analysts are able to focus their energies on system analysis and design. Users can actively participate in the design phase, assisting the analyst in prototyping sessions made feasible by the capabilities of the software.

As one proceeds through the design and construction of an application, the system generator can be in constant communication with the data dictionary, offering lookups to existing entity definitions, as well as the opportunity to load new entities and relationships into the dictionary.

With this approach we create a symbiotic relationship between our application development tools and the data dictionary. When working with the application generator we retain full functionality of its native operating characteristics, but at the same time, avail ourselves to the centralized store of existing data documentation. The data dictionary can be maintained and updated automatically in a consistent fashion by the application generator.

In this role, the data dictionary is accessed at system development time, offering time saving assistance to the system developers. Existing data definitions can be extracted as programs are developed. Not to act as a constraint however, we can also create new entities for this application as needed. This is crucial in order for prototyping activities to succeed. When the development project is completed, the software can update the dictionary automatically, loading the definitions of any new entities created for the application. We can also load new relationships into the dictionary, identifying the data entities accessed by this application. These "where used" relationships will facilitate impact analyses required when changes to data definitions are contemplated.

The programmatic interface between the application generator and the dictionary can be implemented with the following objectives in mind:

1) The existence of the dictionary should be reasonably transparent to the average user. The dictionary is there to assist in the development effort - we should not be unduly tied to it or restricted by it.

2) Since the application generator holds all of the specifications of our application - both data and processing - it should be responsible for loading definitions into the dictionary.

3) The presence of the dictionary should not discourage or hinder application prototyping.

4) The application generator already maintains information about the application. There is no need to duplicate this information in the dictionary, unless it would be of use in the development of other applications.

3.1 Accessing System Dictionary

When contemplating how the dictionary can best serve our interests in application development, we must consider two general situations with respect to our data:

1) The application database(s) already exist, and are defined in the dictionary.

2) The application database(s) do not yet exist, and are not defined in the dictionary.

The term *database* as used above, refers to the aggregate of data processed by the application. This data may reside in one or more Image Databases, KSAM, or MPE files. The two general situations outlined above may be combined to formulate additional situations; eg: some of the files already exist, but some do not; of the files that do not yet physically exist, some of them are defined in the dictionary; etc.

How we approach these situations with respect to dictionary usage impacts on our development methodology.

If we adopt an approach that demands all data elements be defined in the dictionary prior to being referenced in the creation of an application program, then prototyping will be very effectively stifled. This approach requires a development methodology that begins with a rigorous definition of all of the application data. These definitions would be held in the dictionary and be accessible during program development. This latter activity of designing the application processing would need to proceed in a very predetermined fashion.

A development methodology centered around prototyping relies on the ability to draw on existing definitions, amend existing defintions, and create new definitions, of both data and processing, throughout the design phase. With the proper tools and expertise, this methodology can result in an application engineered to the customer's needs.

A flexible approach to interfacing with the dictionary can accomodate either development methodology. Accessing the dictionary can be approached this way:

1) Existing data definitions can be extracted from the dictionary as needed, during program development. This applies to data structures (Databases or files) that already exist, as well as new data structures that we need to create. Consider that for a new file or Database under development, individual field definitions or complete record layouts may already be defined in the dictionary, belonging to another file or database.

2) New data definitions can be created "on the fly", as needed. This is required in order to undertake effective prototyping. The new definitions would initially be stored locally, specific to this application. At some point in time, these new defintions can be uploaded to the dictionary.

Once the application is complete, it can be loaded into the System Dictionary so that a complete list of the entities processed by the application is available.

3.2 Customizing the Dictionary

A certain amount of customisation may be required to a System Dictionary for it to suit our application development purposes. This customization can be undertaken by using the "extensibility" feature, adding additional entities and relationships to the core set.

Firstly, we may have a need for additional logical attributes, describing individual data elements. The core set already provides several logical attributes such as DISPLAY-LENGTH, DECIMALS, and EDIT-MASK. Depending on your needs you may wish to extend this list with other attributes like MATCH-PATTERN for example.

Another useful attribute can be attached to a number of entities, marking them "Private" to this application. Elements or Records marked as "Private" would not be accessable to other applications. Once the development effort is completed, new entities created by the application can have their definitions marked "Public", and hence be available to other development projects.

A number of customized relationship types might also be in order, such as:

- SYSTEM processes ELEMENT
- SYSTEM processes RECORD
- SYSTEM processes IMAGE-DATASET
- SYSTEM processes IMAGE-DATABASE
- SYSTEM processes KSAMFILE
- SYSTEM processes FILE
- SYSTEM owns ELEMENT
- SYSTEM owns RECORD
- SYSTEM owns IMAGE-DATASET
- SYSTEM owns IMAGE-DATABASE.

The first group of relationship types are used to identify the entities which are processed by a particular application. The second group is used to indicate which application was responsible for creating the entities in the database. Later on, these relationships can be extracted from the dictionary, providing valuable data administration information.

3.3 Loading Applications into the System Dictionary

At any time during development of an application, it should be possible to load the application into the System Dictionary. Definitions of new data elements created for this application must be loaded, as well as a complete set of relationships, such as: which application created this element, which application processes it.

