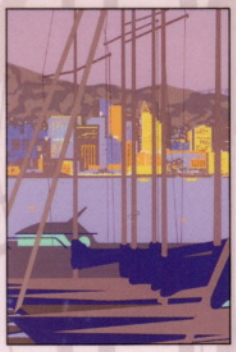
INTEREX



HP Users Conference

August
5-8, 1991
San Diego

PROCEEDINGS

VOLUME 1

MPE, MPE XL, O.A., Networking, Management

Sponsored by

INTEREX

The International Association of Hewlett-Packard Computer Users

Catch the Wave of HP Computing!

INTEREX

The International Association of Hewlett-Packard Computer Users

PROCEEDINGS

of the

1991 INTEREX HP Users Conference

MPE, MPE XL, O.A., Networking, Management

at

San Diego, California

August 5-8, 1991

Volume 1

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 Author

Index by Author

Index by Category

MPE/MPE XL SYSTEMS/APPLICATIONS

Index by Category

Index by Category

PC INTEGRATION

MANAGEMENT

Index by Category

NEW USER

TUTORIALS

Paper Number 3101

Surviving in a Multiple Protocol World

(The Sargassum Sea Frogfish and Networking)

Submitted By Jay W. Swearingen
The Apex Group
7151 Columbia Gateway Drive
Suite F
Columbia, MD 21046
(301)290-1606

There is a little-known realm of nature where seeing is not necessarily believing, for in it nothing is ever quite what it seems. It is a world peopled with innumerable animals that spend their lives in a carnival of make-believe, masquerading in a fantastic array of costumes. But there is nothing frivolous about this carnival world. For its inhabitants wear their disguises for only one purpose - to escape death.

James Poling

I. INTRODUCTION

Ever since the beginning of time, every animal that has ever existed has had the same problem. If animals are to survive they must confront their predators or avoid them. If it has some sort of strength - physical, chemical or armored - it can face up boldly to its enemies and defy them. If it is harmless, tasty, or soft-bodied, it must find some way of concealing itself.

The worst part of this threat is that there is never any escape, because no animal can hide from its predators indefinitely. Under duress it may flee and hide, but it can only do this occasionally. It has to come out of hiding at regular intervals to hunt its own food. The animal is forced to play a dual role - one of the hunter and the hunted most of its waking life. When not performing these life sustaining activities, it must render itself inconspicuous. It must conceal its form with some sort of camouflage.

In nature there are creatures on land and sea that disguise themselves as twigs, or leaves, or sponges, or any number of other objects of no appeal to a meal-seeking predator. A "dew drop" on a leaf that a passing bird ignores, for example, may be a tasty, edible beetle. Or a swaying "reed" in the bulrushes may on close inspection prove to be a long-necked bird deliberately imitating the breeze-blown reeds surrounding it.

The sargassum fish is named for its home, the Sargasso Sea. This is a large part of the Atlantic just west of the West Indies which is blanketed with a tangled mass of brownish seaweed, kept afloat by berrylike growths that are really air-filled bladders. While it is a fish, it is hard to believe, even when it is view up close. It is so well disguised with fleshy tassels, weedlike fibers, nodes, and bladder-shaped growths that it truly looks more like a clump of weeds than an animal.

The sargassum fish never leaves the protection of the Sargasso Sea's weedbeds because it has no need to. No predator can tell it from the weeds it hides in. Neither can the tiny fish it feeds

on. In fact, the fish it feeds on practically swim into its jaws. It has nearly lost the need to swim. It merely crawls through its weedbed home with the help of a pair of fins it uses as legs. Yet grotesque as it is, the fish is superbly disguised and admirably adapted to its environment.

Today in our Information Systems organizations, we have a similar need to blend and adapt in a complex and varied environment. Networks today no longer consist of single genre of all encompassing data communications implementations. In the past if you needed a network you merely called your computer vendor, and if they had one, they came and installed it. Today in its place, we have many kinds of networks at work in our organizations solving a wide range of problems. It is not uncommon to have network implementations from IBM, Microsoft, Novell, HP, DEC, and those that originated in the public domain to be operating in a single enterprise. There are many opinions on why this diversity in networks exists. Generally these thoughts can be grouped in the following categories:

* The failures of traditional MIS to respond efficiently, inexpensively, and rapidly to end-user requirements. User groups simply "built their own".

* Perpetuation of proprietary network strategies from individual equipment manufacturers (HP, DEC, IBM, etc.) that contributed to the failures above and IS ability to respond.

* The incredible price/performance opportunities of the PC and PC software versus traditional mini and mainframe computing, and

* The success and proliferation of departmental computing and the low barrier to entry of these networks.

Whatever the reasons, it is clear that today we are faced with a multi-vendor, multi-platform, multi-protocol, multi-standard world. Most organizations are grappling with the task of bringing a uniform commonality to these environments and allow the strengths of each individual computing environment to be used as needed for the tasks it is best suited for. For instance, a enterprise-wide electronic mail network, a accounting system, a departmental office automation server, and an executive information system may very well reside on different computing platforms. These platforms probably use different operating systems, protocols, and cabling types, and the workstations that sit on our users desk may very well be also totally different.

How we respond or adapt to these different environments is of great concern in most organizations. This paper will talk about the various methods available to IS or data communications managers for managing these problems. Like animals that are trying to survive in a difficult world, we must adapt to our environment and unite these disparate islands of network automation so that we may seamlessly *camouflage the differences* in these network implementations and *exploit their individual strengths*. Like the Sargassum Sea Frogfish we must disguise and adapt to our environment.

II. AN OVERVIEW OF PROTOCOLS

Why should I know more about protocols? I believe there are several good answers. First, we are moving towards a new model of computing called the Client/Server model. This model relies on a cooperative relationship between workstations (clients) and application processors (servers). We are moving towards this model because of:

1) Processing gets cheaper the closer we get to the processor. A MIP on a PC is significantly cheaper than a MIP on a mainframe.

2) Most organizations have failed to exploit the tremendous power that sits on users desk. Most of the time, this processing power far exceeds the power that sits in the computer room.

3) Organizations have a large investment in productivity and application software and hardware running their companies. The cost of replacing these investments in most cases would be prohibitive. Rather, it makes more sense to continue using these applications and application engines through a computer architecture that can bring them together in a uniform structure.

4) The advances made in relational database technology for managing, storing, and reporting information. Additionally, the sophistication of end user reporting tools that can access these databases is a breakthrough in ease-of-use and flexibility.

Client/Server computing demands a good understanding of the protocols available to you and the ones in use within your organization today. Secondly, as organizations downsize and the recession lessens, IS budgets are going to tighten. Most IS manager will be having to get more with less. This means knowing what you have and how you can get more out of it. A solid understanding of protocols is one area that will pay dividends. Finally, we are experiencing great improvement in the network management tools that are available to us. It is possible today to manage an entire enterprise network from a single workstation. Making correct decisions regarding network management tools requires a good knowledge of the protocols at play.

If managed properly, sophisticated multi-protocol networks can become a strategic weapon for most companies and increase our ability to respond to changes in the organizational structure in the ever-changing corporations in the U.S. Mergers, "spin-offs" and takeovers seem as common as ever. Our ability to adapt our network to meet these changes could be a real competitive advantage.

Before discussing the various ways of managing a multiprotocol environment, it is useful to review the most popular protocol stacks in use today. This list is in no way the "master" list but represents the vast majority of those protocols that are the most popular and identified by IS and data communications professionals as being in their current and future plans.

This section will focus on the following protocols:

TCP/IP (Transmission Control Protocol/Internet Protocol
OSI (the Open Systems Interconnect protocols)
SNA (IBM's System Network Architecture
DECNET (Digital Equipment Corporation network)
NetBEUI (IBM's NetBIOS Extended User Interface)
NetBios (Microsoft's and 3Com's NetBios Protocol)
XNS (Xerox Network System)
AppleTalk (Apple's Network Protocol)
HP Network Services (HP NS)
IPX (Novell's Internetwork Packet Exchange)

These ten protocols surely represent most of the protocols in use throughout the world. There are hundreds of individual protocols that are used by many organization worldwide. These individual protocols usually only define communications on a single network layer. For instance, X.25 the popular packet network access protocol specifies only three levels of connections. In this case it defines the physical level, link level, and packet level that roughly map to the fist three layers of the ISO/OSI model. When we talk of protocols like TCP/IP we are referring to full protocol suites. That is, TCP/IP, XNS and the others above, we are describing protocols that specify all or many layers (or the equivalent) of the ISO model.

A Quick Review of Networking Basics

A network is made up of hardware and software that connect computers and allow them to share resources and devices. Generally, we can describe a network in three different but interrelated components:

1. Physical connections such as coaxial and fiber optic cable, connectors, and interface hardware.

2. Software used to facilitate communications between various participating computers and peripherals. This software is built on rules and conventions that define proper operations. Protocols such as TCP/IP are and example of such rules.

3. Frameworks or structures that have been established for networking. These frameworks can be private or public and allow for the distribution of information throughout a department, a organization, or an enterprise. An example of such a structure could be Compuserve, the Federal Governments Internet, or the private network that connects the hundreds of Hewlett-Packard facilities world wide.

An example of these three network components in its simplest form could be a single ethernet cable that connects several PCs in a local area network and that use TCP/IP protocols. This network could be part of the larger Internet, an organization of wide area networks. The ethernet cable, TCP/IP, and the Internet represent all three components of a network.

Examining Network Protocols

When data communications take place, information is passed between two hosts in two different directions. Information is passed vertically across layers of network boundaries and laterally between corresponding peer layers according to rules know to those interacting layers. These rules, or protocols, make up the process interface between peer layers and permit hosts to communicate with each other.

As illustrated in Figure 1, protocol suites define many different layers. The layers between the *application* and *physical* layer are made up of software. Each succeeding software layer provides increasingly more complex services than the layer below it. Once you reach the *application layer*, an individual without any networking experience can make use of a network without any understanding of the underlying, lower layers.

To truly understand what makes up a protocol suite, it is useful to take apart a single protocol, layer by layer, and understand its inner workings. The other protocols suites we will discuss all have similar structures with a similar layered foundation. Perhaps the most talked about protocol suite that is predicted to be the most popular is the OSI Reference Model (RM). The OSI model defines communication processes into seven layers. Each layer can directly communicate only with adjacent layers, and indirectly only with its peer on another system. Each layer uses the services of the layer below and provides services to the layer above, depending on the direction of the transmission. Each layer manages its individual responsibilities and is aware of only the layer directly above and below it. However, each individual layer can still receive information from any other layer indirectly through adjacent network layers.

Physical Layer

This defines the actual medium chosen for data transmission. This could be 10BaseT, RS-232C and 802.5 token ring - all examples of the physical layer.

The *physical layer* converses with other peer *physical layers* and passes data to the *data link* layer.

Data Link Layer

Device drivers or interfaces that control actual hardware. A Network Interface Card in your PC operates at the *data link* Level.

This layer creates data frames out of raw bits that come over the "cable" into the respective host computer. These

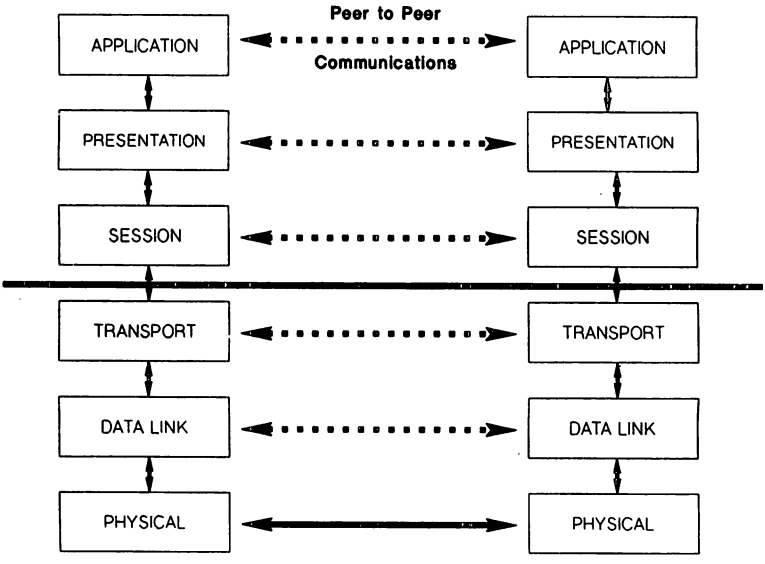


Figure I. Peer to Peer Communications

frames are also known as packets. Network bridges operate at the *data link* layer also known as the MAC layer.

Network Layer

The *network layer* is concerned with routing data over multiple hops from one system to another. If a packet is received, the *network layer* will decide if it was intended for a local or remote host. If it was intended for a local host it will send it up to the *transport layer*. If it was intended for a remote computer it will either be forwarded or discarded. Systems that are responsible for forwarding packets to remote hosts are called routers when they operate at the *network layer*.

Transport Layer

The *transport layer* is primarily concerned with the successful transmission of data in discrete transaction units called datagrams. It receives data from the *session layer* and break it up into packets so they can be used by the *network layer*. This layer is also concerned with flow control between peer hosts.

Devices called gateways operate at this level. A gateway can communicate with more than one network and can take a packet created on one network, translates it so another network can read it, and then sends it on to that network.

Session Layer

This layer manages system sessions between processes by starting, managing, and closing connections to remote hosts. Once a connection is established, an application process can do things like transfer a file, use a virtual terminal service, or perform further interprocess activities.

Presentation Layer

The *presentation layer* manages the representation of data between application processes. It works hand-in-hand with the local operating system (MS-DOS, UNIX, MPE, etc.) to translate the data it wants to send or receive. Through a process called negotiation, each *presentation layer* peer chose the syntax or "language" it will use for this translation.

Application Layer

This layer is at the very top of the OSI model. This layer is almost totally defined and includes programs that conduct virtual terminal, network file transfer, electronic mail transfer, and network management software.

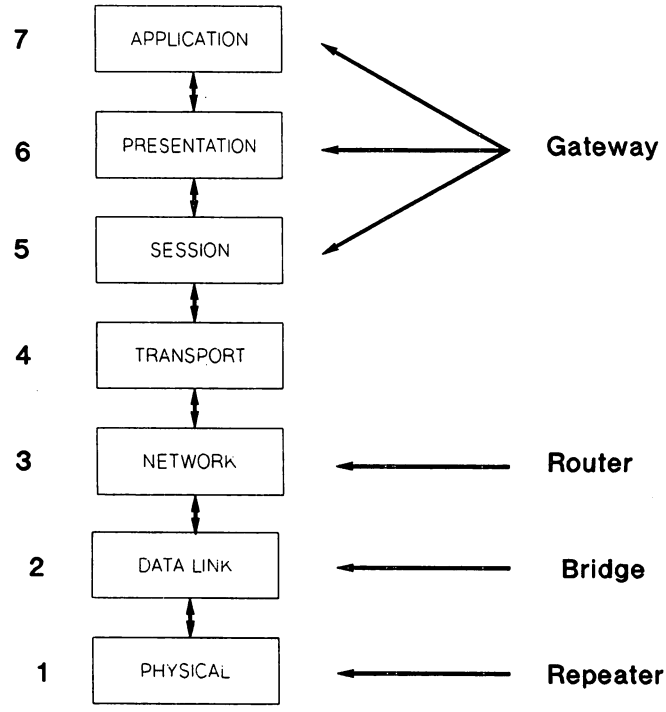


Figure II. OSI reference model layers and internetworking platforms.

A Protocol Summary

TCP/IP (Transmission Control Protocol/Internet Protocol

TCP/IP is a layered protocol suite developed for the U.S. Department of Defense (DOD) by the Advanced Research Projects Agency (ARPA) in the early 1970s. It has become the defacto standard protocol suite for UNIX platforms.

TCP/IP is characterized by its applications including Telnet (Virtual Terminal), FTP (File Transfer), SMTP (Simple Mail Transfer Protocol for E-Mail transfer), and SNMP (Simple Network Management Protocol for managing and reporting networks and network events). TCP/IP provides a somewhat painless migration into OSI in most implementations.

OSI (the Open Systems Interconnect protocols)

As described above, this International Standard Organization model is a hierarchical seven layer protocol suite. While this protocol is not totally implemented by most vendors, it has growing strength and momentum in the marketplace.

SNA (IBM's System Network Architecture

SNA is a network architecture intended to allow IBM customers to construct their own private networks, both host and subnet. It to is a hierarchical network structure allowing for peer-to-peer communications.

DECNET (Digital Equipment Corporation network)

DECNet is Digital Equipment Corporation's proprietary network implementation. It offers robust performance in part because it combines the equivalent of multiple OSI levels. This makes DECNet somewhat less modular than with more standard, open protocols.

NetBEUI (IBM's NetBios Extended User Interface)

NetBEUI is a fast, memory stingy, proprietary protocol for token ring networks developed by IBM. It is compatible with NetBios and is an integral part of IBM's SNA. NetBEUI is the standard protocol that is shipped with Microsoft's LANManager. NetBEUI (like NetBIOS), cannot be routed thus limiting it applicability in WAN applications.

NetBios (Microsoft's and 3Com's NetBios Protocol)

NetBios was developed by 3Com and provides efficient peer-to-peer communications via virtual circuits in both ethernet and token-ring networks. While NetBios does not fully address the entire OSI model, it does address the network, transport, and session layer. This makes it easy to interoperate with many other networking protocols. Like NetBEUI, NetBIOS is weak in wide are network functionality. NetBIOS is usually pared with other more robust WAN rich protocols when needed.

XNS (Xerox Network System)

Developed by Xerox, XNS is a widely used suite of protocols specifically designed for ethernet networks. While more efficient than TCP/IP, it is slower than NBP. Multiple versions of XNS sometimes creates compatibility problems.

AppleTalk (Apple's Network Protocol)

AppleTalk is a function rich protocol suite that operates at a relatively slow speed. It was designed for Macintosh users to access remote resources like printers and other shareable resources. Appletalk uses very little RAM and must have a bridge or gateway to access ethernet or token ring networks.

HP-NS (Hewlett-Packard's Network Services)

Back in the early 80's, HP was one of the first companies to have a very functional, true peer-to-peer network. Back when other companies were only talking about it, HP had virtual terminal, network file transfer, network peripheral access, and a well defined network API. The bad news was that it was extremely proprietary and did not interoperate with anything other than IBM mainframes using 2780/3780 type emulation.

Today NS is based mostly on industry standard protocols, primarily TCP/IP. It still uses proprietary software at the application level. For instance, virtual terminal on the HP 3000, called VT/3000, is still the main way that HP 3000's, Pcs and terminals communicate and is proprietary.

IPX (Novell Internetwork Packet Exchange)

Novell's IPX is a proprietary protocols suite based on XNS. It roughly corresponds to the OSI model and provide a good general purpose functionality. Like NetBIOS and NetBEIU it is very weak in wide area functionality. With NetWare 3.11 its Wide Area capabilities have improve.

These protocols make up the various animals in our kingdom. How we blend in and coexist

with these environments will be the subject of the next section.

III. A MULTIPLE PROTOCOL WORLD

How do we get our unlike networks to work together? There are many ways - some more conventional than other. This sections will concern itself with three main techniques: Shared Packet Drivers, Routers & Gateways, and Protocol Standardization. Each has its own advantages and disadvantages but the methods described are reliable, proven ways to integrate a multi-protocol environment.

Shared Packet Drivers

Shared Packet Drivers are a relatively new concept that have only recently become popular outside the academic community. It used to be that a desktop computer could only run a single protocol at a time. If we wanted to load a second or different protocol it usually required a "reboot." Some organization even resorted to the expensive option of installing multiple network interface cards in a single PC - one for 3Com and one for Novell and maybe even a IRMA card for IBM mainframe communications for example. Not only was this very expensive, but these implementations consumed huge amounts of memory and did not coexist with popular productivity software like Word Perfect and Lotus 123.

Shared Packet Drivers were developed to alleviate these problems while not paying a significant memory or performance penalty. These drivers allow multiple protocols to be loaded and unloaded on demand. For instance, an individual PC could connect to a Novell (IPX), Unix (TCP/IP), and HP 3000 connection (AdvanceNET) all from a single network connection and without rebooting their workstation. The three Shared Packet Driver implementations we will be discussing are the Microsoft/3Com NDIS, Clarkson Packet Driver Specification, and Novell's Open Datalink Interface (ODI).

Microsoft/3Com NDIS - The Network Driver Interface Specification (NDIS) is a standardized interface for OS/2 and DOS network platforms. NDIS drivers can be classified into two types: protocol drivers and Media Access Control (MAC) drivers. The MAC driver forms the bottom layer of the protocol stack and is the driver that controls the network hardware (network interface card). The remaining higher layers of the protocol stack are implemented in one or more protocol drivers (TCP, XNS, etc). The MAC drivers are typically provided by the manufacturer of the network interface board such as 3Com, Western Digital, or HP. These drivers can be used with any vendors NDIS-compliant protocol drivers such as SUNs NFS, FTPs TCP/IP for DOS, Microsoft's LAN Manager and many others.