The loading process is very important. It constitutes the updating of a valuable corporate resource, and as such should be controlled under the data administration function. It makes sense then to approach the loading in a batch fashion, at the conclusion of the development project.

Having the application generator accomplish the loading of the dictionary is one of the more important benefits of integrating development tools with the dictionary. It ensures that the dictionary will be updated automatically, consistently, and accurately.

3.4 A Summary of the Dictionary Interface

The most important aspects of the dictionary interface can be summarized by the following points.

1) There is no run-time access to the dictionary. The interface is entirely in the application generator module.

2) The presence of the data dictionary does not impose a development methodology. One can choose to use the dictionary or not use it, draw existing defintions from the dictionary, create new definitions as needed.

3) Once an application is complete, the following information can be loaded automatically into the dictionary.

   i) A complete definition including logical attributes of the data accessed by the application.

   ii) Relationships which specify the databases, datasets, files and elements processed by the application.

Summary

A comprehensive dictionary product, such as Hewlett-Packard's System Dictionary, combined with sophisticated application development tools can go a long way toward solving a number of traditional dictionary implementation problems.

The features of System Dictionary make it possible to implement a single, system-wide dictionary database. (What a data dictionary was meant to be in the first place!) The dictionary administrator is provided with the necessary tools to address security and version control issues. Extensiblity facilitates standardisation - one dictionary can suit all (or most) purposes.

The new breed of fourth generation software can be interfaced to the dictionary such that we obtain the benefits of a centralized repository of data definitions, with few of the traditional problems. Prototyping is not stifled, the dictionary is accessed neither at run time nor at compile time, alleviating some system performance and reliability bottlenecks. The development software maintains the dictionary automatically, consistently, and accurately.

The data dictionary assumes an unobtrusive, yet immensely helpful role in application development.

Using MPE Message Files - An Applications Approach.

Patrick Fioravanti
Infocentre Corporation
7420 Airport Road
Suite 201
Mississauga, Ontario
Canada L4T 4E5

Message files are a feature of the MPE file system that permit two or more processes (programs) running concurrently to communicate with each other. Typically this Inter-Process communication is used to coordinate the activites of the two processes. In this light, the processing of an application task (Order Entry for example) can be distributed across a number of different programs, yielding useful benefits in an efficient manner.

This paper takes an applications approach to describing the purpose and functionality of Message files, rather than a hard core technical approach. It illustrates in layman terms how this under utilized feature of the file system can be incorporated into the design of many application systems. The discussion will be augmented with programming examples taken from an Order Entry Application developed in Speedware, to show how Message files are accessed and manipulated. Along the way some of the application design parameters that can be tweaked in order to maximize the benefits from a Message file implementation, will be discussed.

Audience level of my abstract: 1-3 years.

This paper would best fit in Track 3.

What are Message Files and why would one use them?

Message files are a feature of the MPE File System, available to HP3000 applications. They are a specific type of sequential file intended for applications that use *Inter Process Communication.* IPC is a mechanism whereby two programs running concurrently can pass information back and forth to each other. Message files were designed for ease of use and efficient communication between processes. They have some intriguing characteristics:

- FIFO Queues. Typically two or more programs will be accessing the message file concurrently. The program that Reads the file, will read the records in the order in which they were written to the file.

- Destructive Reads. As records are read from the message file, they are deleted. Note that the programmer can influence this with a call to FCONTROL, Control code 47.

- The MPE File System causes programs to wait until their Message File I/O is complete. *Readers* wait on a read request until there is a record to be read. *Writers* wait on a Write until there is room in the file to accomodate the record. Optionally the programmer can limit the wait to a pre-specified number of seconds.

- When reading, an EOF condition is returned by the File System when there are no records in the file **and** no *Write* processes have the file open. This characteristic may also be influenced by the programmer via a call to FCONTROL, Control code 45.

- Unidirectional flow. A program can Read from a message file, or Write to it but not both. Access is specified at file open time.

- Many concurrent Readers and Writers are permitted, although One Reader and one or many Writers is typical.

- Performant. Message files are partially memory resident, and partially disc based. Usually most I/O's are done in memory.

Given these characteristics of Message Files, some interesting applications come to mind. The nature of message file behaviour makes it feasible to distribute application processing across several processes, and coordinate the activities of those processes. Often it is the case that an On-Line task is *stream lined* (to make it run faster) leaving some *clean up* work for a subsequent batch task. Consider as an example an Order Entry application. The On-Line task is the entry of the Customer Orders. To prevent this process from being bogged down, usually the printing of the Order form or invoice is deferred to a nightly batch run. The entry of the order at the terminal along with the printing of the invoice constitutes the whole order process. It has been split into On-Line and nightly batch tasks in order to stream-line the on-line task. What if .... we still want the

On Line task streamlined, but we also want the Invoices printed as we continue entering orders? This can be accomplished by running the Batch Print Routine in the background as the terminals are entering the orders. As each order is entered, Inter Process Communication is required for the On-Line process to tell the Batch Process to *Print an invoice for the Order I just entered, and send the output to the printer located beside my desk.* An ideal application for Message Files.

Why use a Message File in the above Scenario? First off let's clarify how the Message file might be used. In the Data Entry routine, as each Order is Entered, a new record can be written to the Message file, identifying an order by its Order Number, and supplying additional processing instructions. There are multiple terminal sessions doing Order Entry, and each of these sessions have the Message file open for Write Access. The Print routine opens the Message file for Read Access. It waits in the background for a record to be written to the Message file. When a record comes in, the Print routine Extracts the Order information and formats and prints the invoice, then waits for the next Message file record. This Print Routine can be run On-Line or in Batch, in the BS, CS, DS, or ES Queues, the choice is yours.

A message file is well suited to this application because:

- The Print Routine will suspend on the Read, waiting for an order to print. It doesn't waste CPU seconds looping around looking for something to do.

- Since Message files are a FIFO Queue, Orders are printed in the order in which they were entered. Furthermore, the list of orders to be printed maintains itself, once a record is read from the message file it is also deleted from the file.

- The Print Routine will not receive an EOF condition from the File System until all records have been read, and no other processes have the file open for Write access. So when the terminal operators stop entering orders, and all the queued invoices have been printed, the Print Routine can be programmed to automatically terminate.

- Performance. The reading and writing of records to/from the message file will involve few if any Disc I/O's. Furthermore this approach requires the launching of only one process to print all of the day's invoices as they are entered. Other approaches might involve launching a separate process or job to print each invoice as it was entered.

How would we implement this solution in a Speedware Application? The first step is to define the message file. For this example, the message file is used to pass an Order Number from the Data Entry process to the Print process. Additionally we may want a general purpose Character field in the Message file layout for future use. Our Order Number is a J2 field, and we will tack on an eight character text field. Accordingly we need a Message file with a Logical Record Length of 12 characters.

```
:BUILD MSGFILE;REC=-12,,F,ASCII;DISC=100;MSG
```

The MPE BUILD command illustrated above will create the Message file. This is a typical BUILD command with the exception of the **;MSG** parameter, which of course tells MPE to create the file as a Message file. The file limit you specify is significant. Because of the nature of Message File operation, the file limit is set according to the number of records that the file must hold at any one time. Remember that records are deleted from the file as they are read. When setting the file capacity, ask yourself this question: Assuming that the On-Line processes can dump records into the file faster than the Print routine can read them, what is likely to be the biggest number of orders queued for printing at any time? This number (plus a bit more) should be your file limit. Temper your judgement with the understanding that Write Processes will be forced to wait for the Write to complete, if the Message file is full. In the above example, no Blocking Factor was set, and MPE will decide on one itself. The Blocking Factor for Message Files doesn't have a big impact on I/O performance since most or all of the Reads and Writes are done in memory. The Blocking Factor will influence the amount of Disc Space consumed by the Message file at BUILD time (along with other factors like Record Length, File Limit, and extents).

Once the Message file is built, it may be accessed from Speedware Applications. The Speedware Programmer will code a FILE Section to describe the Record Layout, and from there can read or write to the file using the FOR and CREATE commands.

For our example, we could code a File section like this:

```
FILE-MSG: (MSGFILE.DATA.INFOSYS)

 ORDER#   [1-4]    TYPE(J0);
 TEXT     [5-12];

EXIT;
```

Before running our application and accessing the Message file we need to supply some of the file Access Options to be used when the file is opened. We specify these options with an MPE FILE command. The FILE command for the *Writers* (On-Line processes doing the Data Entry) would look like this:

```
:FILE MSGFILE.DATA.INFOSYS;MSG;ACC=OUT;SHR;GMULTI
```

;ACC=OUT specifies that we will be writing (OUTPUT) records to the Message File. Remember that a given process can only have one type of Access to a Message File; either Read or Write.

;SHR means that we won't have exclusive access to the file, rather a number of concurrent processes can access the file.

;GMULTI means that the shared multiple access is GLOBAL. In other words, the multiple processes can belong to different jobs/sessions. If we specify **;MULTI** instead, then the multiple processes accessing the file must all have been spawned from the same terminal session or batch job.

For the Print Routine, a different FILE command is necessary, this one specifying *Read* Access:

```
     :FILE MSGFILE.DATA.INFOSYS;MSG;ACC=IN;SHR;GMULTI
```

Now let's address the programming details.

1) The Data Entry Screen.

Let's suppose that our application already contains a data entry screen used to enter new orders. Currently this Screen is comprised of two formats: the first one writes a record to the Order-Header Dataset, and the second format is concerned with the line items, maintained in the Order-Detail Dataset. Order entry is completed when the terminal operator completes both formats.

This Screen Program can be modified to write a record into the Message File at the conclusion of Data Entry for each order. A COMPUTE paragraph called from this screen in both Add and Modify modes can accomplish the task.

The COMPUTE code might look like this:

```
     COMPUTE-ADDMSG: AM;

     CREATE FILE:MSGFILE WITH
            ORDER-NO    = ORDER#,
            'PRINT IT'  = TEXT;

     EXIT;
```

2) The Print Routine.

Our application already has a Report Program that is used to print the invoices. This Report uses the Order Number as a key to access the Order Header and Order Detail files, and from there is able to directly access Customer and Product information. All of this data is extracted, sorted, then printed on the Invoice Form.