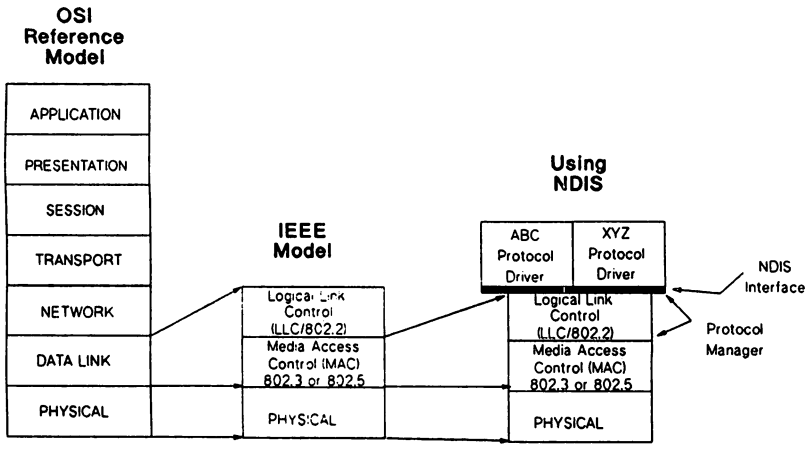


Figure III. NDIS – Dual Protocol, Single MACs

All NDIS drivers, both MAC and protocol, have a common structure. Each driver has an upper and lower boundary. The drivers are linked to form a stack by connecting, or *binding*, the upper boundary of one driver (MAC) to the lower level of the other driver (protocol driver). This binding can be repeat multiple time thus creating a separate protocol stacks using a single network interface card. The MAC driver at the bottom of the stack always is connected to the network hardware. NDIS also allows us to have two completely parallel stacks in one machine, each with its own adapter card and MAC driver, to implement two different protocols. This could be used to create a bridge with one protocol connected to two networks.

This driver implementation would allow a single workstation to run applications on many different systems using different protocols.

Packet Driver - A solution for easily "swapping" multiple protocol stacks in and out of desktops was first developed in 1981 at Clarkson University. They developed the Packet Driver Specification (PDS) for maintaining several protocols using a single network interface card (NIC). Recently TCP/IP software vendor FTP and Brigham Young University have been collaborating for providing a full range of multi-protocol management software. Clarkson has written compatible drivers for nearly all popular NICs including 3Com, IBM, BICC, ISOLAN, InterLan, and Western Digital. Both ethernet and token ring are supported under this specification. Other software vendors like The Wollongong Group also support the Packet Driver Specification. If Novell IPX is one of the protocols that you need to run, BYU will provide a modified Netware shell specifically designed to work with PDS.

While this solution provides an excellent way for operating multiple protocol stacks, some IS managers will feel uncomfortable operating software that resides in the public domain as does Clarkson PDS. This concern may be well founded given what is at stake in the corporate networks of today.

Novell ODI - A third solution from Novell is called the Open Datalink Interface. This interface supports multiple, simultaneous protocol stacks on Novell servers. It is only supported on NetWare 3.1 or greater and is somewhat of a Novell-centric approach.

Unlike NDIS, ODI does not just change the client software running in Novell nets. Rather it manages the multiple protocol connections at the server. This means that all network traffic must go through the Netware Server before leaving the Netware world into a different protocol.

With the introduction of 3.11, Novell has announced support for TCP/IP and Sun's NFS. They meld XNS (IPX is their version of XNS) and TCP/IP together in a manner that minimizes their differences, yet exploits their complimentary characteristics. To integrate TCP/IP with Netware, the operating system uses four Netware Loadable Modules (NLMs) which manage most of the multiprotocol tasking. Unfortunately, most network interface cards do not yet support ODI. You will either have to replace your NICs or put pressure on your vendor to write an ODI driver for the ones you have.

These three shared driver implementations make the concept of the "universal workstation" possible. That is, shared drivers could allow workstations to be configured similarly for multiple hardware and software platform access with minimum knowledge about the underlying network activity such as protocol or host interface.

When we add the a graphical user interface such as Windows 3.0 or NewWave, a common user interface can also be obtained. While most applications available today would not fully exploit the Windows environment, their is still significant power in the universal workstation concept. For instance, a individual workstation could logon to a variety of host, extract information and transfer it back using the Windows Clipboard or file transfer. This would allow for a "cut & paste" capability among all participating hosts.

Shared Packet Drivers do not address the coexistence of all protocol stacks. The next section will talk about gateway technologies as another method for managing multiple protocol environments.

Using Routers & Gateways in Multiple Protocol Environments

Another technique for blending unlike protocol stacks is the use of routers and gateways. These network components can be standalone black boxes, dedicated Pcs, or software and hardware that resides on a server platform. First a quick definition of Routers and Gateways:

Router - A single protocol router connects two networks running the same high-level protocol. Operating at the *network layer* routers possesses a higher level of intelligence than bridges. This makes them particularly suited for complex or large networks. Routers use a hierarchical addressing scheme that distinguishes between device addresses and network address. They allow the logical separation of an internetwork into many subnetworks. The subnetworks are logically independent administrative domains allowing distributed network management strategies to be enforced. Unlike bridges, routers do not impose constraints on network topology. They can accommodate any number of paths and active loops.

Routers use internal routing tables that includes network addresses and the devices on each network. Unlike bridges, they do not include the specific network address of each device. Where bridges make a simple forward or discard decision based on the routing table, routers use this information to select the best route for each packet. Routers also contain one or more algorithm that determines the number of hops between network segments and calculates the best or shortest path for network packets.

Gateways - These devices operate above the *network layer* and uses the *session, presentation, and application layers*. Typically gateways *translate* data into the format expected to be seen by the host it is destined for. That is, the function of a gateway is to convert packets from one protocol to another. To use and over-used metaphor, a gateway is analogous to a United Nations interpreter who can accept Chinese input and produce English output.

Routers In Action

A Multiprotocol router is simply a router that supports more than one protocol. It can provide software support for routing a number of high-level protocols simultaneously and within a single system. Multiprotocol routers eliminate the need for, and the expense of, having a router for every high-level protocol on the internetwork.

The differences between routers and gateways have become blurred over the past couple years. The emergence of "magic boxes" that can do bridging, routing, bridging & routing (brouting) and low level gateway functions have been the main cause. However, we typically think in terms of routers not doing any translation or protocol conversion. We also generally think in terms of two or more routers. That is routers only receive packets addressed to it by either an end station (source address) or another router. Based on the address of the final destination network and the information contained in the routing table, the router determines the next segment to which to send the packet, and the entire routing process occurs on this hop-to-hop basis.

A simple example of a multiprotocol router could be a multiprotocol router such as a those manufactured by cisco Systems. The cisco will route over 15 separate protocols from a single box with multiple interface cards. The packets can be DECNet, TCP/IP, Appletalk, XNS and others. The physical interfaces can be ethernet, token ring, AppleTalk or some wide area serial connection. The router acts as a traffic cop directing packets to their appointed destination based on its knowledge of the network. The router will not translate multiple high level protocols and allow them to interoperate.

Another multiprotocol router implementation would be a Appletalk to ethernet "black box". By placing additional client software on the MAC, data could travel over an Appletalk connection into the router and out over the ethernet for running TCP/IP applications. The client software simply encapsulates TCP/IP packets inside of Appletalk

packets, transports them to the Apple/TCP router, then strips them off leaving "virgin" TCP/IP packets for use by a compliant, participating host.

Gateways As Interpreters

A dedicated PC with gateway software that has an ethernet card running TCP/IP and a Token Ring Card running Novell IPX is an example of a simple gateway. The device is translating high level protocols into forms that the destination host is expecting to see. Users on the Novell network can transparently connect to host computers running TCP/IP on ethernet as though they were directly connected to that network. Another example of a gateway is HPs NS LAN Gateway. It provides personal computer integration capability from Pcs running Novell NetWare to LAN based HP 3000s. It is a dedicated PC running gateway software. Novell client Pcs install HP NS-User Services on their Pcs and it allows them to run HP DeskManager, access HP 3000 based peripherals, and conduct network file transfer.

Gateways can also be used for communicating to SNA environments, performing protocol conversion, and full translation to the IBM world. Products like the DCA/Microsoft Comm Server provides access to mainframe applications and convenient file transfer facility, no matter what physical links are needed. Consisting of two separate software components: a server component that runs on a server system, and a client application the runs on user Pcs, it provides full 3270 terminal emulation printer emulation, and file transfer. In this implementation, the network server does the majority of the work unlike traditional SNA Gateway implementations. This product provides the link to the IBM mainframe, the SNA software on the server, and client software for management of the applications.

Other examples of high level gateways are electronic mail gateways such as X.400 and SMTP gateways. These gateways provide message translation to and from unlike E-Mail Systems for translation and interoperability.

Standardization of Protocols

As Shared Packet Drivers and Routers & Gateways are applied to the opening analogy of adapting to ones environment as does various species of the animal kingdom, then the next technique is analogous to burning down the forest and choosing which animals will live in the new forest. An effective, but sometimes politically unacceptable, techniques is to run a single protocol stack on all host throughout your organization.

It is quite possible to run TCP/IP on every major hardware and software platform on the market. No other protocol stack even comes close. This includes IBM mainframes, Apple MACs, Digital VAXs, nearly everyone's Unix, DOS, OS/2, and others. As mentioned earlier, this whole business of multivendor networking was caused primarily

as a bottom-up phenomenon. Workgroup using AppleTalk, XNS, Novell IPX, and TCP/IP were initially isolated affairs with little need to get onto the corporate backbone and access the wide area. Today, however, business requires integration of LANs of all types regardless of the preferences of corporation's centralized planners.

Conceptually, if one could standardize on a single protocol, many of the enterprise-wide problems would go away. We would have no need for multiple protocol stacks in PC, no need for gateways, and no needs for different network emulation.

For instance, PC nets could run MS LANmanager, minicomputer servers running Unix could run LanManager/X, Mainframe computers could run MicroTempus' LanManager for MVS, DEC's Pathworks (LM for VMS) and they could all be connected via ethernet and TCP/IP. This would provide virtual terminal, network file transfer, resource sharing, electronic mail transfer, network management services, common application APIs for client/server computing, and much more.

If you can do it politically, salvage your investment in application software and hardware, and be able to pay for it, things would be great. You would also be well positioned to migrate your entire enterprise to OSI as it becomes more widely implemented.

IV. NOW WHAT?

By now I have probably only given you an adequate survey of techniques for tackling multivendor networking. Most of these techniques should apply to your organization, but probably not totally.

A Scenario

Let's take a fictional organization that is geographically distributed in San Francisco, Chicago, New York, and Denver. The organization if running financial and accounting software at corporate (New York) on an IBM 3090 running MVS. New York is also running a 200 node Novell network using token ring cards and OS/2 LANManager. San Francisco has a large VAX cluster running engineering applications and a Novell Netware network on a 486 server using 10BaseT ethernet cards. Chicago has a Marketing group with 50 node Appletalk network and Denver is operating a manufacturing facility using HP 3000s in a classical timesharing mode with terminals and a PC net using HP Resource Sharing also connected to the HP 3000.

The Requirement

This company has the following requirements for data communication:

New York Receive financial information from each of the locations

	Send and receive E-Mail from its NYC mainframe to all locations Receive, approve, and return promotion plans from Chicago
San Fran	Send Bill of Material files to Denver's HP 3000 Access product safety information from New York on OS/2 server Send Chicago product design information Pull customer service records from Denver
Denver	Send financial reporting to NYC mainframe Receive Bill of Material and Engineering enhancements from San Francisco Send and receive electronic mail to everyone
Chicago	Send and Receive E-Mail Send budget data to New York mainframe Get production statistic from Denver for quarterly and annual report

As illustrated in Figure IV, a solution could be implemented to provide the needed requirement and to allow unlike computer platforms to communicate. The specific design could include:

1) A high speed brouter network connecting all four cities. It is configured in a star configuration so that each city has an alternate path if network downtime is experienced. All protocols in use could be routed and those not routable could be bridged as necessary. The brouters talk Simple Network Management Protocol (SNMP) for proper network management as do most of the other network components.

2) New York - Full TCP/IP would be installed on the IBM 3090. This includes all TCP applications such as telnet, ftp, SNMP, and SMTP. The token ring network is connected through a token card in the FEP. LANManager stations could run dual stacks using NDIS. In this case they could run XNS for local LANManager functions and then dynamically switch to TCP/IP to talk to the mainframe, Appletalk, VAX, or HP hosts. 3270 users could continue to run E-Mail and terminals while the PCs could logon using TN3270 using TCP/IP.

3) Chicago - The Appletalk network could continue to use its native protocols for normal operations. Using a Apple/TCP/IP gateway, they could logon to the mainframe using TN3270 for E-Mail, use FTP to pull data from Denver and transfer data to NYC as needed.

4) San Francisco - By installing TCP/IP on the VAX cluster, we could logon on to the product safety database in NYC, transfer bill of materials and engineering files to Denver, and send new product designs to Chicago. The Novell network could have its own E-Mail system with a SMTP gateway (shipped with 3.11) for

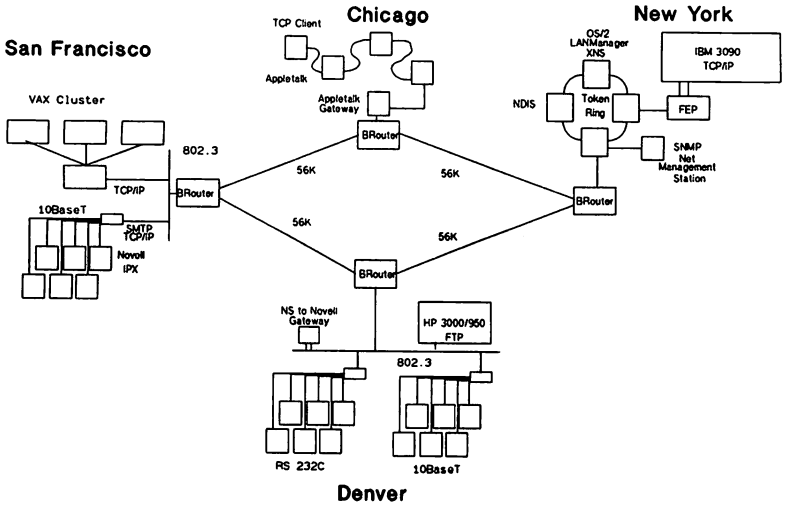


Figure IV. Fictional Enterprise Network

interacting with the mainframe E-Mail system. Using ODI, we could also access any other TCP/IP implemented host for file transfer and virtual terminal.

5) Denver - Novell communications take place through a HP NS gateway. This allows all Netware connected Pcs in San Francisco to access customer service records on the HP Resource file system. FTP is installed on the HP 3000 for transferring financial data to New York and bill of materials from San Francisco. PC based TN3270 software is installed on the Denver Pcs for accessing the electronic mail system in New York.

V. CONCLUSION

There are as many different ways to solve this problem as there are network engineers. The important thing is that there are very good ways to make a bunch of different protocols talk together within an organization. Rather than fighting the differences in these protocols, it is usually better to exploit their strengths and leverage off of the significant hardware and software investment your organization has made. This is not only smart business but will position your organization for tightening IS budgets and the emerging client/sever model of computing.

Like the Sargassum Sea Frogfish, we should adapt to our environment and blend into it seamlessly. In this way we can keep "afloat", well disguised with out anyone even knowing we are different.

Thank You.

BIBLIOGRAPHY

Allers, Rex, "Tech Talk - NDIS Concepts", 3Tech, Winter 1991, pg. 5-16.

Barry, David, "Running the Plays: Linking LANS in Mixed-Protocol Environments", Connect, Winter 1991, pg. 30-32.

Bradner, Scott O., "Testing Multiprotocol Routers: How Fast is Fast Enough", Data Communications, pg. 70-86.

Clark, Donald F., "The Protocol Playbook", Connect, Winter 1991, pg. 26-28.

Day, Michael, "Netware 3.11 Successfully Combines TCP/IP and XNS", LAN Times, April 15, 1991, pg. 53-56.

Dortch, Michael, "A Protocols Chalk Talk", Connect, Winter 1991, pg. 25.

Harrison, Brad, "Magic Boxes with Multi-Standard Support Emerge", LAN Computing, February 29, 1991, pg. 1, 25.

King Steven, "Multiport Bridges: A New Architecture for Ethernet", Data Communications, August 1990, pg. 72-75.

Knack, Kella, "UNIX International's OSA Will Become SVR4 Development Base", LAN Computing, February 29, 1991, pg. 1, 25.

Liebing, Edward, "Netware 3.11 Close to Its Promise", LAN Times, April 15, 1991, pg. 1, 95.

Morris, Desmond, Animal Watching, Crown Publishers, Inc., New York, 1990.

Morrow, Monique, "Charting a Path Through the Protocol Maze", LAN Technology, December 1990, pg. 79-82.

Poling, James, Animals in Disguise, W.W. Norton & Company, New York, 1966.

Roman, Bob, "Making the Big Connections: Implementing Internetworks with Bridges, Routers, and Brouters", 3Tech, Summer 1990, pg. 14-25.

Sabo, L. Michael, Interworking LANs Through WANs, McGraw-Hill, 1990.

Tannenbaum, Andrew S., Computer Networks, Prentice-Hall, 1981.

The Wollongong Group, Internetworking: An Introduction, The Wollongong Group, Palo Alto, California, 1989.

Paper Number: 3102

Save the Trees! (and your printers and people)

Debra B. M. Canfield
Dairylea Cooperative Inc.
P.O. Box 4844
Syracuse, New York 13221-4844
315-433-0100

I grew up in the woods. My hometown is the second largest city in New Jersey, by area that is. "*53 square miles of scenic beauty*" was the caption on the only postcard I can ever remember seeing of Estell Manor, and that showed a picture of a tree-lined country road. With a population of about 650, we had a lot more trees than people. So, I'm naturally interested in saving the trees. However, to be honest with you, my interest in reducing the amount of printing we do at work didn't start with a desire to save the trees. It came from a desire to make my life easier by reducing the amount of paper we had to look at, print, load into the printer, take off the printer, distribute, etc.

This paper presents the techniques we've used at Dairylea Cooperative Inc. to reduce printing. You probably won't find anything here which will be in competition for "great idea of the year," but hopefully, at least one of the ideas will be something you had not thought of before or a catalyst to help you think of something better.

First, a number of techniques for reducing the amount of printing which is done will be discussed. Then, we'll look at general principles for deciding where to start and how to implement printing reduction. Finally, we'll review the benefits produced by reducing your printing requirements.

Techniques to Reduce Printing Requirements

;NOCCTL

Carriage-control is a good technique for quickly adding white space to your paper. When you need white space, your printer will put out the pages much faster by advancing pages, etc., than by *printing* blank lines. But sometimes, you really don't need all that white space.

For instance, our general ledger system produces nicely laid out maintenance reports. Among its features is a page advance every time the department changes. If you add the same account number to fifty locations, the report you get to check your results is fifty pages long. All I really want is a one-page report with one line for each of the fifty accounts. Instead of being able to quickly scan down one page for errors, I have to flip through fifty pages.

What are my alternatives? I really need to see the report, so I can't just delete it. The maintenance program is part of a (horribly written) purchased package, so changing it is not feasible. My solution is to print it without carriage-control. All I had to do was modify the file equation in the job stream by adding ;NOCCTL as follows:

```
:FILE LP;DEV=LP;NOCCTL
```

No, reports without carriage-control are not pretty. But, if you just need to look at it fast and aren't trying to impress the chairman of the board, who cares? I find ;NOCCTL to be a great solution when I need to see a report but don't care what it looks like, and when modifying the creating program is impossible or simply not worth the time it would take.

Figure 1 summarizes the implementation effort cost of using ;NOCCTL. Implementation is rated on a scale from easy to difficult, and cost ranges from free to expensive. ;NOCCTL is easy to implement, since you only have to modify file equations. As with any technique you use for the first time, your initial implementation will take a little more effort, since you need to test it and make sure you understand how it works. Once the file equations are changed, no effort is needed to use ;NOCCTL on an ongoing basis. Since ;NOCCTL is built into MPE, the only cost involved is the time you spend on implementation.

Stdlist Handling

I have always been annoyed by all the page breaks MPE puts into Stdlists. What a waste of paper and of my time when I have to look through them. Every time the job runs FCOPY or QUERY, does a LISTF or STORE/RESTORE, etc., you get a new page. Do you need three FCOPYs in a row? You get three pages. Unfortunately, adding ;NOCCTL to the end of the JOB command doesn't work.

What are my alternatives? I could just delete the Stdlists. I could figure out a way to print them without carriage-control. Or I could buy, or write, software to take care of them.

Deleting Stdlists is easy. Just add the following line anywhere in your job

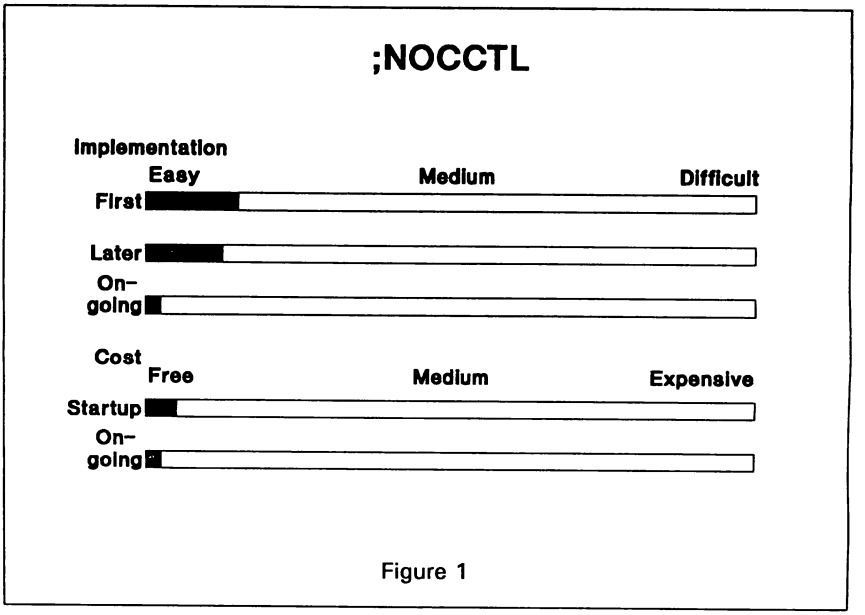


Figure 1

streams. I recommend that you add it right before the :EOJ line, so your Stdlist won't be deleted unless your job stream runs all the way through.

```
:SET STDLIST=DELETE
```

Personally, I find deleting Stdlists spooky. I don't always look at Stdlists, but it's nice to have them if I want to look at them. Many times I've gone back to old Stdlists to see what actually happened or to show a user what jobs they actually ran (versus what they say they ran).

There are two cases where I do use :SET STDLIST=DELETE. First, for security reasons. If there is something in the job stream that I definitely do not want anyone to see, I want it to disappear. Second, we use :SET STDLIST=DELETE to get rid of successful compile streams. Other criteria may make more sense at your shop, but while we may delete Stdlists for test jobs, we never delete them for production. In fact, we save them for two months.

Back in 1986 at the Detroit conference I discovered a vendor selling a software package that would look through standard lists for errors and would print them all together at a later time, removing the carriage-control for me. At that time the software was called "$Stdlist Management Processor," and that's exactly what it did. One day of running the trial copy was enough to know we could no longer live without it.

Granted, the biggest plus for us is looking through the Stdlists for errors automatically (our operators used to have to do this), but a big fringe benefit is the way it prints the Stdlists. I have it set to merge them all together at 4:00 A.M., and then we print them, index included, in the morning. Instead of our Stdlists being printed and piled up during the day, making them more likely to be lost, the listing is much smaller, because the carriage-control is gone, and the Stdlists don't get lost, because they are printed all together the first thing each morning.

$Stdlist Management Processor eventually developed into JOBRSQ, which does much more than what I've briefly mentioned here. In fact, to even further reduce your printing, JOBRSQ has an option for sending your Stdlists to microfiche. If you're interested in finding out more about JOBRSQ, contact NSD.

If you don't want to buy a complete Stdlist solution, and you don't have the resources to develop something fancy for yourself, you can still compress the carriage-control out of your Stdlists on line or in batch with the routines shown in Figures 2 through 4.

Figure 2 shows a command file for on line spoolfile compression using the native mode spooler for MPE XL. The approach is to build a file to hold the spoolfiles temporarily and set a ;NOCCTL file equation for it. The command file prompts for spoolfile numbers, which it copies to the holding file. When the last file is copied, indicated by pressing <Enter> in response to the spoolfile number prompt, the holding file is printed and then purged.

Note that since the command file simply prompts for spoolfile numbers, it can be used to eliminate carriage-control from any spoolfile. Therefore, it might be useful for reports for which you want to eliminate carriage-control as discussed above, but for which you cannot access the file equation.

The job stream in figure 3 is strictly for Stdlists with the native mode spooler, since it uses the LISTSPF command to get a list of spoolfiles with a designation of "$STDLIST." It uses a similar approach to the command file, but puts

Command file for On Line Spoolfile Compression with Native Mode Spooler

```
COMMENT Command file to compress printing by eliminating CCTL.
PURGE NOCCHOLD
BUILD NOCCHOLD;REC=-1008;DISC=100000;SPOOL
FILE NOCCHOLD;NOCCTL;ACC=APPEND
WHILE SETVAR (SPOOLFIL, INPUT ("Spoolfile # to compress:")) <> ""
IF FINFO('O!SPOOLFIL.OUT.HPSPOOL',0)
   FCOPY FROM=O!SPOOLFIL.OUT.HPSPOOL;TO=*NOCCHOLD
   SPOOLF O!SPOOLFIL;DELETE
ELSE
   ECHO Spoolfile !SPOOLFIL does not exist.  Stdlist not copied.
ENDIF
ENDWHILE
SPOOLF NOCCHOLD;PRINT;DEV=LP
PURGE NOCCHOLD
```

Figure 2

Batch Stdlist Compression with Native Mode Spooler

```
:JOB NOCCSTRM,OPERATOR.SYS;OUTCLASS=,1
:COMMENT Job stream to compress all Stdlists.
:PURGE NOCCHOLD
:BUILD NOCCHOLD;REC=-1008;DISC=100000;SPOOL
:FILE NOCCHOLD;NOCCTL;ACC=APPEND
:PURGE NOCCLIST
:FILE NOCCLIST;REC=-80
:LISTSPF O@;SELEQ=[FILEDES=$STDLIST AND STATE=READY] > *NOCCLIST
:EDITOR
 TEXT NOCCLIST
 DELETE 1/3
 FIND "INPUT SPOOL FILES"
 DELETE *-1
 DELETE */LAST
 CHANGE 9/80 TO " " IN ALL
 CHANGE 9 TO "%" IN ALL
 HOLD ALL
 CHANGE 9/80 TO ".OUT.HPSPOOL;TO=*NOCCHOLD" IN ALL
 CHANGE " ." TO "." IN ALL  (because FCOPY doesn't like spaces in the filename)
 CHANGE " ." TO "." IN ALL
 CHANGE " ." TO "." IN ALL
 CHANGE " ." TO "." IN ALL
 CHANGE " ." TO "." IN ALL
 CHANGE "#" TO "FCOPY FROM=" IN ALL
 ADD ,HOLDQ,NOW
 CHANGE "%" TO ";DELETE" IN ALL
 CHANGE "#" TO "SPOOLF " IN ALL
 KEEP
 END
:SETVAR HPAUTOCONT TRUE (so job will continue if Stdlist disappears before it's copied)
:NOCCLIST
:SPOOLF NOCCHOLD;PRINT;DEV=LP
:PURGE NOCCHOLD
:PURGE NOCCLIST
:SET STDLIST=DELETE
:EOJ
```

Figure 3

Batch Spoolfile Compression with MPE V and SPOOK5

```
:JOB NOCCSTM1,NOCCUSER.SYS;OUTCLASS=NOCC,1
:COMMENT First part of process to compress Spoolfiles with SPOOK.
:COMMENT NOCCUSER.SYS should be a unique user so SPOOK append
:COMMENT in NOCCSTM2 doesn't get confused.
:COMMENT All Spoolfiles which you wish to include should be sent
:COMMENT to a unique device class, NOCC, so other spoolfiles
:COMMENT are not included in the NOCCTL copying.
:SHOWOUT SP;JOB=@J;READY;DEV=NOCC
:STREAM NOCCSTM2
:EOJ

:JOB NOCCSTM2,NOCCUSER.SYS;OUTCLASS=NOCC,1
:COMMENT Second part of process to compress spoolfiles with SPOOK.
:CONTINUE
:PURGE NOCCLIST
:RUN SPOOK5.PUB.SYS
 FILE NOCCLIST;REC=-71,3,F,ASCII
 APPEND NOCCUSER.SYS;ALL,*NOCCLIST
 EXIT
:EDITOR
 TEXT NOCCLIST
 WHILE
 FIND ":JOB"
 DELETE 1/*
 FIND FIRST
 FIND "DEV/CL"
 DELETE 1/*
 DELETE *        (This procedure will not work unless more than one spoolfile
 FIND "FILE"     is found by SHOWOUT in NOCCSTM1.  If there are less than 2
 DELETE */LAST   spoolfiles, "FILE" will not be found.)
 DELETE *
 CHANGE 19/72 TO " " IN ALL
 CHANGE 1/10 TO "%" IN ALL
 HOLD ALL
 CHANGE 10/72 TO ";ALL,*NOCCHOLD" IN ALL
 CHANGE "%" TO " APPEND " IN ALL
 ADD ,HOLDQ,NOW
 CHANGE "%" TO " PURGE " IN ALL
 ADD
 EXIT
 YES
 :FILE LP;DEV=LP;NOCCTL
 :FCOPY FROM=NOCCHOLD;TO=*LP
 :SET STDLIST=DELETE
 $EOJ
 //
 CHANGE "$EOJ" TO ":EOJ" IN ALL
 ADD .1
 $JOB NOCCSTM3,NOCCUSER.SYS;OUTCLASS=NOCC,1
 :PURGE NOCCHOLD
 :BUILD NOCCHOLD;REC=-132;DISC=100000
 :FILE NOCCHOLD;NOCCTL;ACC=APPEND
 :RUN SPOOK5.PUB.SYS
 //
 CHANGE "$JOB" TO ":JOB" IN .1
 KEEP
 END
:STREAM NOCCLIST
:SET STDLIST=DELETE
:EOJ
```

Figure 4

the listing of Stdlists from LISTSPF into a file and then uses EDITOR to create a command file which copies and purges the spoolfiles. Note that with either approach, the carriage-control characters are not actually gone. They print in the first column of the listing as characters and are no longer interpreted as carriage-control codes.

Figure 4 shows a spoolfile compression job stream routine for MPE V and SPOOK5. A command file routine is not provided, since MPE V does not provide WHILE and INPUT commands. The first job stream, NOCCSTM1, does a SHOWOUT of all ready spoolfiles on a particular device. By sending your Stdlists to a unique device class with the :OUTCLASS parameter, you will insure that only Stdlists are copied. NOCCSTM1 then streams a second job, NOCCSTM2, which uses SPOOK5 to copy the Stdlist just produced for NOCCSTM1. Next, EDITOR is used to manipulate the Stdlist, leaving only the spoolfile numbers and adding commands to create a job stream which appends and purges the spoolfiles. Finally, it streams the third job, NOCCSTM3, which it just created.

You cannot use :SET STDLIST=DELETE in NOCCSTM1, since its Stdlist is used in NOCCSTM2. When the job is run the second time, the Stdlist from the first run will be included in the SHOWOUT and will be printed and purged in NOCCSTM3. NOCCSTM2 uses APPEND in SPOOK5 and a WHILE loop in EDITOR to delete all but the last Stdlist. [*]

The command file and job streams shown are designed to serve as examples. If you decide to use them, you might wish to enhance them. For instance you could add a :STREAM command with the ;IN parameter to the job stream to have it

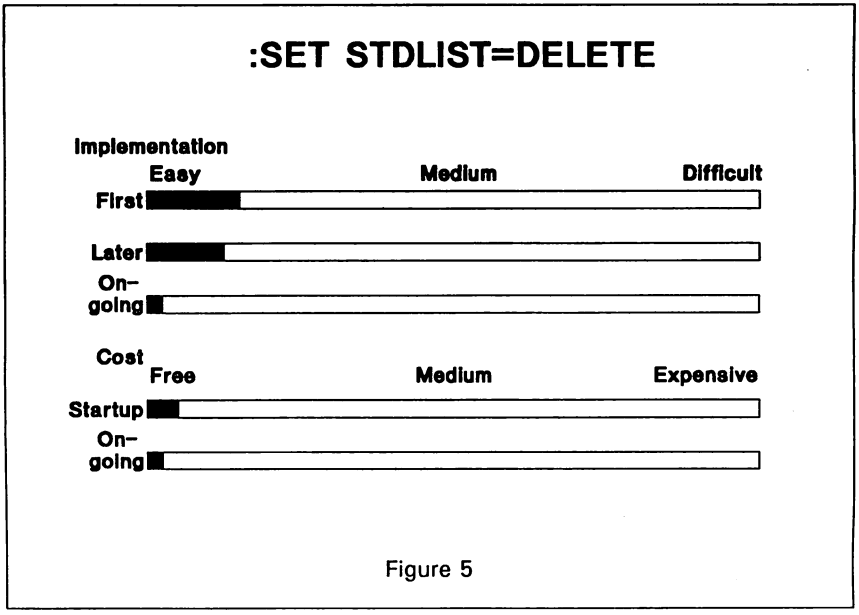


Figure 5

* I wish to express my thanks to Dave Stanton of Chestnut Data Systems for testing and debugging the MPE V/SPOOK5 job streams. Since Dairylea now uses MPE XL and the native mode spooler, I was unable to test the SPOOK routine for myself.

automatically restream itself to run at a certain time each day. If you are going to print many Stdlists one after another, you may wish to add lines of ***s or some other character just before :EOJ or just after :JOB in your job streams to help you find where each job starts and ends.

Remember, you can't take the carriage-control out of a Stdlist if it starts printing before you grab it. If you want to access your Stdlists, you need to keep your outfence up, or use a low output priority with the ;OUTCLASS parameter in your job streams as shown in figures 3 and 4.

Bear in mind that any Stdlist handling option other than :SET STDLIST=DELETE trades processing for printing. If your system is already overloaded, you may be able to compensate in part by scheduling your Stdlist manipulation for off-peak hours. Also, if errors in your Stdlists are your only notification of problems, you will need a method to either override your special handling and print the Stdlist in question immediately or develop another method for error reporting.

Figures 5 through 7 summarize the implementation effort and costs for the described Stdlist techniques. :SET STDLIST=DELETE is similar to ;NOCCTL in that it comes with MPE and involves only a simple addition to job streams. JOBRSQ requires some thought and decisions up front, but then just runs automatically, taking care of itself. Since it comes from a vendor, there is an initial license fee and ongoing maintenance charges. Maintenance includes periodic updates which require only a little effort to install and a little more to read about and decide whether to use the new features provided.

Using a job stream versus a command file to compress Stdlists will probably require more initial thought and planning, but a command file will require more

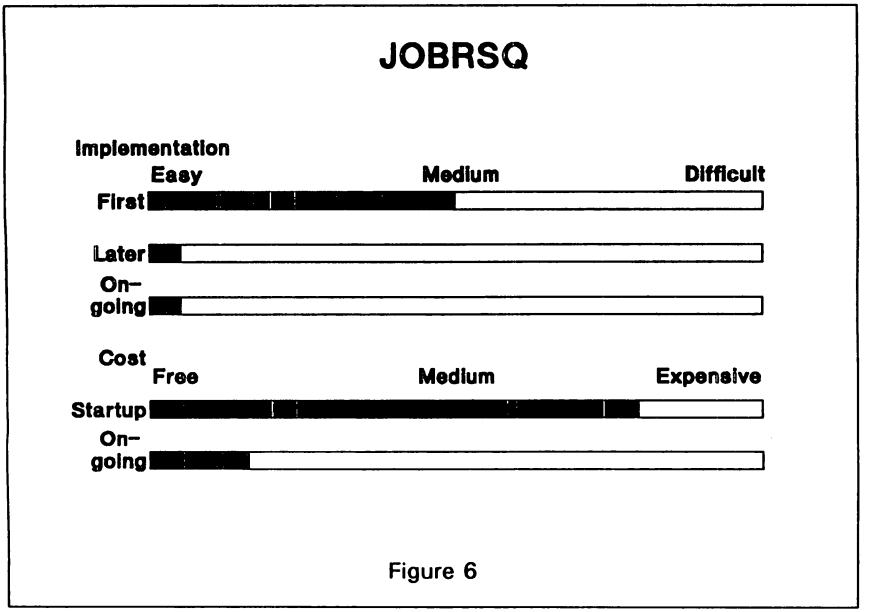


Figure 6

ongoing effort, since spoolfile numbers need to be entered one by one each time they are used. Since they only require MPE, the cost is low.

Microfiche

Back in 1983 when I started working for Dairylea, my first assignment was to work on the accounting reports. When I asked to see the year-to-date general ledger, I was told that they only printed monthly ledgers. A year-to-date one would be too long to print.

They were probably surprised at the request, because how could I possibly think they'd print something so long? After all, you could just look at all the monthly reports to get the same information. I was surprised at the answer, because I certainly didn't expect to be handed the report on *paper*. Hadn't they ever heard of microfiche?

Microfiche is a highly reduced alternative to printing. Reduction ratios can vary among 24, 42, 48 and 72. At a reduction ratio of 48, 270 pages of printed material fit on a single sheet of microfiche, which measures about 4" by 6". 42 and 48 are the most common reduction ratios used. Using a ratio of 24 requires more fiche, and at 72 it is difficult to get consistent, clear results. While you can hold microfiche up to a light and get a general idea of the type of pages it contains, you really need a microfiche reader to view it. Microfiche readers are ordered with lens sizes to match the reduction you use, and reader/printers are also available. Generally a company using microfiche would need at least one reader/printer to provide hard copy when microfiche is the only source for certain information.

The cost of microfiche is about $2.50 per sheet for originals and 19 cents per sheet for copies. Compare that to the cost for 270 pieces of paper, the time it takes to

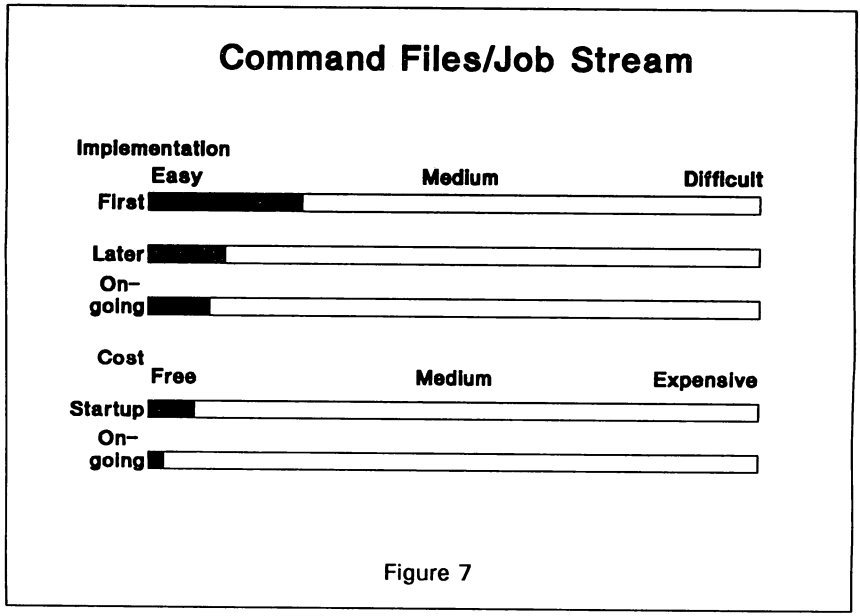


Figure 7

print and distribute it and the space it takes to store. If you can replace more than one copy of a report with microfiche, the media cost will actually be less with microfiche than with paper. Prices on microfiche readers and reader/printers vary widely, but as a rough estimate, you could expect to pay about $250 for a reader and $1700 for a reader/printer. Companies with large printing requirements may be able to recoup this cost in dollars as well as ease of use. It's not easy to lug around reports hundreds of pages long when you need to look at them.

For Dairylea, microfiche has made a big difference. Now, rather than not having year-to-date general ledgers because they are too long to print, we can fit twenty years' worth in a shoe box. We used to buy two-part checks, and our copies filled a room. When someone needed a copy, it was cumbersome to find the original and put the large binder on the copy machine to get it. Now we save money with one-part checks, have all our copies easily available on microfiche and quickly make copies when needed with the reader/printer.

The procedure for obtaining microfiche is quite simple. We create the reports we want to go to microfiche initially as a disc file when we don't need to print a copy, and then FCOPY the disc file to tape. If we do need to print a copy, as in the case of checks, we save a copy of the spoolfile and copy that to tape. Actually with our checks we offer direct deposit as an alternative, so for the microfiche we append the spoolfile from the direct deposit confirmations, which are laid out like checks, to the actual check spoolfile to obtain a single set of microfiche for the check run. We do our FCOPYs in jobs which are streamed by the creating job for disc files and are streamed by JOBRSQ for spoolfiles. The only thing we need to do is load a tape, call the microfiche company to come get it, and we have the microfiche back the next day.

Where do you start if you want to try microfiche? Look in the yellow pages under "Microfilming Service, Equip & Supls." Call a couple of companies. They will be happy to pay you a visit and tell you all about it. They will have you create a sample tape and give them a few sample printed pages. They can automatically include background information on the microfiche, such as a check form, so that when you view and print from the microfiche, you will see the *check*, rather than just what you print on the check. You can also find companies to sell you readers and reader/printers in the same section of the yellow pages. As with anything else, the companies will vary. Even though we had used the same company for years, we switched to a different company about a year ago when we found someone who provides better prices and service.

Just a note on microfilm: Microfilm is more familiar to the general public than microfiche. Microfilm is similar to microfiche, except that it comes in long narrow rolls, something like camera film. As I understand it, microfilm was available first, but in about 1979 or 1980 the industry changed, and many companies who were using microfilm for C.O.M. applications switched to microfiche. C.O.M., which stands for computer output microfilm, is what you use when you skip the paper and go right from the computer (generally via magnetic tape). Ironically, C.O.M. is produced primarily on *microfiche*, rather than microfilm as the name implies. Microfilm is widely used for source documents applications, that is, by copying hard copy documents onto microfilm.

The implementation effort and costs of microfiche are summarized in figure 8. Getting started requires some investigation and testing, but once you have your first procedures worked out, setting up a new job for microfiche is relatively easy. When you automatically create the required files and job streams to copy them, the ongoing effort involves little more than loading a magnetic tape. The initial expense involves the cost of the readers or reader/printers you require. Depending on volume, the

ongoing use of microfiche can actually produce a savings rather than an expense when compared to printing on paper.

Reports on Computer Screens

Detroit, back in 1986, was a great conference for me. Besides finding JOBRSQ, I read a short product announcement in the conference *Daily* for software to let you view your reports, instead of printing them. That sounded like good news to me, so I went right over to the Chestnut Data Systems booth to ask about SCAN. Unfortunately for me, it was *brand new*, as in *not quite done yet*. Fortunately, as soon as it was done, they sent a trial copy, and after loading just one report into it, I knew we had to have it.