This Report program can be modified to be driven by the Message file. The coding of the Extraction Phase might look like this:

```
FOR FILE:MSGFILE          (* Read Message File *)
    BEGIN;

       FOR ORDER-HEADER.ORDER-NO<ORDER#>
           BEGIN;
              .  .  .
              .  .  .
              END;
       END;

SORT ON .  .  .  ;
```

As you can see, the records are read from the Message File as they come in. From there, the Extraction proceeds, beginning with the access of the Order-Header record identified by the key value contained in the Message File ORDER# field.

3) Coordination of the two processes.

We need to exercise some control over the activities of the Data Entry and Print processes so that they work in harmony. It makes sense for the Print Routine to execute in batch: that way it won't be dependent on a terminal session, and by default it will operate in the DS queue at a lower priority than the On-line sessions. Having decided that, we need automatic mechanisms in place to start the job when users begin entering orders, and to stop the job when the users are finished. As described earlier, the standard functioning of Message Files dictates that *Readers* waiting for a record will be supplied an EOF condition when the last of the *Writers* closes the Message File. It would seem that the MPE File System automatically provides us with the job shutdown mechanism, since an EOF on the Message File will terminate the FOR Loop driving the Report extraction phase. As we all know, things are never this simple. In addition to the automatic shutdown provided by an EOF condition, we should also have the means available within the application for the user to request the termination of the Print Routine. Additionally the Print Routine needs to have its Extraction Phase adjusted for it to be of much use. As it stands now, the Extraction Phase will not terminate until it reaches EOF on the Message File (ie: The end of the day when users have stopped entering orders), at which time the Sort and Print Phases will be activated. The end result is, no orders will be physically printed until the end of the day, and then they will all be printed. That's not what we want. Orders should be printed as they are entered, one at a time.

Let's address these *coordination* issues one at a time:

i) Automatically launching the Print Job.

The batch job that runs the Report must be executing when terminal operators are entering orders, otherwise records will just queue up in the Message File and nothing will be printed. We want the submission of the job to happen automatically, but we only want the job submitted if an operator is about to do some Order Entry and the job has not already been submitted. This is a job for an EXEC Procedure. As terminal operators make a menu selection to do Order Entry, they will be sent to an EXEC which will Stream the Print job if it's not already running, then send the operator into the Order Entry Screen. The EXEC could be coded like this:

```
MENU: "Order Entry", KEY("Order Entry"),
                      (EXEC-ORDERS, SCREEN-ORDERS),

EXEC-ORDERS:
!$IF DATA-PRINTER.DATA.INFOSYS  $CANCEL
!$*
!$* Please wait a moment while I initiate
!$* the Invoice Print Job.
!BUILD PRINTER;REC=-256,1,F,ASCII;DISC=1
!$GS STREAM-PRINT-INV
!$*
!$ASKR 1 \Okay, we're all set now, <cr> To Continue\
!EOJ
```

The EXEC needs to test a condition to know whether or not the job is already running. EXEC procedures can check many types of conditions, one of them is the existence of a disk file. So for this application we'll establish a convention whereby whenever the Print job is running there will be a file created called PRINTER.DATA.INFOSYS. Upon job completion, the file will be purged. Therefore, if the file PRINTER.DATA.INFOSYS exists, then we can say that the job is running.

ii) Terminating the Print Job.

The Print job must be set up in a such a way that it will print one invoice at a time, and be receptive to requests for termination.

We can use the Message File to send *commands* to the Batch routine in addition to *Order Numbers*. So if we wish to be able to terminate the Job from our On-Line Session, we can establish a convention whereby the value **STOP** in the *TEXT* field will command the routine to terminate. Let's change the Extraction Phase of the Report to look something like this:

```
CALCUL 'NO' = #EXTRACTED;
FOR  FILE:MSGFILE  BEGIN

    IF TEXT <> 'STOP'    (* Cue to terminate *)
       THEN BEGIN;
            CALCUL 'YES' = #EXTRACTED;
            DISPLAY ORDER#, $TIMES;
            FOR ORDER-HEADER.ORDER-NO<ORDER#>
                BEGIN;
                   .  .  .
                   .  .  .
                   END;
            END;
    BREAK;            (* Exit after Reading 1 Record *)
    END;

IF #EXTRACTED = 'NO'                (* EOF or STOP *)
   THEN BEGIN;
      COMMAND ('PURGE PRINTER.DATA.INFOSYS');
      DISPLAY 'I QUIT';
      END;

SORT ON .  .  .;
```

The BREAK Command limits the extraction phase to processing one Message file record. There are three possible outcomes:

- The Message File record will contain an Order Number. This Order will be extracted from the Database and the Invoice will be printed.

- The Message File record will contain the value **STOP** in the *TEXT* field. Accordingly, it won't extract an order for printing.

- An EOF condition will be returned by the File System. The Report will not extract an order for printing.

Based on the outcome, the flag #EXTRACTED will take on a value of YES or NO which conditionally triggers the COMMAND to purge the *PRINTER* file, and ultimately end the job.

The last missing piece is another EXEC procedure to control the operation of the Batch Report. This EXEC will establish a loop, continually actioning the Report which as we have seen will extract and print one order at a time. Based on the presence (or absence) of the *PRINTER* file, the EXEC will also control job termination. It looks something like this:

```
EXEC-BATCH:
!$TAG LOOP
!$IFF DATA-PRINTER.DATA.INFOSYS $GOTO EXIT
!$GS REPORT-PRINTINV
!$GOTO LOOP
!$TAG EXIT
!EOJ
```

Recalling that the Report uses the COMMAND Command to purge the Printer File upon receiving an EOF condition or **STOP** from the Message File, we can see that after the printing of each invoice we check for the presence of the *PRINTER* file. When the *PRINTER* file is purged, the job terminates.