The first reports we used SCAN for were, guess what, those same general ledger reports for which we were using microfiche. Each month when we ran the new year-to-date general ledger, we threw out the microfiche from the prior month. The only one we saved was the final for the year. With SCAN, we could update the general ledger as often as we wished without the cost of producing microfiche. We still produce a copy on microfiche for permanent storage at the end of the year, but the cost of SCAN was recovered in less than a year just from what we saved by not microfiching those general ledger reports each month.

The next application for which we used SCAN was our member trial balances. It seemed like accounting was always running them, and they always had to have them right away, so someone in our department would be interrupted from their own work to get the report off the printer. Since the report is fairly long, it wasn't feasible to send it to a slower printer in the accounting department. Also, since the report was laid out so nice, jumping to a new page for each member, it wasted lots of paper.

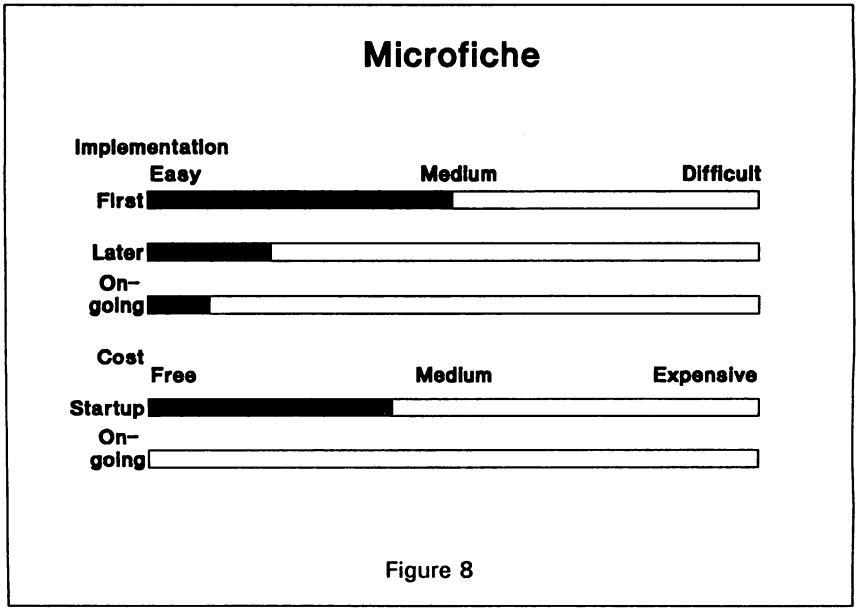


Figure 8

I talked to the culprit in accounting. Why do you run so many? *If they made any transactions, they needed a new copy.* Why do you need it in such a hurry? *Something about getting checks out (life or death in our business).* Do you keep them all? *No, just the final for the month.* Do you look at the whole report each time you run it? *No, just the total on the last page to make sure it is right, and maybe the pages for a couple members.* How would you like to be able to look at it on your screen and just print out the whole report or certain pages to whatever printer you want only when you need a hard copy? *Yes, indeed.* It was just the solution for them and us.

Now, when accounting runs a trial balance they have an option for printing it or putting it into SCAN. They always choose SCAN. The job stream displays a message back on their terminal when it's loaded, and then they can call it up, jump directly to the last page to check totals and jump directly to the page for any member they need to check. When they have their final for the month, they simply print it through SCAN. They aren't in a hurry to get the printed copy, because they know it's right, since they looked at it on their screen. They get their work done faster, and we don't get interrupted or waste paper.

To be useful to us, software to provide viewing reports on the screen has to have the following features:

1. It has to be easy to get the report into it. I don't want to be interrupted, and the user has enough problems.

2. It has to be easy to use. See note about users in point 1.

3. It has to provide direct access. It's not realistic to hunt through hundreds of pages on your screen to find a particular account number or member. You can't flip through screens as fast as you can through paper.

4. It needs good printing options. While part of the benefit is to eliminate printing, we do need to print some pages sometimes, and we want to be able to select the printer and environment we want to use.

5. Adequate security must be provided. Our users don't have access to MPE, and we only want certain people to be able to see and otherwise handle certain reports.

Besides passing on all the above points, SCAN offers other useful features such as archiving reports to tape, keeping a certain number of copies of each report, or keeping it for a certain number of days, freezing page headings, 132 column mode, etc. For more information contact Chestnut Data Systems.

Shortly after we bought SCAN one of our other third-party software packages added screen report viewing to their package. My first reaction was, "Oh no, I wasted my money." But when I saw what they had to offer, I quickly changed my mind. Making reports available in an editor type of environment, even if you can't edit it, is not my idea of a useful way for users to view reports on their computer screens.

Viewing reports on your computer screen instead of printing them is great for long reports, reports you need fast, reports you simply throw away, reports you update frequently, reports which need to be shared, etc. While it may seem like what you need in some of these cases is something truly *on line* rather than just a report, it is often not the case. Many reports would not be practical to produce with an on line program, and if you are using purchased software, you may not be able to write an on

line program. Also, if you already have a report which is just what you want, setting it up for screen viewing is nothing compared to the work it would take to write an on line application to do the same thing.

Remember, if you can view reports on your screen, you are keeping them in computer storage, which is a tradeoff for the paper you save and will affect your backups. In the right circumstances, it is well worth it.

Figure 9 summarizes the implementation effort and cost of using SCAN. As with any new software, SCAN requires some study up front to understand how to use it. After the first report, later reports are relatively easy to set up, and ongoing use requires no effort other than installing new updates. The cost is an initial license fee and ongoing maintenance charges.

Reduction Printing

While eliminating carriage control can make more fit on a page by eliminating empty space, reduction printing puts more on a page by treating a single piece of paper as if it was made up of multiple pages. The ability to print with reduction ratios of 2:1 (2 up) and 4:1 (4 up) was one of the major reasons we bought our HP2680A printer back in 1987. The savings in paper helped pay for it. Now with LaserJet printers and software written for them, reduction printing is available on a wider scale. Also, since reduction printing requires fewer physical pages, print speed increases.

The most common way we print on our HP2680A is 2 up, saving 50% of the paper used with standard printing. With this format, the characters are still easy to read, and it is more convenient when using a report. Take a compile listing, for example. How often do you flip back and forth when looking at a program to follow

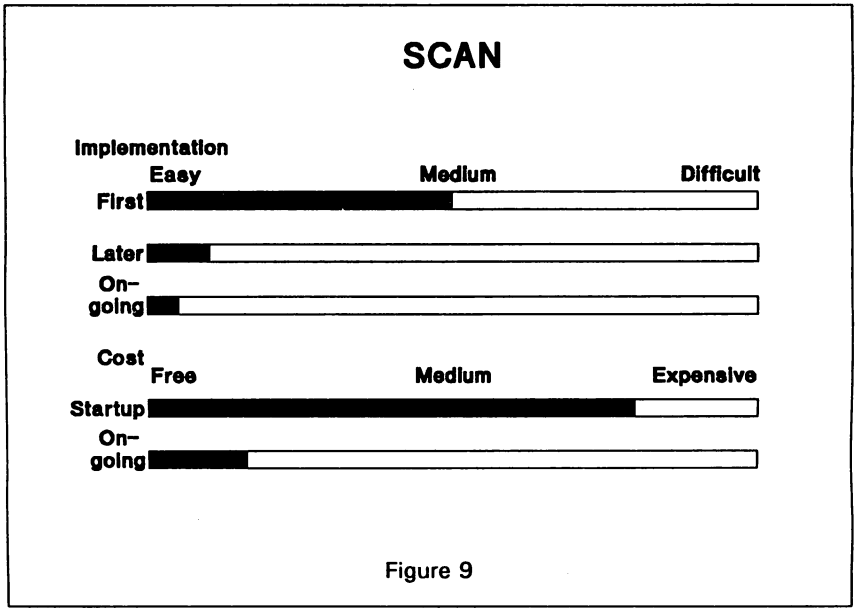


Figure 9

the logic? With two pages printed on a single sheet of paper, you don't need to flip pages as often.

We also do a fair amount of 4 up printing, saving 75% of the paper used with standard printing. We use 4 up printing for reports which we don't need to look at often, if ever, or which we only need to take a quick look at without actually reading. For example, we print our Stdlists 4 up. As I mentioned earlier, we want to print them in case we need them, but we don't really look at them unless we have a question or problem. 4 up is small, and when I have to read it line by line, I personally use a magnifying glass, but that inconvenience is more than made up for by all the paper we save and all the days we don't look at them at all.

4 up is also good for maintenance reports you only need to quickly glance through looking for errors. On the ones we print 4 up, any abnormalities show up because the line length or format is different, and if we spot this, we can look closely and read what it says. Accounting prints their monthly journal entry listing this way. Apparently they look at it only rarely, if ever, but they can't quite bring themselves to do without it completely.

Printing standard one page per sheet of paper (1 up) is the exception for us. People need a good reason to justify printing this way. One allowable reason is that they have to tear the report apart to distribute it. Paper tears easily on the perforation, but it's a pain to cut apart 2 up reports (though for some reports they still prefer to print 2 up and cut them apart). Another valid reason to print 1 up is when people need to write all over the report. If you need to write comments or numbers beside printed lines, it's hard to do it when the printing is small.

All we had to do to start using reduction printing, once we had our HP2680A laser printer, was to add one of the following parameters to our file equations:

```
;ENV=LP2.HPENV.SYS  for 2 up printing
;ENV=LP4.HPENV.SYS  for 4 up printing.
```

We have not tried any of the reduction printing software available for LaserJets, so I can't make any estimates of how easy they are to use or how much they cost. Depending upon the type and quantity of your printing, it might be well worth your time to investigate.

Another way to create reduced printing is to treat pages of narrow reports programmatically as multiple columns on a page, the way you create three across labels. This can be tricky programmatically, but we use Hewlett Packard's Business Report Writer (BRW), which gives us this option. In BRW you make this selection as "Number of Horizontal Pages" on the Define Reports screen. When a report is narrow and two pages can be printed side by side, we do so. Combining this with our standard 2 up printing, we essentially achieve 4 up paper savings with 2 up character legibility. Similarly, even if you don't have a printer which does 2 up printing, you can create the same paper savings with multiple horizontal pages.

With LaserJet IIDs and LaserJet IIIDs, duplex printing offers another paper saving opportunity. It isn't used much at Dairylea. In fact the only thing which I know of which is regularly printed duplex is the computer users newsletter I write. But duplex may be just the thing for something you do in your business. Duplex printing is controlled with the following escape sequences:

```
<Esc>&l1S for long-edge binding
<Esc>&l2S for short-edge binding
```

The implementation effort and costs of reduction printing are summarized in figures 10 and 11. While using ;ENV theoretically only involves changing a file equation, we found that the MAXDATA needs to be increased for some programs which create reports with an environment specified in the file equation. As part of MPE, the only cost, assuming you already have a printer capable of using them, is the effort to make the changes.

The implementation effort for creating multiple column reports which are essentially two or more pages side by side varies depending how you do it. If you have

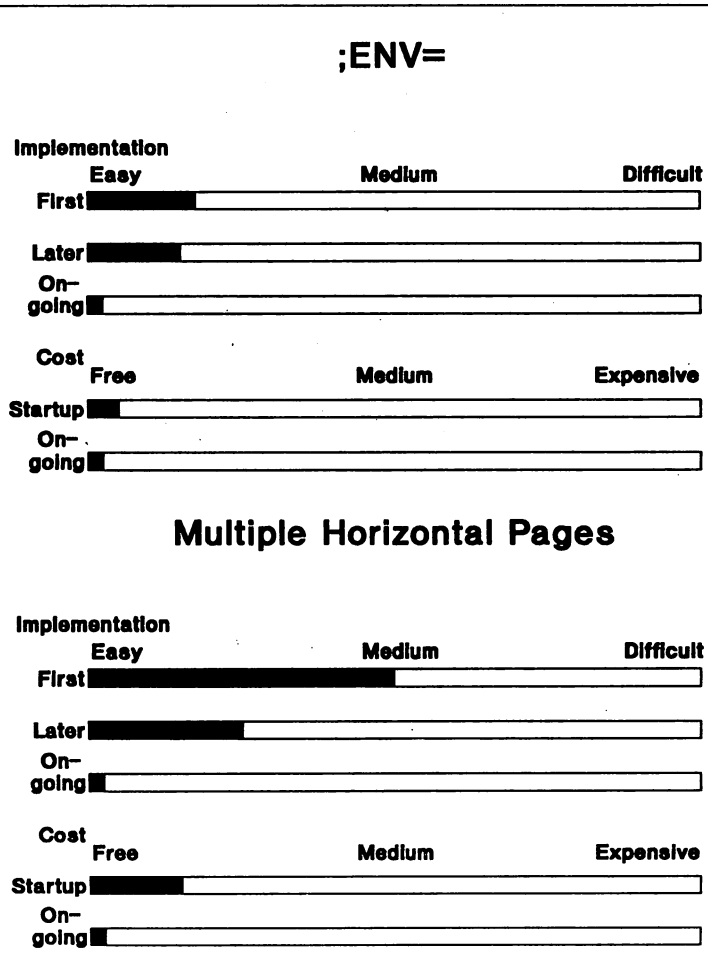


Figure 10

software which provides it as an option, it is quite simple. It will be more complex if you have to modify your programs to accommodate it. Duplex printing is easily accomplished with software which supports it directly or with a little more effort by adding the appropriate escape sequence as the first line of your printed report. Don't forget to add the reset escape sequence (<Esc>E) as the last line. Again, assuming you have a printer which supports it, the cost of duplex is simply some initial effort.

Eliminate

One of the funniest computer cartoons I ever saw was of a printer and a paper shredder side by side. As the paper was coming off the printer it was feeding directly into the shredder. Perhaps you don't have the direct feed option installed, but do you have any reports where this is the end result?

The biggest printing savings that can be achieved is when you can simply eliminate something you used to print. This sounds obvious, but with all the different reports and all the different people using them, it may be hard to spot when a report is not used. If you can't eliminate a report completely, you may be able to eliminate one or more of the copies printed.

Besides reports which aren't used at all, you may be able to eliminate printing if you discover that only a little piece of a report is used. For example, you may have a report showing pages of detail and a grand total. If the total is all that is used, you could shorten it to one page, or eliminate it altogether by having the total displayed on the screen of the requesting user. In one case where we do this, we still print the one page, so the user can save it as backup, but they can continue with their next task sooner by not having to wait for the piece of paper. That crucial number for which they are waiting is already displayed on their screen.

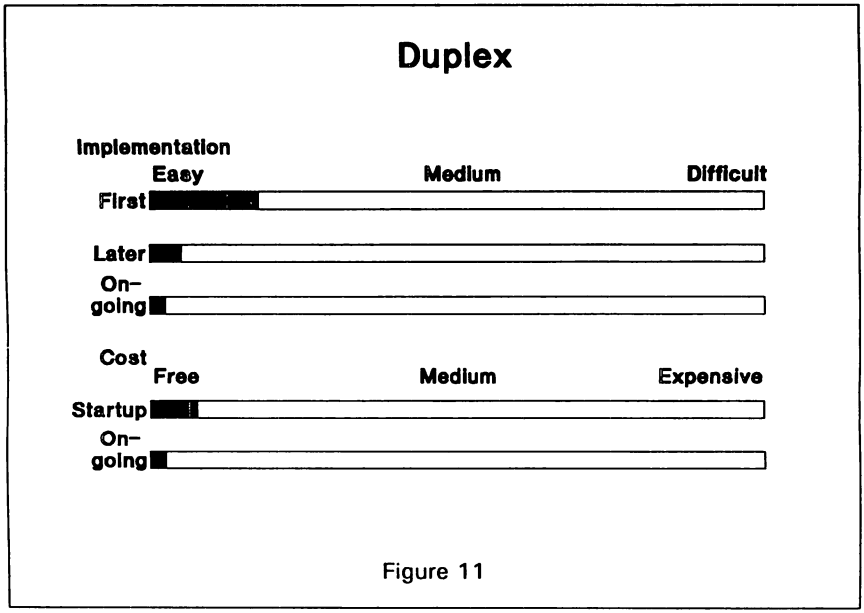


Figure 11

The old standby method of how to determine which reports are really needed is to stop distributing all of them and wait to see which ones people come looking for. However, just because someone comes for it doesn't prove that the report is actually used. Joe in sales may come looking for that report because he knows he has to run it every day and put it on Ann's desk. Joe may not realize that it goes right from Ann's desk into the paper recycling container.

Figure 12 shows the implementation effort and cost related to eliminating reports. The biggest difficulty with eliminating reports is identifying what can be eliminated. Or, if a report can't be eliminated entirely, you need to come up with an appropriate alternative. The cost is simply a factor of the time involved.

Direct Fax/EDI

Direct Fax and Electronic Data Interchange (EDI) have two things in common of which I am aware. First, they both seem to be potential ways to reduce printing requirements. Second, I don't know much about either one. They are included here for completeness and to suggest you investigate them if they appear to be a good fit for your organization.

If you are printing reports only to fax them to someone else, you'll reduce your printing and a big nuisance if you implement faxing directly from your HP3000. If you are in the kind of business where EDI would be helpful, go for it. Dairylea isn't, so I don't have time to look into it.

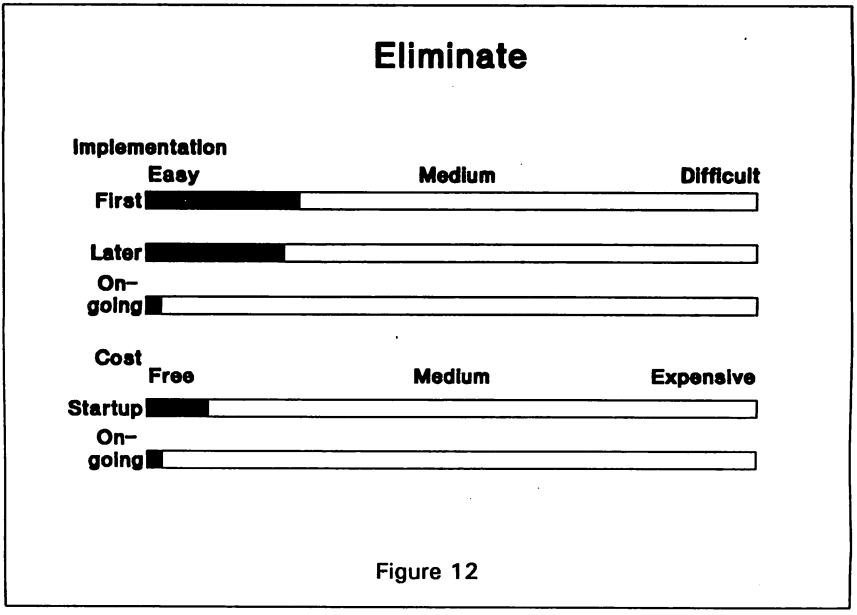


Figure 12

Implementation

So, you want to reduce your printing requirements, and now you have some ideas about how to do it. Where do your start?

First, identify areas where you can make an improvement. Try asking these questions:

1. What is the longest printed report?

2. Which report is run over and over?

3. What printing interrupts you?

4. What would be the biggest help to a particular department?

5. What will be simple/reasonable for you to do?

You may find a report which you can completely eliminate. Great! But if it's only two pages long and run once a year, big deal. You may identify an elegant solution for a given department when, in fact, you could have been a bigger help to them with a simple solution.

Don't start with a big complex project. Start with something easy, so you can get immediate results. You can enhance your solution as a second step, if necessary. For example, if you want to get into microfiche, you might want to pick a long report which doesn't require a special form, produce a disc or spoolfile, whichever is easier, FCOPY it to tape and get your first microfiche report into the hands of your users. Later you can work out the best way to get it to tape in a more automated fashion on a regular basis.

Next, determine which printing reduction technique is most appropriate for the particular printing requirement. Try these questions:

1. Does it have to look nice? If so, forget ;NOCCTL.

2. How long is it?

3. How long does it need to be kept, and how long does it need to be readily available?

4. How is it used? Does it need to be cut up? Does anybody write on it?

5. How many people use it?

6. Who uses it?

7. How quickly is it needed?

For example, when I talked about microfiche, I talked about how we use it for long general ledger reports and for copies of checks. When I talked about viewing reports on the computer screen, I talked about how it was even better than microfiche for the general ledger reports, but I didn't talk about the check copies. Setting the check copies up for viewing on the computer screen would be totally inappropriate.

For one thing there are way too many of them. I'd certainly have to buy more computer storage, and it would complicate my backups. More importantly, the users simply have no need for that kind of immediate access. They rarely use the check copies, but they need to keep them forever. Likewise, I didn't talk about member trial balances on microfiche. One of the main requirements for these trial balances is immediate access. They certainly can't wait for the overnight turnaround of microfiche.

Similarly, neither microfiche nor screen viewing makes sense when the people who use the report don't have access to computers or microfiche readers, and giving somebody with a visual handicap 4 up reports to use isn't very thoughtful.

Finally, put your solution into practice. Make sure you consider the people who will use the new solution and involve them early enough in the process. Unless you are absolutely positive they will be thrilled with the new approach, don't just throw it at them one day. People typically don't like surprises in the work place, and a little advance salesmanship on your part will make a big difference in whether you have a smooth transition and acceptance of the new approach. Emphasize what it will do for them, not what it will do for you.

Make sure that the people involved thoroughly understand the new approach and any new procedures they must use. At times, eliminating the possibility of using the old method may be an appropriate means of insuring that the new technique is used. Other times it is better to provide options for whether and how to print a particular report each time it is run.