A programming detail we have not covered yet is the On-Line transaction which sends the **STOP** command to the Message File. This would probably be set up as a menu selection restricted to the Order Entry Supervisor. Regardless of the menu security, the Menu Action would lead to an EXEC which checks for the existence of the *PRINTER* file. If it exists, then a Prompt Screen would be presented wherin the user would confirm the intent to stop the job, then write the appropriate record into the Message file. The Menu selection and associated processing is illustrated on the next page.

```
MENU: "Stop the Invoice Printing Job", EXEC-STOPINV;

EXEC-STOPINV:

!$IFF DATA-PRINTER.DATA.INFOSYS $GOTO OOOPS
!$PROMPT SCREEN-STOP
!$CANCEL
!$TAG OOOPS
!$* Oooops.  You wanted to stop the Printer Job, However
!$* my indications are that it is not running.  I cannot
!$* stop a job that is not executing.
!$* For your convenience I will show you a list of
!$* currently executing jobs:
!$PAUSE
!SHOWJOB JOB=@J
!$PAUSE
!EOJ

SCREEN-STOP: $PROMPT, A;

10,20, "Stop the JOB (Y,N)", REC[1-1], MATCH(Y,N),
       CALCUL A("REC[1-1] = #1"),
       COMPUTE-STOP;
END;

COMPUTE-STOP:A;

IF #1 = 'Y' THEN
   CREATE FILE:MSGFILE WITH
              0   = ORDER#,
          'STOP' = TEXT;

EXIT;
```

In operation this example works quite smoothly. Order Entry operators are able to perform their task without being aware that they are actively communicating with the batch print routine, with one exception: the first terminal operator (each day) is automatically taken through the EXEC procedure which submits the batch print routine. As they are generated, the invoices can be printed on any device including a departmental printer local to the data entry operators. The Invoices are printed in batch, at a lower priority than the On-Line processes so as not to impose a negative influence on terminal response times. The mechanism for initiating, feeding, and terminating the batch process is completely automated, and a manual override to stop the job is also available.

The design of this example involved the application of a Message File using one *Reader* (the batch Report) and multiple *Writers.* This configuration can be changed. For example, if there is a sufficient number of Order Entry terminals, orders may be entered faster than they can be printed. It may therefore be desirable to have multiple *Readers*; ie: more than one Batch report job feeding from the Message File. Another configuration may involve multiple Print jobs and multiple Message files, with each Print job dedicated to one Message File. This might be useful if there are several pools of data entry people located throughout the organization each with their own local printer. Each print job could send its invoices to a specific printer, and based on the terminal operator's User Security Environment each user would feed a specific Message File.

This is just one example of using MPE Message files in a Speedware application. Some other applications might be:

- Feeding transactions to another system. Many sites use Speedware to embellish or customize the Data Entry and reporting functions of a purchased application package. In these instances there is often the need to interface the Speedware system with the processing routines provided with the package. Message files could send transactions to a *Background* processing routine which would update the Application Database.

- Undertaking physical deletes. Speedware provides an optional Logical Delete mechanism which provides several benefits in an On-Line Environment. When one adopts a strategy of doing Logical Deletion of data, the need is presented for the periodic physical deletion of the data. Often we defer this to a nightly/weekly/monthly batch job, however the Physical deletion could be done by a background processor, as records are flagged for deletion On-Line. The benefit to this approach is that the physical delete procesing remains a batch task (On-Line performance benefit), yet the processing is self scheduling.

It is the intent of this paper to introduce the general concepts behind MPE Message Files and illustrate how they might be put to good use in a Speedware application. If more technical information is required, the following sources offer a starting point for your research:

1) Reactor Reference Manual, Infocentre Corporation,

Product Number RCT 5.00.00. A source for more detailed information regarding SCREEN formats and processing, FILE definitions, COMPUTE and REPORT processing.

2) MPE File System Reference Manual, Hewlett-Packard,

Part Number 30000-90236. Provides the detailed documentation of the Operating characteristics, features, and file system intrinsics for Message Files. See Chapter 8.

3) "Interprocess Communication Using MPE Message Files",

A technical paper submitted to the Detroit INTEREX Conference 1986, by Lars Borresen, Hewlett-Packard. Paper 3112 in the Conference Proceedings.

Distributed Application Processing and How to use it.
or
Stop Wasting those PC Mips!

Patrick Fioravanti
Infocentre Corporation
7420 Airport Road
Suite 201
Mississauga, Ontario
Canada L4T 4E5

As Fourth Generation software and data communications technology becomes more prevalent throughout the HP3000 community, the number of opportunities available for networking the mini with our PC's grows significantly.

With many organizations reaching and exceeding the computing capacity of their HP3000, the concept of redistributing the load, 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 HP3000 with personal computers. We will first introduce some of the concepts at work, then describe the various data communication topologies 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 processing concept based on the capabilities, strengths, and weaknesses of the various system components (both hardware and software) within the network.

Introduction

Historically, within the HP3000 environment we have seen the flow of corporate information managed almost exclusively by the central computing facility. Serious business applications have been developed and implemented on our HP3000s. With many organizations reaching and exceeding the computing capacity of their HP3000, the concept of redistributing the load, moving some of the application processing to the microcomputer becomes an attractive alternative. Pursuing this avenue has led a number of HP installations into the realm of distributed processing, where applications are developed that utilize PC hardware and software resources in addition to HP3000 based resources.

What are the perceived benefits of distributed processing?

The benefits associated with the integration of PCs with the current computing environment can be discussed under these headings:

* Offloading the central machine,
* Reducing costs,
* Reducing application downtime.

Offloading the central machine

Today the HP3000 is used typically by data processing professionals and end users alike. Programmers, analysts, database administrators define, develop, implement, and maintain production application systems. End users access these implemented systems, undertaking the application processing. In a lot of cases, the combination of system development and production activities strain the system resources, sometimes yielding unacceptable levels of performance during the peak processing hours of each business day. Something has to give.

Consider that each of the PCs sitting on desks in the user community as well as the DP department, has its own CPU coupled with significant storage and memory capacity. Aggregated, there is a lot of computing power in our PCs waiting to be harnessed effectively. Using PCs that are already cost justified, to potentially double the available computing power represents a significant opportunity to shift some of the current load away from the HP3000.

Reduced costs

A number of computing costs can be reduced by migrating some of the processing from the HP3000 to the PC. Hardware acquisition is one of these costs. It is not difficult to acquire a PC with a generous configuration for about the same price as a good quality display terminal. The same applies to software, as PC software is available for a fraction of the price of minicomputer software possessing similar functionality. Once the PC is established as an extension of the central facility, the incremental cost of hardware or software additions is substantially lower than on an HP3000.

Thirdly, we can look for decreases in communication costs. In cases where remote terminal workstations are replaced with PCs, the PC can execute the application locally. While doing so it may be possible to sever the link to the HP3000, thereby reducing data communication costs significantly.

Reduced application downtime.

With the PC comes the ability to collect local transactions on an attached disk system that can later be used to update the minicomputer. The PC then acts as a spare machine that can be pressed into service should the HP3000 be unavailable. Furthermore, a termporarily defective PC can be replaced far more easily than can an HP3000. With PCs, hardware redundancy becomes affordable.

Distributed processing - How?

If you accept that there are real benefits associated with PC integration, then we should examine the necessary tools, as well as some of the concepts behind them.

Ensuring success in distributed processing hinges largely on the compatiblity between the various machine environments. Application developers and users must be able to move easily between PC and HP3000 based processing. We can accomplish this by adopting an application development environment that operates on both machines. Specifically we require the same programming language, and the same database management system. A common application development environment provides two very important benefits:

- The application development staff can build PC applications with minimal retraining. All existing skills relative to the programming language and database design, creation, manipulation are transferable.

- Application development activities can be undertaken either on the PC or the HP3000 regardless of where the finished product will ultimately run.

There are a number of strategies for distributing application development and processing. A very simple strategy results in standalone applications being developed for processing on one or more PCs. This strategy is useful, but generally finds itself restricted to simple applications processing non critical data. More sophisticated strategies are required for applications that must be distributed across a number of machines. That is, the processing may be split between the HP3000 and PCs as well as the data. Implementing distributed applications in this fashion requires a communication facility - a means to transfer data between the two machine environments.

We need to set in place appropriate data security and access control procedures. In the past, with all application processing taking place on the HP3000, we have not felt the need to be concerned with these issues. MPE provides reasonable facilities for access control and data security. Operational procedures are already in place to ensure proper backup copies of data are taken on a regular basis. None of this security environment is automatically available on a PC. Rules need to be implemented that ensure PC resident data will be adequately protected. Furthermore PC application security is needed to control, as much as possible, access to the processing capabilities provided by the workstation.

With these tools and controls in place, we can turn our attention to developing and delivering distributed applications. These applications must be designed wisely, distributing the data, the processing, and fine tuning the various connections such that the application satisfies the users requirements and makes optimal use of all available computing resources.

The communication facility.

A communication facility is a central requirement for implementing distributed applications. The inter-machine communication enables the activities on the machines to remain coordinated. There are several ways to design the PC to HP3000 communication, lets examine first a batch approach.

Using standard PC - HP3000 file transfer utilities it is quite straightforward to implement a batched, bidirectional communication facility. This can be used during system development to transfer text source files between machines, and equally during system processing to transfer text or binary data files back and forth. With this approach we can design distributed applications that are *batch integrated*. This can be useful with applications that are based on processing cycles. During a processing cycle, transactions can be entered and captured on a PC. At the end of the cycle (nightly, weekly, monthly, etc.) the detailed transactions can be extracted from the PC database and uploaded to the HP3000 where they are posted to the central database. At the same time, updated versions of the reference or master files can be extracted from the HP3000 and downloaded to the PC(s). Software products are available that will work in conjunction with the file transfer utility to automate a scheduled transfer of files.