Benefits

While many of the benefits of reducing printing requirements have been mentioned throughout this paper, they are listed as follows by way of summary:

1. Save paper, ribbons, toner, etc.
2. Save wear and tear on printers
3. Save wear and tear on people
4. Decrease storage requirements
5. Ease of use
6. Convenience
7. Available sooner
8. Available more widely
9. Reduce interruptions
10. Move toward operatorless environment

Not every technique will provide every benefit, but appropriate techniques for each circumstance will benefit all and help make your company a better place to live.

Appendix

Disclaimer: This paper is not a review of all software which might be used to reduce printing. Software used by Dairylea Cooperative Inc. is described as a way of illustrating a particular technique. If you are interested in a particular technique, you should investigate alternatives for yourself. Vendor shows at Hewlett Packard user conferences and advertising and reviews in HP related periodicals are good places to start looking for third-party solutions. Ask other users for recommendations.

In case you are interested in contacting either of the vendors mentioned in this paper, their addresses follow.

Chestnut Data Systems
Suite 613
6981 N. Park Drive
Pennsauken, NJ 08109
609-662-1611

NSD, Inc.
1400 Fashion Island Blvd.
Fourth Floor
San Mateo, CA 94404
415-573-5923

A Quick Look At The MPEXL Memory Dump For System Managers

Donald E. DeFreese
McDonnell Aircraft Company
5695 Campus Parkway
Building 274 Department 462C Mailcode 274 1125
Hazelwood, Missouri 63042-2338
(314) 233-3332

For most all HP sites, a system failure is bound to happen. I give HP credit for working very hard at providing the best releases of operating systems possible for general distribution. Howevei nothing is perfect. You may go for many months without a single problem and suddenly experience a crash. Some sites simply shrug their shoulders, document the crash and reboot the machine. But the problem can still hit you again when you least expect it. Others faithfully perform memory dumps at every failure, but allow HP to review the data to form a problem resolution for you. But to be an informed System Manager and get your hands alittle dirty in the dump could save you from experiencing that next crash.

This paper is not intended to be the all encompassing dump analysis to pinpoint exact causes of the failure and be 'the expert in the field'. Instead, use the information as a learning tool for yourself to gain added information on your own system. Don't be afraid to look at the dump. Your entrance will only be in an examining mode to look at the instant of death. At no time will you be modifying the actual operating system. Your view is thru the window and not smashing it.

There are two areas to address in the reading of the memory dump:
(1) Proper Execution of Dump and Restoration
(2) Interrogating Dump for Failure Information.

I realize that other papers and HP documentation has been written on performing memory dumps, but without proper procedures, valuable data could be lost.

A Quick Look At The MPEXL Memory Dump For System Managers

Step One - Execution of Dump and Restoration

Performing the memory dump has been documented by many individuals in different manuals and services. One that I recommend is on HP's Supportline dialup service from the Response Center available to Teamline and Responseline customers. You can obtain a copy by searching for document number MVAN000183 titled 'MPE/XL System Interrupt Recovery Procedures'. Supportline has a print command (PR DOC) that prints the document to a slaved printer. You should have a copy of this document handy for writing up your own operator procedures for inclusion into a workbook near the system console. This will reduce the phone calls asking how to do the dump.

As a synopsys of the dump process the steps are as follows:

(1) Write down System Abort information from console
The system abort number will be used at a later time for restoring the dump back to the system. This number will set the filename for use by DAT.

(2) Label a scratch tape with abort number, date and time.
Proper labeling of the dump tape will reduce questions later if multiple aborts occur.

(3) Mount the scratch tape in LDEV 7
This drive is also known as the 'Alternate Boot Path' you use for system updates and patches. The tape should be loaded and online before continuing.

(4) Press <ctrl> B to display the CM> prompt
If the prompt does not display, you may have the cpu locked out. On the series 950-980, a key is on the display panel with three possible positions. If the key is in the 'Secured' position, you must turn the key straight up to the 'Console Enabled' position. Then return to the console and press the <ctrl> B again. This time, the CM> prompt should display.

(5) Type TC and press return
The TC command is a Transfer Control command that reboots the cpu from disk without resetting the hardware. A very important point is to use only the TC command and NOT the RS command. If you should enter the RS command, the system memory will be erased making a memory dump impossible.

(6) If autoboot is enabled, press any key to cancel operation.
On many systems, the autoboot option is enabled that will execute a preset startup mode without operator intervention. If you do not interrupt this operation, the system will reboot itself and wipe out all memory data. If this option is not enabled, the system will stop at the ISL> prompt.

(7) Boot from primary boot path (Y or N)?
Since the system is to be booted from disk rather than tape, answer Y to boot from primary path.

(8) Interact with IPL (Y or N)?>
You must enter Y to interact with IPL if a dump is to be taken. Otherwise, the system will start under a START RECOVERY and wipe out all memory data.

(9) At the ISL prompt, enter **DUMP**
This will begin execution of the dump program. The program is loaded from disk and start operation.

(10) Enter user identification string for this dump (80 chars or less):
The dump program allows you to enter a description of up to 80 characters that will be written to the dump tape. This is not required and can be bypassed by pressing return.

(11) Wait for dump tape to be written to.
Depending on the size of your machine (amount of real memory plus number of disk drives for virtual memory) the dump could take quite awhile. You will receive messages about the amount of virtual memory on each drive that is being dumped.

(12) Once the dump is complete, system will reboot
The system restarts the boot process all over again after the conclusion of the dump. This is so the ISL is executed and the type of start can be initiated.

(13) If autoboot is enabled, press any key to cancel operation.
On many systems, the autoboot option is enabled that will execute a preset startup mode without operator intervention. If you do not interrupt this operation, the system will reboot itself and wipe out all memory data. If this option is not enabled, the system will stop at the ISL> prompt.

(14) Boot from primary boot path (Y or N)?:
Since the system is to be booted from disk rather than tape, answer Y to boot from primary path.

(15) Interact with IPL (Y or N)?:
Enter Y to interact if you wish to restart the machine under a START NORECOVERY. Otherwise the system will start by default under a START RECOVERY.

(16) At the ISL prompt, enter **START RECOVERY**
After any system abort, a START RECOVERY is the preferred method to restart the operating system. This will retain all streamed in wait or restart jobs as well as all spool files. Under Native Mode Spooler, a START RECOVERY is not required to retain output spoolfiles. But if a NORECOVERY was executed, streamed jobs would be removed from the system.

(17) Bring the system back up for system recovery.
Some applications that were executing will require recovery after system aborts. Your site procedures will dictate what functions Operations should perform at this time.

Now that the system is back up, users could log back on and resume activity. HP however recomends that a START NORECOVERY be performed as soon as possible following any type of system abort. This is a good practice to recover fully from any type of abort to completely rebuild all system tables, clearing all transaction manager activity in a normal manner. System reliability is not assured under the START RECOVERY mode.

Following a successful restart of the system, the load of the dump to the system is needed. To restore the dump tape, log onto MGR.TELESUP. The

Response Center prefers the dump be restored to the 'DAT' group where the dump program resides. But my on opinion is to create a new group called 'DUMP', home your logon to this group and create the dump files there. This way, all dump files are centrally located, separate from HP's own code and easily found for later cleanup. You may also want to exclude the group from any backups you perform due to the large size of the file created. The dump that the following examples were taken from was over 900,000 sectors (128 megabytes of real memory plus 10 system disk drives for virtual memory). If you use a HP 7980XC tape drive for backups, this file set will take up almost 1/2 of a single tape.

As you have probably performed in the past, the command to load the dump back to the system is GETDUMP. The Supportline application note recomends that the dumpname to use begin with the letter 'A' followed by the abort number, or the word 'HANG' for system hangs. Similar to the use of the 'DUMP' group to find the dumpfiles easier, a standard name for each dume will separate out your current dumpfile from others that may still be on the system.

Step Two - Interrogating Dump for Failure Information.

Now comes the time to review the dump to analyze your failure. From step three, you performed the restore of the dump tape back to the system. If you have exited the DAT program, reexecute DAT.DAT again.

For clarity sake, the use of different character fonts will designate if the characters are input by you, or output from the program. If the characters appear as **SHOWJOB** they are input by you.

Many System Managers may not have noticed the introductory banner that DAT displays to the screen. If you were alittle curious, you may have stumbled into the function HP uses at the Response Center to begin reading the dump. The entry banner for Dat looks something like the following (this was taken from the DAT program provided on MPEXL release 2.2):

```
Type "macstart" to load Macros & Symbols.

MACSTART
Welcome to the DAT Macro facility.

Enter the dump file set name to process: 1
```

The tipoff was the message to Type "macstart" for macros. HP has provided a wealth of predefined macros that setup easy to execute functions that steps you thru the dump. Please perform the following steps to begin the dump reading. Your response to the dump file set is the same file name from the dump restore of step three. Do not enter the full filename of the dump file. Leave off the suffix 'MEM'. DAT assumes the memory dump file always ends in the characters 'MEM'. After entering the filename, the macro displays the abort number read from the file, last active process number and operating system release. The display will look similar to the following:

```
Error reading dumpworthy file name length from descriptor record.

Dump Title: SYSTEM CRASH CAUSED BY CARELESS SYSTEM MANAGER
Last PIN  : c6

WARNING! Errors were encountered during dump open.
Do you wish to continue anyway? (N/Y)  Y

RELEASE: A.41.01   MPE XL HP31900 A.51.07 . USER VERSION: A.41.01

(UNWIND   - Unwinding Out Of Lockup Loop)
(UWLOCKUP - HALT $7,$406 = #7,#1030)
```

This release of MPE that failed was A.41.01 or MPEXL 2.2. The term last pin represents the active process number the system was executing that the system created an abnormal situation. To prevent a possible data corruption, the system will shut down.

Once the macro has determined the release of the operating system, you will be prompted for which macros you would like loaded for you. So the DAT program will display the following question:

```
Please choose which macros & symbol files you wish to load.
Multiple choice may be entered all at once (default <cr> is all).
Valid input examples: "1 2 3" or "OS DB" or <cr> or "ALL".

   1)  OS macros & symbols
   2)  Data Comm macros & symbols
   3)  Data Base Core & Turbo Image Macros & symbols
   8)  User macros & symbols
   9)  Do not load any macros

Enter your selection:     1
```

The selection you should make at this point is selection 1 only. This will load enough information into macros to browse the operating system. Any other macros will only take additional time to load and are not used in this paper.

After entering your selection for OS macros, the program will display the following on your screen:

```
OS Symbol file SYMOS.OSA51.TELESUP is now open.
Next line maps VAMOS.OSA51.TELESUP
  VAMOS.OSA51.TELESUP   10000.0 Bytes = 1bd0
```

You may experience a warning message that reads:

```
WARNING! OS Build ID Timestamps in System Globals & SYMOS do NOT match.
```

You can disregard this warning since the macros will load successfully.

```
Additional date information about symbol files will be displayed before
control is turned over to you.
   OS Build ID Timestamp in System Globals = 1990091016
   OS Build ID Timestamp in SYMOS File     = 1990082715

OS Macros restored from file MOS.OSA51.TELESUP.
OS DAT MACROS  HP30357 A.40.49    Copyright Hewlett-Packard Co. 1987

$11c ($c6) nmdat >
```

Your needed macros are now ready for use. The messages about loading files and maps from a group named OSA51 refers to the release of the operating system that is being analyzed. MPEXL 2.2 operating system itself is known as product number HP31900 release A.51.07. You can find this information by doing a SHOWME command and reading the header acrost the screen. This release is set in the system and can not be changed.

Now let's find out what was happening at the time of the abort. The first macro command you should enter is:

MACHINE_STATE

```
SYSTEM ABORT #1030 FROM SUBSYSTEM #101 (Memory Manager)
An attempt was made to FREEZE an invalid virtual address.

SECONDARY STATUS: INFO = #-34, SUBSYSTEM = #107 (Virtual Space Management)
The length specified was beyond the bounds of the specified object.

(UNWIND   - Unwinding Out Of Lockup Loop)
(UWLOCKUP - HALT $7,$406 = #7,#1030)

MPE/XL VERSION: A.51.07            Last Pin : $c6

SYSTEM CONSOLE AT LDEV #20

                                   CPU: PROCESS_RUNNING

HP3000 SERIES 950 With Processor Revision 0.

CURRENT REGISTERS:

R0 =00000000 c0000000 0035a358 00002490 R4 =0000000a c7638000 4033f290 4033
R8 =00000000 4033f2a0 4033f328 00000000 R12=00000000 00000000 00000000 0000
R16=00000000 00000000 00000000 ffffffff R20=00000000 00ba0120 0000000f ffff
R24=04060000 ffde006b 04060065 c0202008 R28=00007ffd 00000000 4033f408 0035

IPSW=0004ff0f=jthlnxbCvmrQPDI  PRIV=0   SAR=0001 PCQF=a.190ed8   a.190edc

SR0=0000000a 00000000 00000000 00000000 SR4=0000000a 0000009c 0000000b 0000
TR0=006dc800 007dc800 00000058 4033f408 TR4=00000001 00002041 1f000010 0017
PID1=0314=018a(W) PID2=0626=0313(W)     PID3=0000=0000(W) PID4=0000=0000(W)

RCTR=00000000 ISR=0000000a IOR=00000000 IIR=00020005 IVA=0015b800 ITMR=735f
EIEM=ffffffff EIRR=03000000 CCR=0080
```

Much of what is listed here about register values is not needed for simple dump analysis. However, the upper portion does display the actual system abort number, subsystem number with subsystem description and a simple statement on what the problem was that the system detected to force the system abort. For this dump, a secondary status is also displayed with additional information for pinpointing the problem. So up to this point, one program was attempting a function on virtual memory that the memory manager halted before corrupting memory that was owned by another process. Within this display the ldev where the system console was currently set to is shown. This could also help to track down a problem if your shop switches the console location many times throughout the day. Finally, the item labeled 'Last Pin' is the process number of the active program when the system aborted. This process is the cause to the system failing.

Sometimes, your system has aborted in off-hours when no operators are monitoring the console. If you were expecting a console reply for scheduled processes, a search on this data will help later in system recovery. To find out what replies were waiting, enter the command:

UI_RECALL

```
                    Operator Requests Pending
                    -------------------------

No operator Requests Pending
```

For our abort, no replies were waiting. This could be useful if you needed to rerun any jobs that were active. Your recovery steps for restarts may be dependant on what step a job was in.

Next question would be 'what sessions and jobs were active at the time?' No problem with this answer. Simply enter the next command to receive a list of jobs:

UI_SHOWJOB

```
                                                                    JSMAIN
JOBNUM STATE IPRI JIN  JLIST INTRODUCED JOB NAME                      PIN

#S484  EXEC  8    825  825   150   22:13 M191517,MANAGER.SYS,PUB       $7e
#J2    EXEC  15    10S 11    144   23:13 LOGOFF,MANAGER.SYS,PUB        $34
#J3    EXEC  15    10S 8     144   23:13 SCOPEJOB,MANAGER.SYS,SCOPE    $28
#J4    EXEC  15    10S 8     144   23:13 SCOPEJB2,MANAGER.SYS,SCOPE2   $39
#J5    EXEC  15    10S 8     144   23:14 NETBASE,MGR.NETBASE,PUB       $51
#J6    EXEC  15    10S 8     144   23:15 NBRSPOOL,MGR.NETBASE,PUB      $69
#J64   EXEC  15    10S 8     148    5:00 NRJE,MANAGER.SYS,NRJE         $7c
#J8    EXEC  15    10S 8     144   23:17 IBMNODE,OPERATOR.SYS,IBMSPOOL $6b
#S474  EXEC  8     811 811   150   15:48 MONITOR4,OPERATOR.SYS,SYSOPER $3c
#S475  EXEC  8      20 20    150   15:48 CONSOLE4,OPERATOR.SYS,SYSOPER $ce
#S41   EXEC  8       4 4     148    8:51 LABEL1,OPERATOR.SYS,SYSOPER   $a4

11 JOBS:
    0 INITIALIZING;    0 INTRODUCED
    0 WAIT
   11 EXEC;     INCL   4 SESSIONS
    0 SCHEDULED;       0 SUSPENDED
    0 TERMINATING;     0 ERROR STATE
     JOBFENCE= 1; JLIMIT= 10; SLIMIT= 100
```

The format the UI_SHOWJOB displays is very similar to the MPEXL SHOWJOB command with session/job number, ldev, time and jobname. Under the column jobname is an expanded jobname that also displays the group each session/job is homed to. Knowing who was logged on at the time of the crash will help in your recovery procedures if any jobs require restart intervention.

Next, review the programs executing that were running in privilege mode. Quite often, a privilege mode program will attempt to execute a call that will result in a violation that the operating system considers to possibly cause system corruption. The method the system will use then to protect itself is to halt all processing. This is quite drastic, but a safe way to block any further attempts at the bad code.

PM_FAMILY

```
Parent PIN  Program File              JSMAIN PIN  Program File
----------  ------------              ----------  ------------
$bc         CI.PUB.SYS                $7e         JSMAIN.PUB.SYS

Family Tree for Process Number $c6
----------------------------------
$c6 (SCOUT.XLTOOLS.TELESUP)
```

You can see from this example that the current pin c6 was running the program SCOUT.XLTOOLS.TELESUP. This program was an old unsupported system resource monitor used under older releases of the operating system before the availability of HP Glance/XL. The use of this program was never guarenteed to function with every release of the system software due to changes in memory layout. You will see in this particular example, that the current process number (c6) happens to match exactly with the session number #484. You now know which user on the machine that was active at the abort time.

For the next step, a greater detail look at the active process is needed. A single command as follows will show the environment the program was running under:

PROCESS_CURRENT

```
********************* LAST PROCESS INFORMATION ******************
PIN: $c6                          PROGRAM FILE: SCOUT.XLTOOLS.TELESUP

JOB NUMBER: #S484

Process Library Names:

'XL.PUB.SYS'

'NL.PUB.SYS'

'SCOUT.XLTOOLS.TELESUP'

PIN $c6 NOT WAITING ON A SEMAPHORE.

PROCESS TREE:

$c6 (SCOUT.XLTOOLS.TELESUP)  #S484

               ===== DISPATCHER INFORMATION FOR A PROCESS =====
S
y
s
p
r
o
c PIN #  State          Wait Event     Pri   Class  Blocked Reason
- -----  -----          ----------     ---   -----  --------------
  $c6    EXECUTING      Not Waiting    $7aff  BS    NOT_BLOCKED

ENVIRONMENT INFORMATION:

TOTAL # OF SWITCHES      # SWITCHES TO CM        # SWITCHES TO NM
-------------------      ----------------        ----------------
        $0                       $0                      $0

ERROR STACK FOR PIN: $c6

Entries displayed in most recent to least recent order:

SUBSYSTEM      SUBSYSTEM         SUBSYSTEM
NUMBER         ERROR NUMBER      NAME
---------      ------------      ---------
#107           #-34              Virtual Space Management
The length specified was beyond the bounds of the specified object.

#8             #-1
Illegal DB register setting  (FSERR 1)
```

```
#8            #-150
INVALID ITEM NUMBER  (FSERR 150)

STACK MARKER TRACE:

(UNWIND    - Unwinding Out Of Lockup Loop)
(UWLOCKUP - HALT $7,$406 = #7,#1030)
       PC=a.00190ed8 system_abort
NM* 0) SP=4033f408 RP=a.0035a358 freeze+$1a0
NM  1) SP=4033f408 RP=a.002fa234 get_tune_dispatching_info+$264
NM  2) SP=4033f268 RP=a.002f9760 ?get_tune_dispatching_info+$8
```

Unless you want to debug the system code, the stack dump is of little use to you. However, the identification of the executing program name - in this case SCOUT.XLTOOLS.TELESUP from session number 484 pinpoints the precise user that caused the system abort. Referring back to the UI_SHOWJOB display from above, the logon id associated with the program is M191517,MANAGER.SYS (this happens to be me... I caused the abort by running the program). You could now go to the user and ask him/her questions on what was being attempted by running the program. You could come up with alternatives to keep the program from aborting, or locking the program up from access until a fix can be found later.

The solution to this problem was simple and required no action on the part of the Response Center. Under MPEXL 2.2, the program can no longer be run to monitor system activity. A supported method would be to use another product such as HP Glance/XL designed for the particluar release of the operating system included on the subsys tape. For our particular crash, the answer was to purge the program and eliminate the risk of ever running the program again. Problem solved, little downtime and a reduced window for failures.

Some additional commands you may want to use for further data gathering on the suspect process that caused the abort are:

UI_CIHISTORY
Displays the redo stack of the MPEXL commands. This is only limited to the limit set for the variable HPREDOSIZE.

UI_SHOWVAR
All system pre-defined and user defined variables.

UI_TEMPFILES
Temporary files either open or closed for the session.

UI_USER
Fully qualified session_name,user.account,homegroup.

In conclusion, you should feel free to look at the dump to identify the possible cause to your problem. Many times, these problems are self-inflicted by programs created yourself. Sometimes the problems are from programs contributed from other sources that are not compatible with the release of your operating system. Be aware that any upgrade to MPEXL could cause perfectly happy programs to become vicious little monsters to active systems. And sometimes, you just can't find a reason for the failure. If the latter is true, let HP do the research for you. They are the ones with the resources, knowledge and people to find the solutions for you. Use the information you gather from the dump as as an assistant would to the master carpenter. Point out to HP what was going on at the time, but don't be telling them what caused it. Your help can save you and them time in finding a resolution to your problem. And for you, a system that will stay at a higher level of uptime for happier users.

Paper # 3104

Technical Evaluation of Relational Technology on HP 3000/950 at Mohawk College.

Robert Hilverth, *I.S.P.*
Systems Analyst,
Systems and Programming Dept.
Computer Services Division
Mohawk College of Applied Arts and Technology
P.O. Box 2034,
Hamilton, Ontario Canada
L8N 3T2

Phone:(416) 575-1212 x3052
Fax:(416) 575-2334

I. Introduction

This evaluation resulted from a recommendation placed before the Mohawk College Computer Steering Committee by a financial software acquisition team. They wished to quality assure their selection of financial package which required a specific relational technology upon which to run. The Systems and Programming Department was requested to examine the suitability of the branded relational technology in the Mohawk College administrative computing environment.

Each DBMS has unique performance characteristics which, when matched with the "appropriate" computer, provides the necessary information throughput to carry on the business of the organization. The primary intent of this evaluation was to quantitatively determine whether the identified brand name relational technology running on an HP3000/950 would allow us to meet the currently established users' expectations for information throughput. For comparison purposes the current Mohawk College Turbo Image / 4GL environment was used as the baseline.

The intent of this paper is not to identify a particular product and show its superiority/inferiority compared to another product, but rather, to present the process which actually was used to do the evaluation. Therefore, in the remainder of this paper the word RELATIONAL should be interpreted as the name of a brand name relational technology and NETWORK/4GL to be the benchmark against which the comparison is made.

Support

The quantitative data and its interpretation was reviewed by the relational technology vendor, HP and another Ontario Community College, running the same relational technology, to ensure objectivity in reporting the results.

II. Method

Assumptions

Any process which runs on any computer system uses computer resources. These computer resources are the CPU, DISC and MEMORY. The quantity and mix of resources used determine the performance of that process. Processes that use less of any of these resources are better than processes that use more of these resources.

Design of comparable test systems (NETWORK/4GL and RELATIONAL)

To make the tests meaningful in our environment, we chose to take a subset of our existing student database and restructure it into a form where all "like" data is together, and repetitive (redundant) data exists once only. This is called "3rd Normal Form", and is the desired way to store information in any network or relational database. The following data was chosen for our tests:

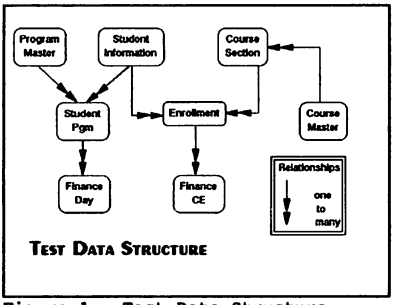


Figure 1 - Test Data Structure

COURSE-MASTER (12,560 records)
Data for each course offered at Mohawk. Includes the course identifier, fees, possible grades and course title.

PROGRAM-MASTER (227 records)
Data for each program of study offered at Mohawk. Includes program identifier and program title.

COURSE-SECTION (8,941 records)
Data for each section of a course offered at Mohawk. Includes course and section identifier, term, course controller, campus, start/end date, enrollment counts, instructor name, fees and discounts etc..

STUDENT-INFORMATION (9,926 records)
Data for each student. Includes Mohawk Id, first and surname, birthdate, address etc..

STUDENT-PGM (1,804 records)
One or more data entries for each full-time day student. Includes a program of study identifier, status in the program, mature student signal etc..

ENROLLMENT (19,989 records)
Multiple data entries for each student who has ever registered in a continuing education course section. This dataset (table) ties a student to a specific course section. Includes course identifier, section identifier, student identifier, term, status in the section, grade achieved, etc..

FINANCE-CE (9,259 records)
Multiple data entries for each ENROLLMENT which identifies course fee information. Includes course, section and student identifiers as well as the amount and fee type.

FINANCE-DAY (4,059 records)
Multiple data entries for each STUDENT-PGM which identifies fees paid for a specific term in any program of study by a student. Includes student and program identifiers, term, amount and the fee type.

Tests

Two different types of tests were conducted. Measurements were taken of RELATIONAL and NETWORK/4GL in a "head-to-head", single-user mode with normal daily processing running simultaneously. Measurements also were taken to determine RELATIONAL's multi-user capabilities.

For the single-user test we measured the length of time (elapse) each test took and the amount of time the process was active (CPU time). Elapse time could be measured to the nearest minute and CPU time could be measured to the nearest second. In addition, during the test period, "snapshots" were taken of system performance using GLANCE. Every 30 seconds GLANCE provided data on the existing mix of processes running. Of interest was the percentage of CPU usage for the NETWORK/4GL and RELATIONAL processes as well as the total CPU utilization. At the same time, measurements were taken for NETWORK/4GL and RELATIONAL which showed the amount of memory each process was using and the rate of disc access.

Specifically, the tests carried out were: Test 1 (A-H), load the eight datasets (tables) with data; Test 2, change a date updated field for all records in the ENROLLMENT table; Test 3 (and 3A), count the number of students who have no ENROLLMENT information; Test 4, report the essential data needed to create a course section list in section/student surname sequence; Test 5, report the essential data needed to create a student transcript; Test 6 (and 6A), report the accumulated fees paid for each program of study; Test 7, report the accumulated continuing education course fee paid by each student; Test 8, delete data from all tables.

The multi-user RELATIONAL performance evaluation was done to determine, if possible, what the effect of many RELATIONAL users would be on system performance.

The multi-user test initiated some of the same tests from the single-user trial. Each of six RELATIONAL users initiated the same stream of tests at stepped intervals.

Each user and test component was timed, and GLANCE was used to capture performance statistics. Again, this test was performed with normal day time activities carried on in the background.

To ensure that the result represented fairly the capabilities of RELATIONAL in the Mohawk College environment, RELATIONAL was given the opportunity to rerun any test. If the results were deemed to be below those expected by RELATIONAL, tuning of the code was allowed and the test was re-run.

Once graphic representation of the raw data was completed the results were examined by Computer Services staff from another Ontario Community College. Their input was sought because of their experience with RELATIONAL on an HP3000/925. In addition, our Hewlett-Packard Systems Engineer and RELATIONAL's Systems Engineer examined our results. In each case, we requested clarification on the interpretation of the results achieved by our testing.

Measurements for Single-User Tests

Head-to-head performance testing was carried on during the business day. Both the RELATIONAL and NETWORK/4GL tests were initiated at the same time, and were competing for resources against each other, as well as against the other normal daily processes that were running. Therefore, the comparison of RELATIONAL to NETWORK/4GL is running with the same background activity.

Each test was designed to measure DBMS performance, not report writing capabilities. Attempts at producing "pretty" reports were avoided. Only essential data was reported in the simplest form possible.

RELATIONAL was provided every opportunity to show peak performance. A software support engineer from RELATIONAL was allowed to tune any code for tests which showed "poor" results and was allowed multiple tries in achieving RELATIONAL's "best" results.

Testing was broken down into 3 major activities over the 3 day test period.

The first activity was to load data from existing MPE files into the target data structures. Each of the previously identified data sets (tables) were loaded individually. The tasks were started on two separate terminals at the same time and a third terminal was used to gather the GLANCE statistics. The load process was started on the 1st day and carried into the morning of the second day.

The second activity, once data had been loaded was to manipulate it and generate some common reports. The second half of the 2nd day was used to measure the results of data manipulation and reporting.

The third activity was to delete the data from the respective NETWORK and RELATIONAL data structures. This activity was carried on during the third day of testing.

Measurements for Multi-User Tests

A separate day was set aside to determine the effect of multiple RELATIONAL users on the HP3000/950. For this test 6 RELATIONAL users were logged on and ran a subset of tests used in the single-user tests. Each RELATIONAL user was logged on at stepped intervals. The measurements taken identified the interactive effect of one RELATIONAL user on other RELATIONAL users.

III. Results from Single-User Tests

Over the 3 day test period, no significant difference could be detected in the length of time (Elapse minutes) and the amount of processing time (CPU seconds) consumed, with the exception of 2 tests.

Test #2 (Date Update) and test #3A (Course Section List) did show some significant differences. In test #2 RELATIONAL used significantly less CPU time and completed the task 4 minutes faster than NETWORK/4GL. In test #3A, a variation on test #3, an alternate style of coding in RELATIONAL produced significantly poorer results than NETWORK/4GL. RELATIONAL test #3A ran approximately 1 hour longer than the NETWORK/4GL equivalent. As test #3 did the same task more efficiently, test #3A was not included in the overall analysis.

Figures 2 through 5 identify the aggregate and daily comparisons of Elapse and CPU time for all single-user tests. RELATIONAL's performance was surprisingly good. Historically, relational technology was known to be slower than other DBMS technologies. RELATIONAL seems to have overcome this.

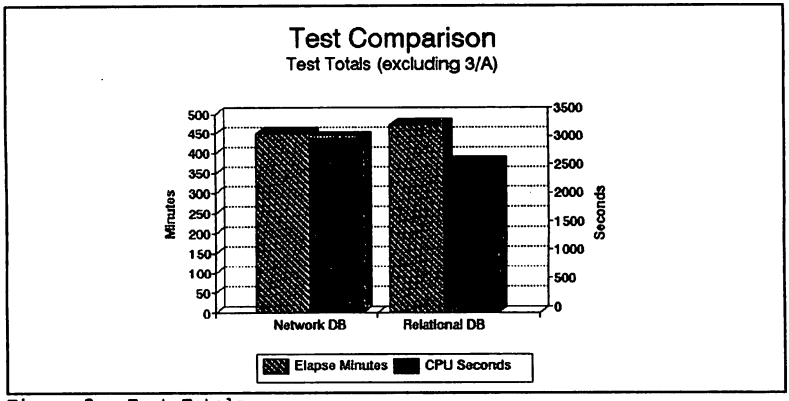


Figure 2 - Test Totals

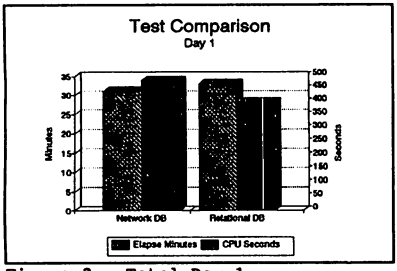


Figure 3 - Total Day 1

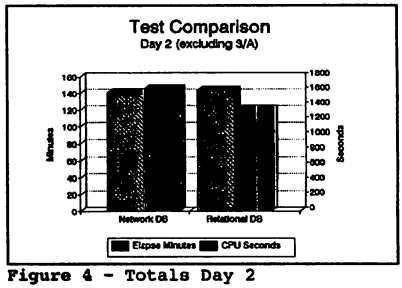


Figure 4 - Totals Day 2

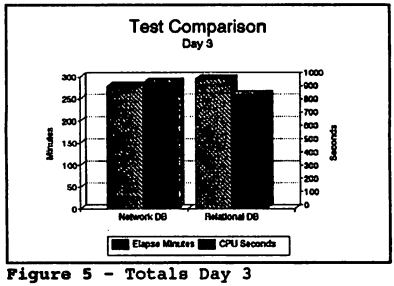


Figure 5 - Totals Day 3

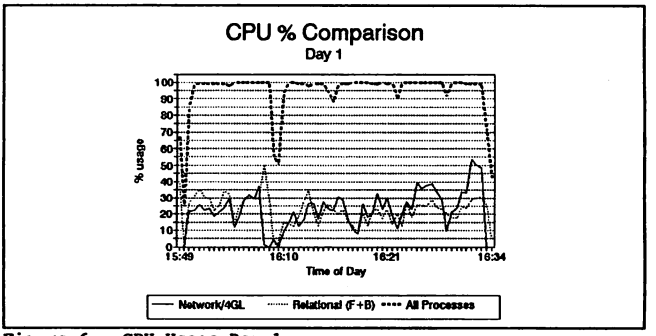


Figure 6 - CPU Usage Day 1

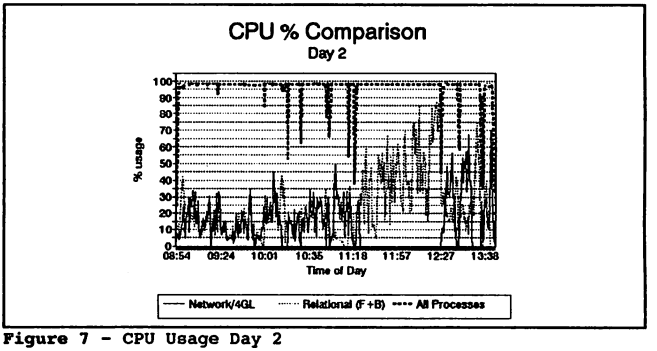


Figure 7 - CPU Usage Day 2

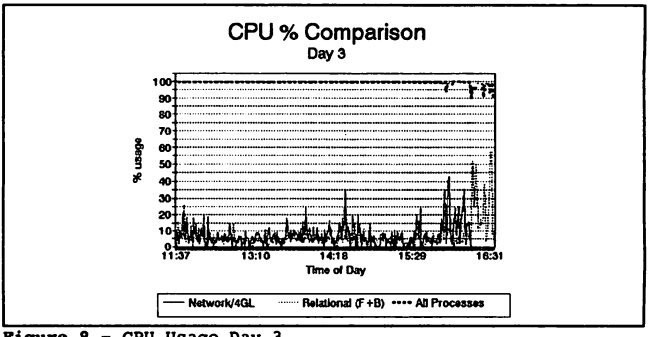


Figure 8 - CPU Usage Day 3

Central Processing Unit (CPU) Usage

During the 3 day test period, the HP analysis tool, GLANCE, was used to capture performance information every 30 seconds.

On a day by day basis, the comparison (as a %) of CPU time used by NETWORK/4GL and by RELATIONAL tended to be the same. For the purposes of this daily analysis, the two components of RELATIONAL (Foreground and Background) have been combined. The RELATIONAL foreground process can be directly attributed to the activity of a particular user. The background processes receive data from the foreground processes (multiple RELATIONAL users) and actually read and write data to the database, as well as performing various data maintenance/audit tasks.

The heavy line, which tended toward 100% is the total system wide CPU activity. On day 1 and 2 there were very short periods of time when total CPU capacity was not being used. On day 3, only towards the end of the day, was CPU activity reduced. This high level of CPU activity is considered "normal" for an HP3000/MPE XL system.

On day 2 between 11:18 and 12:27 test #3A was running. Due to an inefficient coding style, the RELATIONAL process ran for about an hour after the completion of the equivalent NETWORK/4GL task. After 1 hour the RELATIONAL test was aborted.

The trend shown by the graphs (figure 2 - 8) confirms that RELATIONAL performs similarly to NETWORK/4GL when comparing Elapse time and CPU time, running as a single user.

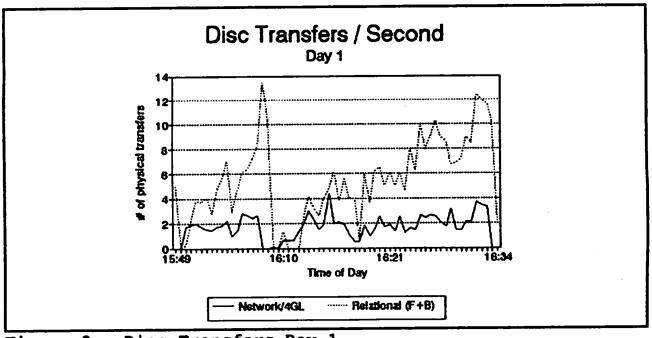


Figure 9 - Disc Transfers Day 1

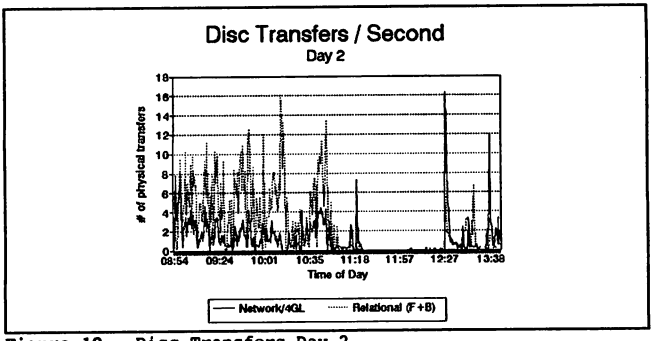


Figure 10 - Disc Transfers Day 2

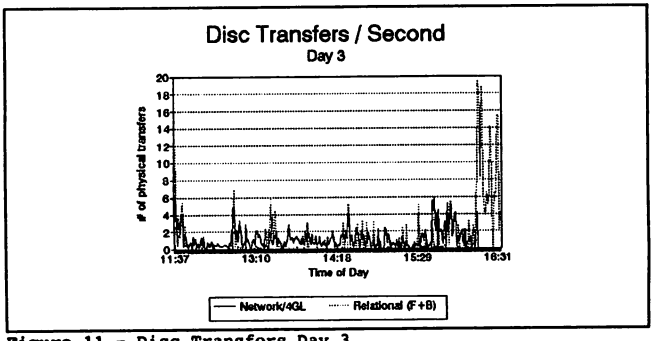


Figure 11 - Disc Transfers Day 3

Physical Disc Activity

Significant differences were detected on all three days of single-user head-to-head testing of RELATIONAL and NETWORK/4GL when comparing the rate at which processes make use of disc.

Over the 3 test days the rate of physical disc transfers per second tended to be 3 times higher for RELATIONAL than for NETWORK/4GL.

On day 1 and day 2, until 10:54, data was being loaded into both databases. It is during this phase of activity that the greatest difference can be seen between RELATIONAL and NETWORK/4GL.

From 10:54 on day 2 an update process and several report processes were run. The degree of disc interaction for both RELATIONAL and NETWORK/4GL was reduced.

On day 3 the data in both databases was being deleted. Although the difference between the RELATIONAL and NETWORK/4GL is not as great as loading data, the RELATIONAL rate of disc transfers is higher. Toward the end of the trial, the NETWORK/4GL process finished before the RELATIONAL process. When there was no longer any contention for disc drive 1 (both databases were on the same disc drive) the rate for RELATIONAL dramatically increased.

Upon checking the individual test results, the high disc transfer rate can be attributed to the RELATIONAL background processes. These processes actually write data to the database as well as maintain several other data files for audit and recovery purposes. When compared to the activity of NETWORK/4GL, RELATIONAL is actually doing more reading of and writing to disc.

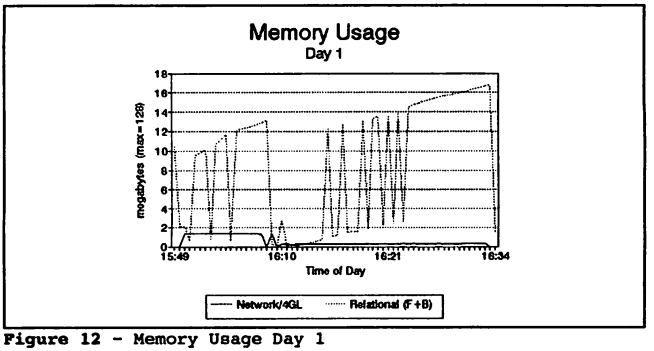


Figure 12 - Memory Usage Day 1

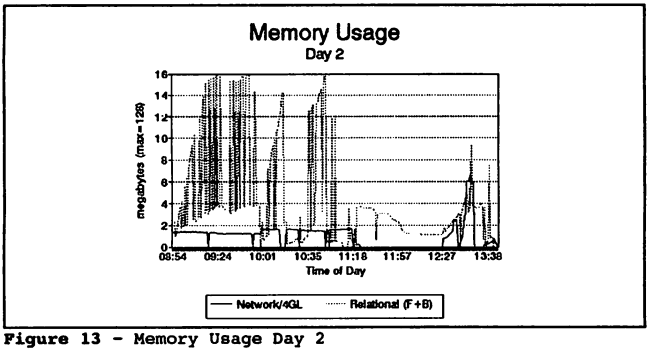


Figure 13 - Memory Usage Day 2

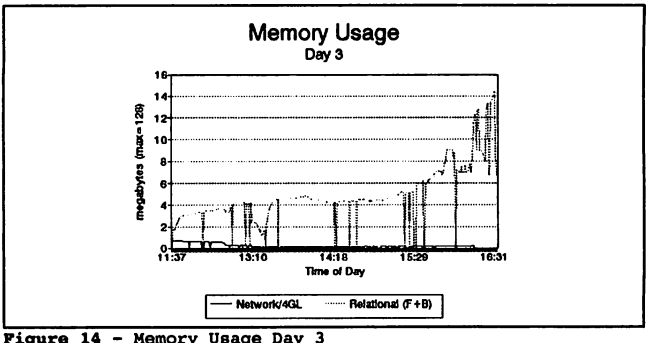


Figure 14 - Memory Usage Day 3

Memory Utilization

As with physical disc activity, significant differences were detected in memory usage on all three days of single-user "head-to-head" testing.

Over the 3 test days, the amount of memory usage tended to be 5 times higher for RELATIONAL than for NETWORK/4GL.

There was a high degree of correlation between high rates of disc transfers and high points of memory utilization for RELATIONAL.

Memory utilization for RELATIONAL tended to be highest and sustained at that level, on day 1 and until 10:54 on day 2, when data was being loaded into the database. Memory usage tended to be lowest when reporting processes were running during the second half of day 2. The delete process running on day 3 had moderately high memory usage until the NETWORK/4GL process had finished. At that point there was a gradual increase in the amount of memory used by RELATIONAL.