On the surface, it may not be immediately obvious how it is that this distributed processing approach provides benefits. Consider however that the processing involved with data editing and general transaction processing is offloaded from the HP3000. The PC earns its keep. For remote workstations, there can be savings realized in data communications costs. The workstation need not be connected all day, and when the connection is made, a concentrated stream of pre-edited transactions is efficiently transferred.

The batch integrated approach, although simple to implement, may not be suitable to all applications. It carries several disadvantages:

- The master files may be so large as to make their downloading impractical, if not impossible.

- The downloaded files are duplicated on multiple PCs. Duplication of corporate data profoundly complicates data and information management.

- Often the data entry workstation needs real time access to HP3000 data. Batch updates may not be sufficient.

Distributed applications requiring more timely access to large, sensitive corporate data files are better served by an *interactive* networking strategy. This more sophisticated form of communication results in a tighter, more cohesive integration strategy. Interactive networking enables an application running on a PC to instantly access information (read, write, or update) regardless of where that information resides, in a fashion that is transparent to the application and the user.

Interactive networking is accomplished using data communication software resident at each end of the connection. The PC resident component sends requests to the HP3000 whenever access is required to centralised data. The HP3000 resident component acts as a server to the PC. It waits for data access requests emanating from the PC, and responds to them. Typically the response involves formulating an Image access call, then sending the result of the call back to the PC. In general the type of service provided by the host resident software need not be restricted to accessing Image databases, however this seems to be a very useful service to offer a PC.

This arrangement is conceptually similar to accessing remote Image databases within a DS network of HP3000s. It would be simple enough to implement 'DS' type intrinsic calls within the PC resident software. These intrinsics could be invoked by the application whenever access is required to HP3000 resident datasets. Another approach involves defining the remote datasets within the schema of the PC database. A mechanism is required permitting the database designer to designate one or more datasets as being 'logical' datasets. They are part of the application, but physically reside on another machine. This approach carries three advantages over the 'DS' approach:

1) Accessing remote files is transparent to the application programs. The programs issue a read, write, or update request against a specific record or file. The Database Management System determines where the file physically resides, and hence what is required to satisfy the application request.

2) The local database knows the structure of the remote files since they are defined in the schema. Accordingly any interrogations of the remote dataset structure can be responded to locally, there is no need for remote communications to satisfy the request. This is tremendously beneficial for applications that make extensive use of the DBINFO intrinsic.

3) Within the PC database schema, the remote dataset definition can be a logical view of the actual dataset definition. The PC schema need only define the items of the remote dataset required by the application. Only the items defined in the PC schema will be transferred, thereby optimizing the data communications.

To visualize how this works, take as an example, a manufacturing application involving a large Parts master file. We can undertake transaction entry and processing on the PC, designating the Parts file as a logical dataset, physically resident on the HP3000. At run time, any "DBGETs" directed against the parts file will be trapped by the PC Database software, and sent via the communication link to the HP3000. The request is accepted by the network server, and formulated into a "DBGET" directed at the appropriate Image database. The result of the call (the record buffer if successful, otherwise the status array with the appropriate error information) is then transferred back to the PC where it is accepted by the Database software and returned to the application program. The same concept applies to other database access calls (writes, deletes, updates, locks).

Although simple in concept, the actual implementation of this communication facility can be quite complex, allowing for all of the possible data communication configurations, data compression techniques (for faster data transfer rates), and data encryption (for securing data as it is passing through the communication link).

The interactive networking facility permits the definition of physical (local) and logical (remote) datasets when defining the application database. A third type of dataset can be considered; a blend of local and remote. The implementation of this type of dataset provides a form of *Caching*. For example, if a request is received to read a record from our parts file, the DBMS looks for the record in the local dataset. If it does not exist there, it is retrieved from the HP3000 and written to the local dataset. The next time we wish to access the same record we may retrieve it without a remote access. Over time, the local dataset will be populated, on an 'as needed' basis, until it holds all of the master records required by this PC workstation. Although this implies data duplication, it means that at some point in time the communication link can be severed without service disruption. This concept is well suited to applications that operate according to Paretos' Law, where eighty percent of the processing is directed against twenty percent of the data records.

Regardless of the options chosen, the principle remains the same. By defining the structure and the location of application data within the database, the run time component of the DBMS is able to handle the data communication. Access to the remote files happens automatically in a fashion that is transparent to the programmer and the user. Distributed processing is made possible.

Connections

PCs using the communication facility are connected to the HP3000 in a standard fashion. Once connected, there are several means we can consider to optimize the connection.

At its simplest, the connection takes the form of a serial communication line from the PC to a terminal port on the HP3000. In reality, this connection might be direct, or pass through a very complex data communication network, including modems, multiplexors, and packet switching networks. Generally speaking, if we can connect the PC such that it can initiate a terminal session, then we can acomplish batch or interactive networking through the connection.

When using interactive networking, the PC can initiate a terminal session, and within the session activate the HP3000 resident portion of the communication software. This process might be activated automatically from the PC as part of the "DBOPEN" and similarily terminated automatically as part of the "DBCLOSE".

It may be considered wasteful (or perhaps excessive) to allocate a terminal port, MPE session, and network communication process to each PC in the application network. We can optimize this somewhat by dedicating resources to the application.

If it is known that a specific port or ports will be used only for interactive PC networking, then it becomes possible to launch a single communication server process that treats the ports as files, and waits to service requests emanating from the attached PCs. This strategy reduces the number of HP3000 processes (one per network instead of one per port), and the number of MPE sessions (the PCs don't initiate a terminal session), at the expense of dedicating the port(s) and scheduling the systematic initiation and termination of the network server.

Depending on the design of the application, there may not be sufficient volume of data traffic to warrant the allocation of one port per PC. In many HP3000 shops ports are expensive and scarce resources. In situations where specific ports are dedicated to interactive networking we can attach several PCs to one port and enhance the communication facility to channel the data messages separately for each PC sharing the port. This serves to optimize port utilization.

We can extend this concept further. If it is known that all of the PCs in a given network will be processing the same application, then the HP3000 based server can be customized for the application. By always having files and databases open, and other initialization type processing completed, the PC will never have the servicing of its request delayed unnecessarily.

These options pose a tradeoff: optimizing resource utilization versus flexibility.

Application Design - Resources

Having covered the concepts and some of the possibilities, let's concentrate on putting them into practice. By evaluating the various software and hardware components that we have at our disposal, we can develop application design guidelines that result in optimal use of the resources.

I/ Available pool of software:

HP3000	PC
MPE	MS-DOS
Image / File System	Image Clone
Development Language	Same Language
	Personal Applications

II/ Hardware:

HP3000	PC
Disk (lots, shared)	Disk (less, dedicated)
Memory (limited, expensive)	Memory (limited, cheap)
CPU (powerful, shared)	CPU (powerful, dedicated)
Peripherals (high speed)	Peripherals (few, low capacity)

III/ Data Communication Link:

- Data transfer rate	(slow)
- Expense	(varies with configuration)
- Reliability	(varies with configuration)

An evaluation of the respective strengths and weaknesses of the available resource pool shapes our application design guidelines.

On the software side, MPE provides more sophisticated file system and security facilities. Included with the MPE file system is automatic multi user capability. By equipping ourselves with the same application development environment on both machines, we make Image and our chosen development lanaguage generally available. The PCs have an edge with readily available, powerful and friendly personal application software that can be used for data analysis (spreadsheets), graphics, and text processing.

Of the available hardware resources, the PC complement is dedicated to a single user, while the HP3000 hardware is shared among several (many) competing users. Generally speaking, the hardware devices connected to the HP3000 are higher capacity and boast faster access speeds, although in reality they may service an individual user in a more sluggish fashion when heavily subscribed.

The data communication network is arguably the weakest link in the chain. Even the fastest of serial data transfer rates pale in comparison to the rate at which data is transferred from disk to memory inside an HP3000. Furthermore, the network is susceptible to unavailability due to a software or hardware failure of the host (HP3000), or failure of any component of the communication network.

Application Design - Guidelines

The remaining challenge is to design distributed applications wisely. In our wisdom we must distribute the processing as well as the data storage in a manner that most effectively utilizes the available resources. Based on our evaluation of resources we should:

- fully exploit the availability of PC hardware,
- offload personal data analysis tasks to the PC
- reduce dependence on overtaxed HP3000 resources.
- economize on interactive data transfers
- use the MPE file system when data must be shared, or data security/protection is critical.

This translates into:

1) Utilize the computing capability of the PC for things like data entry tasks. Data edits, validations, calculations, error handling, screen compilations and screen I/O are then offloaded from the HP3000. The data entry operator will appreciate the consistently crisp response that can be provided by the PC.

2) Distribute data such that files requiring shared access remain on the HP3000. These files can of course be accessed through the interactive communication network. For most applications, this guideline results in reference files (frequently the "Master" datasets) residing on the HP3000, while the transactions ("Detail" datasets) can be captured on the PC. Regular validations (lookups) to the reference files during data entry can be handled by the communication facility, the output of the data entry (the transactions) in many cases need not be communicated to the host. This strategy offloads a significant amount of disk I/O from the host and also minimizes traffic on the data communication network. Should the PC resident transactions be required on the HP3000, for batch reconciliation or reporting purposes, we can consider a batch transfer at off hours to accomodate this.

   Although on the surface it may seem advantageous to leave all the files on the HP3000, this strategy heavily taxes the communication network (the weak link), and in some cases floods the HP3000 with file access requests.

3) Design the application to be fault tolerant. Local processing should still be possible during temporary periods of host unavailability. This can be accomplished by making some data edits or validations optional (flag the transactions as requiring some sort of batch validation later on). When host unavailablity is predictable, an optional downloading of the master files (or some subset of them) can be done ahead of time, and the data entry programs instructed to access the local files. This is similar to the application caching concept that automatically downloads remote records as they are accessed, and permits disconnection from the HP3000 without service disruption.

4) Leave large volume batch oriented tasks on the HP3000. The host has the speed, power, and large volume peripheral devices to handle these tasks efficiently.

These points offer general application design guidelines that may prove useful as a starting position when conceptualizing the design of distributed applications.

Summary

The technology exists today making distributed processing a reality. For many HP3000 shops this constitutes a new frontier, presenting new opportunities to address traditional problems of computing resource allocation and utilization. It also adds a new dimension to application design considerations.

It is through a solid understanding of the underlying concepts, and available options that we can adopt and implement an effective distributed processing strategy that will help to achieve the stated goals of PC integration.