Running the functionally equivalent tests head-to-head, we found that the RELATIONAL product and NETWORK/4GL seemed to perform similarly. The length of time (elapse minutes) the comparative test took and the amount of Central Processing time (CPU seconds) tended to be the same. These results would suggest that there is no significant difference between NETWORK/4GL and RELATIONAL. During the running of the head-to-head comparisons we used a performance evaluation tool (GLANCE) which was set to take a performance "snapshot" of CPU usage, DISC activity and MEMORY usage every 30 seconds. Using these "snapshots", some significant differences were detected. Over the 3 day testing period, comparisons of CPU utilization (as a % of the total available) showed no discernable difference. However, physical DISC accesses and MEMORY usage did show marked differences. On average the DISC activity (physical transfers per second) is about 3 times greater for RELATIONAL than for NETWORK/4GL. Similarly, on average the MEMORY usage (megabytes) is about 5 times greater for RELATIONAL than for NETWORK/4GL.

In the single-user head-to-head comparisons the RELATIONAL processes tended toward significantly greater DISC and MEMORY usage and tended to be equal in CPU usage.

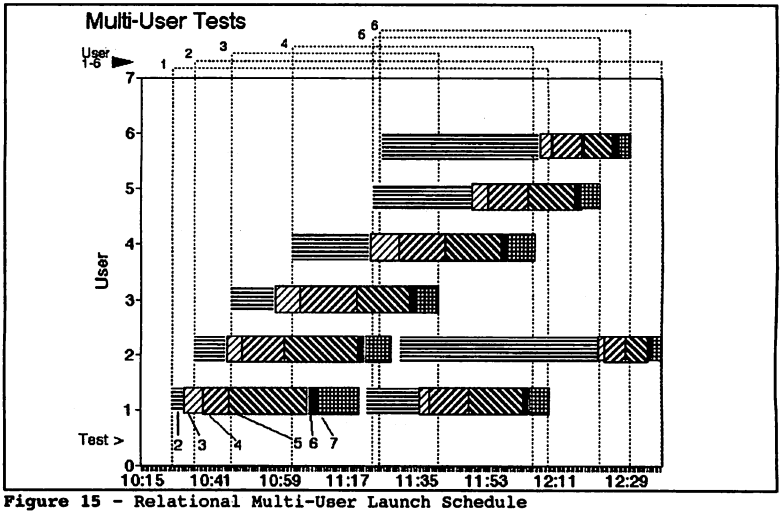


Figure 15 - Relational Multi-User Launch Schedule

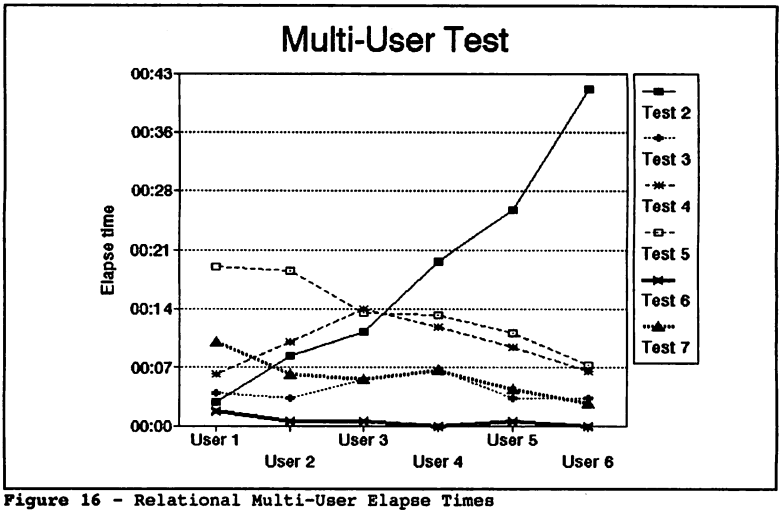


Figure 16 - Relational Multi-User Elapse Times

IV. Results from Multi-User Tests

To determine the effect of running multiple RELATIONAL users, the same series of tests (test 2 through test 7) were run by 6 users. Each user initiated the test sequence after a set interval had elapsed. Figure 15 identifies the sequence of tests run, and the start/end times for each of the 6 RELATIONAL users. For User 1 and 2, the test sequence was repeated but the second occurrence is not included in the elapse time evaluation presented as figure 16.

Tests 3 through 7 reported on data in the RELATIONAL database. Test 2 was an update request. The elapse time data for tests 3 through 7 suggests a longer elapse time for the first user and a decrease in processing elapse time for subsequent users. Test 2 shows the opposite trend.

Tests 3 through 7, because the request is repeated verbatim, stores the request in a buffer area. Subsequent users who request the same query do not require that the query be re-interpreted. Therefore the request is processed faster.

Test 2 required an update to a date field for all records in a table with a unique value for each user. Since each request was different, RELATIONAL had to re-interpret each request prior to processing it. Consequently, each user from 1 through 4, had longer and longer elapse times for test 2.

By the time that User 5 had started test 2, User 1 had restarted test 2 for the second time. User 1 had locked the table that required the update and User 5 had to wait until User 1 had finished.

Similarly, User 6 had to wait until User 5 had finished and User 2 had to wait until User 6 had finished. Because the entire table is locked, no other User/process could complete its task. This would be true even if each process needed to update different records.

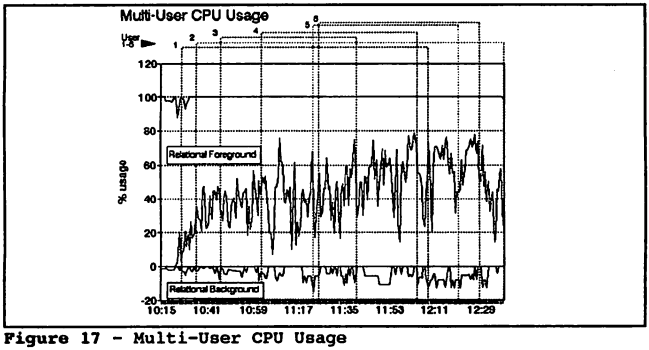


Figure 17 - Multi-User CPU Usage

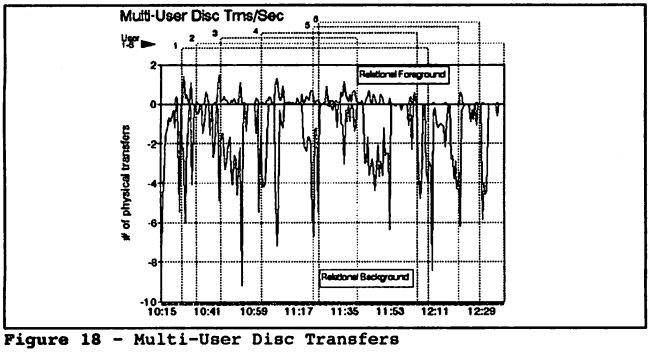


Figure 18 - Multi-User Disc Transfers

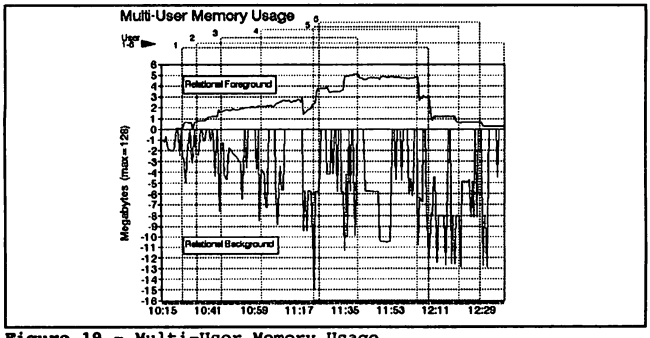


Figure 19 - Multi-User Memory Usage

Multi-User RELATIONAL CPU, Disc and Memory Usage.

There was a need to be able to uniquely identify the RELATIONAL Foreground and RELATIONAL Background related functions for CPU, Disc activity and Memory usage. To clearly show each over time, the RELATIONAL Background values were multiplied by -1. Of particular interest is the effect of multiple RELATIONAL users on the amount of memory used by RELATIONAL as a whole.

CPU Usage

Total system wide CPU usage was consistently close to 100%. The foreground processes, composed of user test requests, were significantly higher than the background processes which consume very little CPU. These CPU usage curves represent "normal" activity. If there is any available CPU then the system consumes it by processing faster.

Disc Transfer Rate

The bulk of the disc accesses are done by the background processes. Although the interpretation of the graph is difficult, there is a tendency for the background disc function to mirror the background memory usage. Whenever background memory peaks, background disc transfer rates increase. Whenever background memory sustains some level, background disc transfer rates sustain a correlated level.

Memory Usage

Memory usage for the foreground activity tends to show a step function, incrementing the amount of memory being used for each additional user. Increasing to 6 users from 10:15 to 11:35, then decreasing to 0 users from 11:35 to the end of the test, there is a step-like rise and fall in foreground memory usage. The background memory usage consumes more memory but tends to grow gradually until 11:17, then remains relatively level, with no decline until there is only 1 user remaining active at 12:29.

V. Conclusions

RELATIONAL's database management technology is more complex than the 2 level network technology used by Image. We were impressed by the speed at which data was processed by RELATIONAL. Traditionally, relational database management systems tended to be slow in comparison to "hierarchical" or "network" databases. RELATIONAL was able to prove to us that its single-user processing speed (Elapse and CPU times) was as good as NETWORK/4GL's speed.

However, processing speed was only one of three criteria being evaluated. On the HP3000/950, there is a top limit of 128 megabytes of memory. About 1/3 to 1/2 of this is used by the system for the management of the system (operating system, program code, program data stacks and extra data segments/objects). Assuming the worst case, the remainder of about 64 megabytes is set aside for file space. This must be shared by all system users. It is our belief that RELATIONAL's high of 16 megabytes usage is an indicator that our current 950 model of HP3000 is not large enough to support our norm of 70 to 90 concurrent users, if all the processes they execute would be RELATIONAL processes. The multi-user RELATIONAL test shed a little more light on memory usage. We believe that the high rate of DISC transactions for RELATIONAL is a byproduct of the large amount of memory being utilized.

The high rate of physical disc transfers also may be an indicator of poor performance. The HP3000/MPE XL systems use a "virtual memory" technology to allow many users to share the same memory space. When the operating system determines that a portion of memory has not been used for some time, it swaps that memory to disc, so as to provide another user with memory space. When the process that originally was using the swapped out memory requires it again, the second user's memory is moved to disc and the first user's memory is returned from disc. The large amounts of memory required by RELATIONAL and the high disc transfer rates that are evident, suggest that adding a large number of RELATIONAL users would cause the computer system to do more "virtual memory" swapping, and less user processing. This would lead to slower elapse times for the tested processes.

The multi-user test provided more information on how the RELATIONAL foreground and background processes effected memory usage, and system performance.

By executing the same series of processes by a number of users, several problem areas were illuminated.

In general, CPU utilization, DISC transfers and MEMORY use tended to follow the pattern set by the single-user tests. Inquiry test components tended to be marginally slower (elapse time) for the first user than the rest. This is due to the first user interpreting an inquiry and subsequent users processing that interpreted inquiry. The update test showed increasingly longer elapse times with each subsequent user. When one user overlapped another user processing the update test, the table locking strategy, which prevents 2 or more users from updating the same data table at the same time, caused significant increases in processing time. One user had to wait for the previous user to finish the task on the data table before access was allowed.

Multiple users can cause processing bottlenecks due to their interaction with one another. As more users make unique queries, the longer the elapse time for the completion of each query.

When multiple users are updating, creating or deleting records in the same table, each must wait in a queue for their turn in processing against that table. In a production transaction processing environment this is not an acceptable locking strategy.

In general, most of the existing computer power on our HP3000/950 is used for transaction processing. The minority is used for retrieval of data for reporting only. Characteristically, transaction processing requires the accessing of information to determine if it exists. If it is present, the data may be updated or deleted. If it is absent, the data may be created. These update, create, delete activities require the data to be locked while the process is active. The narrower range of data that is affected by the lock the better. Ideally, only those bits of data that are being changed/deleted/created should be unavailable to other users. Currently, RELATIONAL's locking strategy is to lock an entire table (grouping) of data.

RELATIONAL processing consists of two components: foreground processes which interact with the user, and background processes which control data retrieval, storage, and audit/integrity. In the multi-user test each new user required just under one megabyte of additional memory for the foreground process. Our consultation with another Ontario Community College suggested that our test results were on the low side. Their experience suggested that 1.5 megabytes foreground per RELATIONAL user is more likely. The memory requirement for the background processes, which are shared by all RELATIONAL users peaked at 15 megabytes, but tended on average between 10 and 12 megabytes. Given the worst case of 15 megabytes for the background and 1.5 megabytes foreground for each RELATIONAL user, a serious processing bottleneck could occur with 30 to 35 "heavy" data processing users. With "lighter" users this number may be doubled to 60 - 70. In our estimation, at this point our existing HP3000/950 computer system would be spending more time swapping memory to disc and back than actually processing users' tasks. This could translate into unacceptable response times for the our computer clients.

Examination of the data suggested that the HP3000/950 is not large enough to support our current number of users if all processes running would be RELATIONAL. In our estimation the tested version of RELATIONAL would not meet performance expectations required to carry on the College's business.

#3105
Mainframe Software Management Techniques: What Every HP 3000 User Should Know

Betsy Leight
OPERATIONS CONTROL SYSTEMS
560 San Antonio Raod
Palo Alto, California 94306
(415) 493-4122

INTRODUCTION

Today's HP 3000 professionals are becoming increasingly concerned about software and data integrity. And as a result, implementing a reliable method of tightening control over the changes to systems and applications has become a prime objective of many shops.

An isolated change may initially seem harmless, but the ripple effect upon interrelated systems can often be devastating. As the volume of code used in a shop becomes greater, the effect of these change can become staggering. With an industry average cost of $50 per line, the software investment of even a small site can be valued well into the millions of dollars.

Despite the enormous value of today's software applications, much of it is vulnerable to inadvertent errors. The reason: MIS departments often use tedious manual procedures to track program changes, and these procedures are seldom if ever followed faithfully throughout the organization. Without an improved approach to monitoring how and why changes are made, a company's entire computer and software asset are at serious risk.

First, consider the case of our firm, Operations Control Systems. With over 4000 products installed throughout the world, our customer support staff receives requests for dozens of modifications every month. At the same time as these maintenance and enhancement requests are coming in, our R&D team is also under pressure to develop new products. Shipping a new release represents a major project for almost every department within the company. Without an efficient means to manage changes, maintenance work would forever delay progress on new development. Through the task of managing our own internal software maintenance and release management process, we at OCS developed much of our initial expertise in the area of change management.

Through feedback from our customers, we started to refine these techniques and make them available to the general HP user. When we first surveyed our customer base in 1986, we found that the application maintenance backlog was not unique to software developers. Some customers who relied entirely on vendor-supplied software with seemingly minor modifications ran as many as 6 months to 2 years behind schedule on their maintenance work. Thus, while many non-data-processing executives think that their MIS staff is focusing on new applications, the reality may be that MIS is bogged down in maintenance and change related activities.

This highlights a second major need for change management. Change management improves the ability of the MIS department to react to user demands for program changes.

THE NEED FOR CHANGE CONTROL

In order to understand the change control problem and how to solve it, let's explore the nature of the problem in an uncontrolled environment. The problem stems from the fact that in many HP3000 shops, programmers can access production source directly. This can have several consequences.

First, consider an environment where multiple programmers make simultaneous changes to source code files. In this situation, the last programmer to update each file overwrites the changes made by other programmers. The result is frustration and wasted time synchronizing and retesting all of the changes.

Second, a careless programmer might inadvertently destroy the source code of a critical application. If a production error occurs and the source code cannot be restored easily, production could be delayed for hours or even days. The original author may not be available. The program may even have to be rewritten from an old copy, wasting development and test time and possibly reintroducing old bugs or creating new ones.

Third, the development procedures found in many shops eventually result in discrepancies between source and object code files. That is, the master (or official) source files may not recompile into the current production object files. Since it is almost impossible to recreate source from object code, days might be spent searching for the correct version, recompiling it and validating it against the current production code. And, with all of this effort, there will be no guarantee that the source will exactly recreate the current production code.

These situations point out a fundamental rule of good change control: Programmers should never have unrestricted access to production files!

At first glance, these problems may appear obvious, but the fact is that many HP 3000 data centers continually react to the problems they cause, rather than introduce standards and controls to avoid them. After all, a data center is there to support its users. This charter forces management to focus resources on meeting daily user requests, distributing reports, managing hardware and completing batch production, rather than on implementing long-term solutions to problems that are not well understood. Although many shops can run error free for months, an innocent looking error can snowball into a major catastrophe.

If this is the orientation of your shop, you might want to consider this fact: the efficiency and accuracy of your work depends upon the integrity of your software assets. Is your data center a candidate for better change control? Try taking the following test.

What software systems do you have?
Where and why were the last changes made?
What programs comprise these systems?
Where are the appropriate backup copies?
How often do these programs change?
Which user requested the last set of changes?
Who is responsible for the changes?
What is the status of a specific change request?

If you cannot answer all these questions readily, your shop probably needs a better approach to managing changes.

Proper change control provides immediate answers to these critical questions. The result is more than just visibility and auditability. It is more than data integrity and improved maintenance efficiency. A good change control system ensures that every program change is authorized and controlled, making it possible to quickly restore systems when necessary. Change control techniques also make it possible to monitor changes made to software from the earliest stages of development through the testing phase and all the way through the move to production process.

Change control also improves the Quality Assurance process. It allows operations personnel to test new systems, moving software from development to test, or from test to production. And, all of this will be done with complete audit trails that allow the identification of each entity promoted and tested.

Another important aspect of change control is provision for management inquiry in areas such as: the status of a change, the current version of a file, the change history of a file, and the impact of a proposed change. Supplying this information can often be a tremendous benefit to managing the software life cycle.

RECOMMENDED CONTROLS FOR ALL SHOPS

In order to achieve the benefits of change control, a shop should address each of the following five major categories:

Control through stages: Keeping control over the movement of software through the development and maintenance process.

Change history: Providing a complete history of every change requested and made to the source code.

Control over source and procedures: Maintaining control over users authorized to access source code and the processes that render source into executable form.

Security: Retaining control over the personnel who are authorized to make program changes.

Inventory: Maintaining an accurate inventory of all the programs, files and other components of each system.

A more detailed checklist identifying good change control practices appears at the end of this article. With these items in mind, lets look at the costs and benefits of manual change control techniques.

MANUAL CHANGE CONTROL

HP 3000 users have developed a variety of creative procedures for controlling changes in their environments.

A common strategy requires programmers to FCOPY source code from the production location into a development location. This strategy can be implemented at three levels.

1. The development area may be nothing more than a set of individual groups where files reside. In this case, each programmer copies all of the necessary files into his own group where he makes the appropriate changes. There is no standardized testing environment.

2. Separate account structures may be maintained for production and development. Often a separate account structure is maintained for testing as well, to duplicate the production account structure. Thus, each programmer can be confident that testing is conducted in an environment that closely resembles production. Establishing separate account structures on a single computer often produces adequate results.

3. A separate development computer may be used, thus eliminating the possibility of direct access to master files located on the production computer. Although this approach is ideal, it is not possible without a separate development computer.

Unless stringent controls exist, however, this strategy does not guarantee that only one individual can check out a particular file at a time. And, since programmers are not restricted from accessing production accounts, there is no way to audit their access.

Once the development is completed, many shops allow the same programmer to test their own work and simply overlay the original production files with the new version of code. Since such a procedure seldom represents adequate control, it is widely suggested that a formal Quality Assurance (Q/A) process be used.

There are several ways to initiate such a process, depending upon the resources that are available. Smaller companies may have programmers Q/A test their colleagues' development efforts. In these cases, tests are usually performed in the development account.

Larger organizations may dedicate one or more individuals whose sole function is testing. Here, a separate Q/A test account is generally set up to reflect the production account structure. In this case it is vital that the developer is finished, all claims to the code are relinquished. After it has been moved to Q/A for testing, the one copy of the code will exist only in Q/A. If the Q/A analyst locates an error, the code will be returned to the original developer for revision. It is now Q/A's turn to relinquish all claim to the code by moving it back to the development location and purging it from Q/A. If simultaneous copies were to exist in both locations, last-minute changes in development might not be included in the Q/A version. Thus, the final production version would not be accurate. It is surprising how often this obvious safeguard is overlooked.

At the conclusion of the Q/A phase, another step will often exist. A higher level manager will perform a final approval on the development code to certify that all standards have been met and all tests have been satisfactorily completed.

Following final approval, the finished code is ready to be moved into the production location. Since the new files will be copied into production with the same names as the original files, the original files must first be backed up to tape to avoid loss of the earlier version. Next code must be recompiled to assure an exact match between source and object. Lastly, JCL and other files must be updated.

This update process is tedious, time-consuming and error prone, especially when large numbers of files are involved. Nevertheless, precise standards cannot be eliminated at this final stage or the shop will run the risk of serious inconsistencies.

Based on the previous discussion, it is possible to state several general rules for good change control:

First, establish separate accounts for production (master files), development and, if possible, test. Test accounts should be mirror copies of production. This allows files to retain their original FILE.GROUP names as they are moved from test to production and makes it possible to visualize the link between a developing program and its production/master version.

Second, when files are checked out" from the production location to development, a copy should be made. The original source should never be destroyed. However, when development is complete, files should be moved to the test location, thus eliminating synchronization problems with old versions.

Third, when Q/A 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. If system loads are heavy, such updates could

occur in batch and should be initiated by operations or a production Librarian. The timing of the release into production should be coordinated with the user's schedules, with the release of other modules and with documentation changes, etc.

Prior to updating the library, the original production copy should be verified and stored. Without this step it is much more difficult to return to a prior version in the case of a problem. As a general rule, we recommend that the authority to release files to production be restricted.

AUTOMATED CHANGE CONTROL

Although this article has described the minimum requirements necessary to achieve reliable change control, implementing these controls places a considerable burden on everyone involved in the change process . . . unless the process is automated.

Although an automated change control system will require some initial planning, once it is operational, the burden on the MIS staff will actually decrease. The process would be as follows:

The first step in implementing an automated change control system is to define your development environment. Decide what files you want to control, what types of file movements will be performed, what approvals will be required at which stage, and who will be authorized to perform or authorize each type of file movement. This will depend on such things as:

The size of your development staff
The levels of management and job responsibilities in your development group
The size, complexity, and volatility of your applications
The amount of packaged software used by your firm
The physical hardware configuration of your computers
The degree of control which you and your auditors agree is appropriate.

The following examples show four typical development environments, presented in order of increasing complexity.

As you read through them however, keep in mind that these are only intended to be representative examples; each shop is different. A good automated change control system must be configurable to meet the specific needs of the organization.

EXAMPLE 1 - BASIC DEVELOPMENT ENVIRONMENT

The basic development environment is a small shop with a single computer and a staff consisting of two programmers, one day-shift operator, and a "shirt sleeves" manager. The development control objectives are basic, but critical:

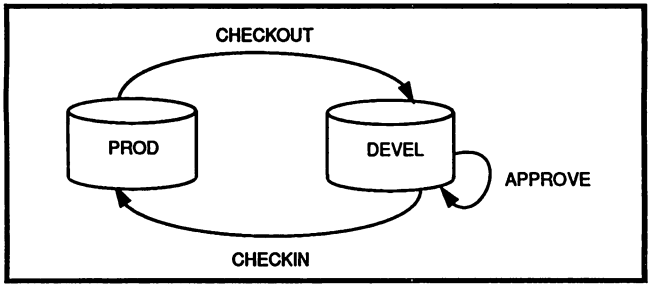
- They need to know where the current production versions of all source, object, and job files are,
- They need to be sure that all changes are made to copies of those files in a sepa rate test location,
- They need to assure that all changes are approved by the manager before they are put into production, and
- They need to do this without further burdening their staff who is already working long hours.

The first step here, as for most installations, is to identify all of the production source, object, and job files. These files need to be grouped together and secured. Through an automated change control system, this can be accomplished by creating customized filesets which represent whole groups of files. This eliminates the need to change the operating system account/group structure or move any of the files around.

The second step is to define the file movement policy, or steps, that will be allowed in the shop. In this example, we have three basic file movement steps:

1. A CHECKOUT procedure, which copies source, object, or job files from the master location into a development location,
2. An APPROVE step, which allows the manager to stamp a changed file with their approval, and
3. A CHECKIN step, which allows the programmer to move a group of files back into the master library when they have been approved. This step also makes an automatic backup copy of the old master file. In addition, the backup copy is compressed to a small fraction of its original size to save disc space.

This process is illustrated below:



To maintain the security of the master library using MPE alone, the CHECKOUT and CHECKIN steps would have to be performed by a manager, an operator or an additional employee. The approval step would have to be documented on paper. With an automated change control system, the CHECKOUT step can be defined to make test copies of the master files even though the master library is secured. The automated system can keep track of these copies and can prevent multiple programmers from checking out copies of the same file at the same time. The APPROVE step can be defined to mark the file(s) as approved, and can be restricted so that only the manager can use it. Furthermore, the CHECKIN step can be defined to allow the programmer to push his changed, approved files from his test environment into the master library without having to log on to the master account, and automatically archive the old versions.

EXAMPLE 2 - SEPARATE Q/A FUNCTION

This example illustrates a medium-sized shop with a tightly controlled development process. The MIS organization consists of a manager, six programmers, two operators, and a full-time quality assurance staff of two. Due to the critical nature of their applications, this shop insists that their Q/A staff perform full unit and system testing on every change. They also require management approval of the testing procedure before any new or changed software can be put into production. They do, however, need to allow their programming staff to make "quick fixes" on an emergency basis, thus bypassing the delays that would normally accompany the Q/A process. It is especially critical that a complete and reliable audit trail be maintained for these quick fixes.

We can define this environment as follows:

1. A CHECKOUT step as described in Example 1. This allows the programmers that only one copy at a time is modified (other concurrent copies can be made with read-only access).

2. A SUBMIT step, to allow programmers to move their modified source, object and job streams into the Q/A area.

3. A GETCURRENT step to bring read-only copies from the master library directly into the Q/A area. This allows the Q/A staff to perform an integrated test using the current production versions of programs that are not currently being changed.

4. Since not all changes will pass Q/A, we need a way to get the files back into the development location. We define a REJECT step which will be performed by the Q/A staff when a program fails testing.

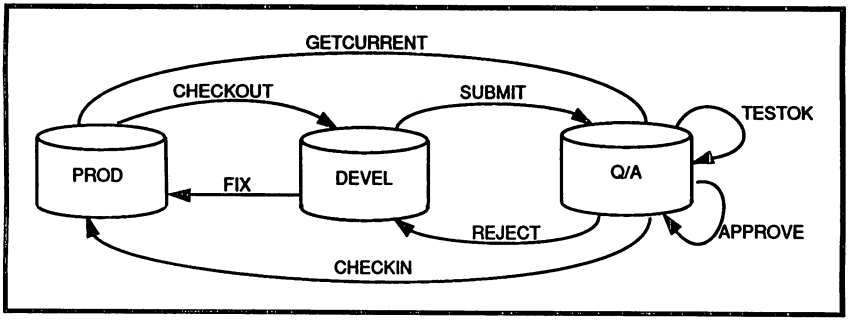
5. Q/A will signify that a change has passed its system tests with a new step called TESTOK. This will help the manager keep track of the status of work on various programs and is a positive indicator that testing is complete.

6. We still need an APPROVE step, but it is defined to operate on files in the Q/A location. The APPROVE step will be performed only after they have been TESTOK'd.

7. The CHECKIN step now moves files directly from Q/A to the master library, once they have been APPROVEd.

8. Finally, we define a FIX step, which moves files directly from the development area to the master library.

This development environment is illustrated below:



Here again, the automated system facilitates file movement, emergency fixes, and management approval without burdening their staff. The CHECKOUT, SUBMIT, REJECT, and CHECKIN steps are defined to operate on specific groups of files, so users do not need to fully qualify file references. Wild-card references may also be used to move groups of files at once. These steps can be defined to automatically purge the files from their original location after copying to eliminate any need for additional housekeeping efforts. Because an audit trail is maintained for each step, no extra effort is required to control use of the emergency FIX capability - the manager gains complete visibility by simply listing all uses of the FIX step.

EXAMPLE 3 - NETWORKING, SOFTWARE DISTRIBUTION, SOURCE/OBJECT SYNCHRONIZATION

The primary difference between this example and the two previous ones is that there are separate computers for production and development. In addition, the company's applications run on multiple remote computers. As in previous examples, the master library contains the production source, object, and job files, but in this case, the object and job files are actually executed on the various remote computers. This adds the important control objective of assuring that all of the remote sites are running the correct versions of the code. To this we will add the audit requirement of source-object synchronization: we must assure that the object files in the master library (and on the remote computers) were in fact generated from the source files in the master library.

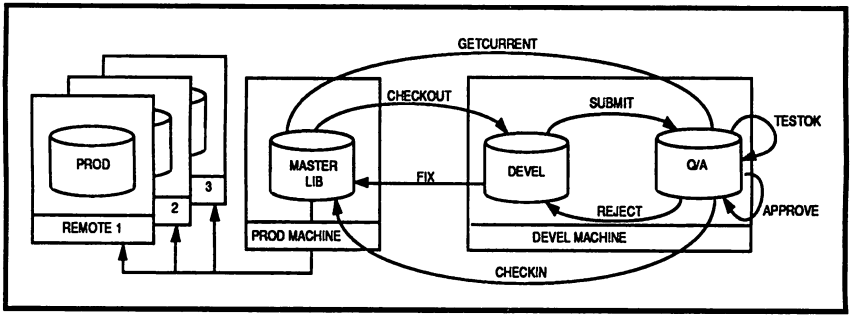
The rules and file movement steps for this environment are the same as for Example #2, with the following exceptions:

1. A new step, called RELEASE, is defined to distribute groups of object and job files to the remote computers. This step will copy the files specified to all of the remote computers, and produce an audit trail to verify that the copies were successfully transferred.

2. The CHECKIN and FIX steps will be modified to move only source and job files into the master library, and to automatically stream compile jobs to generate the object files from the new source. Since the new source has already been compiled to test the changes, this step is redundant, but it is an effective way of assuring that the source and object files always match and are synchronized.

3. While the remainder of the steps function as they did in Example #2, the automated system makes the multi-computer environment transparent to the development staff. Now CHECKOUT, GETCURRENT, CHECKIN, AND FIX all operate between the production and development computers without requiring extra steps or command.

The resulting environment is illustrated below:



Mainframe Software Management Techniques 3105-10

EXAMPLE 4 - PACKAGED SOFTWARE, ADVANCED VERSION CONTROL

In this example, the company uses third-party application software and receives periodic releases of the software from the vendor. Their own programmers have also customized portions of the software in-house. They have developed custom reporting and other extensions to the software. Whenever this shop receives a new release from the vendor, the new software is placed into its own separate account. There it is integrated with existing software by the programming staff, tested by the Q/A group and eventually put into production in the same manner as internally developed software.

Since the software is run on a large number of remote computers, the company releases newly approved production software in stages. First, a small number of remote users conduct a beta test for a limited period. When this stage is satisfactorily completed, the software is released to the remainder of the sites. Because of this procedure, the company needs to maintain two or more versions of the software simultaneously, and, in emergency situations, make changes to a prior version without interfering with work performed on the new version.

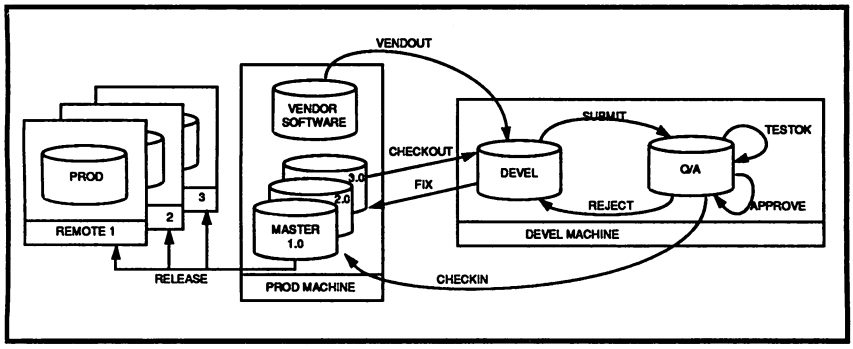
The rules and file movement steps defined for this environment are the same as Example #3 except for the following:

1. A new step, VENDOUT, has been defined for programmers to check out source and job files directly from the vendor software location. The step will prevent the programmer from accidentally getting an old version from a different location.

2. The CHECKOUT step is redefined to select the latest version of the software from the master library. This is accomplished by searching the "new release" location first. If the file is not found there, the "old release" location is searched. This assures that the programmers do not inadvertently use old versions of the software.
3. The CHECKIN and FIX steps are redefined to move files into the "new release" location.

4. The RELEASE step now operates on the version locations much as CHECK OUT does. When specific files are released, the automated system searches for the most recent version, and finds it even if it has not been changed for several versions. If a general release is made, only the files that have changed are distributed. This reduces tape and disc storage requirements as well as network distribution costs by eliminating unnecessary file transfers.

The resulting environment is illustrated below:



It is important to note that this rather complicated development environment can be precisely controlled with minimal effort through the careful choice of predefined file movement steps. Searching through two or more locations for the most current version can be performed automatically by the automated change control system. This enables the shop to maintain the integrity of prior versions while building a new version . . . without replicating files that have not changed. It also assures that programmers always get the most current version of any file.

SUMMARY

HP 3000 data centers have built a substantial asset in their software and data files. Yet, without effective change control, it is virtually impossible to guarantee the integrity of modifications to this valuable asset.

Furthermore, industry surveys show that most firms have a substantial software maintenance backlog. Without efficient means to manage changes, this backlog severely hampers the progress of new development efforts. As a result, it is becoming quite common for HP 3000 shops to establish formal procedures for software change control.

Although many shops initially attempt to implement change control through a manual system, these systems have met with limited success due to the heavy burden they place on the MIS staff and the inherent unreliability of the manual approach. Because automated change control systems provide an effective, reliable solution while reducing the burden on the MIS staff, they have recently become the standard. These automated systems improve development and maintenance efficiency and ensure that all program changes are properly authorized, documented and tracked.

Automated systems provide management with quick access to the status of changes, the change history of a file and the impact of a proposed change. They provide programmers with an environment that allows them to concentrate on programming rather than on searching for the correct versions of files and documenting their usage.

Safeguarding the integrity of the software asset while paving the way for more efficient productive development and maintenance, is one of the most compelling challenges facing HP 3000 professionals today. Automated change control systems can provide the tools to meet this challenge.

CHANGE CONTROL CHECKLIST

Change control procedures for computer programs should be established and followed. The intent of these controls is to prevent unauthorized, inaccurate, and unreliable program changes from being incorporated into the live production environment. Both scheduled and emergency changes must be appropriately controlled to maintain the ongoing integrity of software.

You can use the following techniques to ensure that proper controls are being maintained over your program changes:

- Develop and adhere to formally approved written standards for all program changes
- Define and enforce procedures detailing who can initiate and who can authorize program change requests
- Describe and track the nature and reasons for proposed changes Enforce testing and acceptance procedures for all program changes including emergency changes
- Test all program changes under normal operating conditions
- Involve users in preparing test data and reviewing test results
- Investigate and correct all errors before transferring code to production
- Certify that all test results demonstrate adequate protection from fraud, waste, and misuse of the program
- Document all program changes and update appropriate documentation as changes are made
- Log all completed changes as well as those changes in progress
- Utilize a formal system to report all changes to users and project managers
- Enforce a checkout-checkin procedure that prevents a file from being simultaneously modified by more than one programmer
- Develop procedures to analyze whether other systems are affected by new program modification
- Retain and secure original source code until changes have been processed, tested and updated
- Limit the frequency of program changes, except for emergency cases
- Notify both the user and EDP project manager when emergency changes are made

TITLE: Oracle RDBMS on HP 3000 - Narrow Tolerance Performance Tuhing Tips

AUTHOR: Mirek Zlotkowski

DCE Information Management Consultancy

Prisengracht 747-751; 1017 JX Amsterdam

Amsterdam, L8N 3T3

THE NETHERLANDS

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING.

PAPER NO. 3106

Paper Number 3107

ME AND MY SHADOW

by
John D. Alleyn-Day
Alleyn Day International
1721 M. L. K. Way, Suite 3
Berkeley CA 94709-2101
415-486-8202

Introduction

One of our clients uses the HP3000 for an extremely critical application that can tolerate virtually no downtime. They were upgrading from a Series III and they wanted a "hot backup" - a second machine networked to the first so that a copy of the operating data would be maintained on the second machine and continuously updated with almost zero delay. If the primary system failed, switching to the second machine was to be effected in a minute or two.

Requirements

To understand why we picked the methods that we did, it is necessary to know something about the system. The computer runs a single application with two major components.

The first major component consists of a set of input screens run in block mode and used by about twenty operators. At the busiest time, they each produce about one transaction every 4 minutes consisting of about 1Kbyte of data.

The other major component of the system is a batch program that runs continuously. This job runs a communication system of 10 regular 1200 baud modems and two fax modems. It produces about 20 transactions a minute of about 40 bytes each.

In total, therefore, we are producing data at the maximum rate of about 100 bytes per second. This is very slow by network standards, just under the throughput of a 1200 baud modem. This is especially true, considering that the busiest time only lasts about two hours a day.

The file system consists of IMAGE datasets, several KSAM files, and a number of MPE flat files. The operator screen program adds records to several IMAGE datasets and to three of the KSAM files. The batch program makes updates to one of the KSAM files. This is the major part of the data that must be shadowed from one system to the other.

The data is very transitory, as it expires within a two-week period. It is allowed to accumulate for a month, at which point monthly statistics are computed, the old data purged and the datasets reorganized.

If the main computer were to go down it is desirable to keep as many operator terminals going as possible. However, the outgoing communication network only requires a minimum configuration as all messages are assigned a priority and those of the highest priority would go out even with a small number of modems. Some considerable compromises can be made in the performance of the backup system in the interest of minimizing the cost.

Networking Software

The biggest single decision was the method of networking the computers.

One possible solution was to write our own software to communicate from machine to machine using one of the serial ports on each machine. This would be a solution that would be non-standard, closely tied to the application software and that would probably take a long time for all the bugs to be ironed out. It would also take a long time to write and to get into production.

It was much preferable to find an off-the-shelf solution, if, in fact, one existed. There is a product from HP called "Silhouette" that will shadow databases from one HP3000 to another. Unfortunately, it works only with IMAGE databases. I understand that it uses IMAGE logging to do its thing and this is a fairly high overhead operation. With our heavy reliance upon KSAM files, this product was unlikely to do the trick for us.
Fortunately, HP recommended us to Quest Software and their product "Netbase" and this is the product that we finally used. This product will also shadow files from one computer to another, but it is not restricted to IMAGE files. Netbase intercepts the standard intrinsic calls to the file system, plus a selection of other intrinsics and thereby gets access to the data that is to be passed over the network.

Because the intercept routines are usually kept in an account SL, the application programs must be run with "lib=p" or "lib=g". The interception is done with very little overhead and there is a method of running standard programs like "FCOPY" or "QUERY" in a shadowing mode without having to place routines in the system SL.

In addition a job is kept running all the time that contains

the processes that do the actual transmission and reception, using standard HP network calls. The processes in this job are also responsible for posting data to the shadow files.

Hardware

The hardware selected for our main machine, the "A" computer, was a Micro XE, with a large Eagle drive and 72 ATP ports. Since we were looking for a minimum cost setup, the backup hardware chosen was a Micro GX. This has only 16 ports which is not enough to run all of the terminals and modems, but we decided that, since downtime on an HP3000 is rare, an attenuated performance under emergency conditions was acceptable.

A standard network would have cost us as much as the backup computer by itself, so we were looking for a low-cost alternative. It seemed reasonable that there should be a low cost alternative when we considered the extremely slow transfer rates that we were looking for. HP has such an alternative, but it is not too widely known. The alternative is known as ASNL (ASynchronous Network Link). Using this method, two serial ports on an HP3000 can be linked at a speed of 9600 bits per second and made to look like a standard network. One of the quirks of this system is that it is only officially supported using modems, not hardwired. However, it works beautifully with a $65 cable connecting the two machines, and costs under $3,000 total for both machines. It is more than adequate for our purposes, and the price is right.

Of course, in the case of an emergency, we would have to be able to switch the terminals and modems from one machine to the other. This was accomplished very easily with a set of A-B switches in a rack-mount panel, and this switching scheme is shown diagrammatically in Figure I.

Because the "B" system is seriously limited in the number of ports available, the peripherals are cut to a minimum in the emergency mode. Note the "X" switch that is arranged so that, under normal circumstances, an outgoing modem is attached to the "A" machine and a standard incoming modem is attached to the "B"system. In emergency operation, this incoming modem is lost, to make room for an additional outgoing modem.

It should also be noted that the system has only one tape drive, on a HPIB switch to connect to either machine, and one serial printer, that is connected to the "A" machine. This printer is not capable of being switched to the "B" machine.

Setting It Up

The hardware and software was very easy to set up. Plugging in modems, terminals, and switches is tedious and repetitive labor, but easily accomplished. The terminal and modem switches are rack-mounted and fit neatly in the cabinet containing our tape-drive and Eagle disc drive.

HP took care of setting up the ASNL link (They won't sell it without consulting help), and installing the Netbase system was very simple. However, it took quite a time to understand some of the intricacies of getting files back and forth from computer to computer, and optimizing the transmission. Quest Software were very supportive during this phase, dialing in to the computer when necessary to solve problems for us.

It should be noted that we did not purchase NS from HP, so facilities such as remote access are unavailable.

The software comes in three parts, Shadowing, Network File Access and Remote Spooling. The shadowing portion was clearly needed, but a careful study was made to see whether we needed the other capabilities, since we wished to keep our expense to a minimum.

There is one operation that is not amenable to networking, namely the monthly purges and reorganizations. Not only is the asynchronous connection too slow, but we discovered that programs such as Dbgeneral, which is used for IMAGE reorganization, and Copyrite, which is used for KSAM reorganization, are not amenable to shadowing because they use special privileged mode intrinsics that Netbase will not intercept.

There is an alternative way to handle this using Netbase. The remote spooling handles input files as well as output files, so it is possible to submit a job on one machine and then move it to a second machine for execution. Using this feature, I set up jobs streams that submitted twin jobs, one for the "A" machine and another to be copied to the "B" machine. In this way, the purges and reorganizations were done in parallel without any shadowing. Note that the priority for the job destined for the B machine and the job fence on the B machine must be lower than those for the A machine, so that the job destined for the B machine does not run prematurely on the A machine.

Our testing indicated that we could do all we needed using the Shadowing option together with the Spooling option, so the system has no Remote File Access.

We needed to make one change to the application programs for a satisfactory system. The operator program usually writes a significant number of records to several different files

for a complete transaction. In the case of an emergency switch to the backup machine, we needed a method of recognizing which transactions had been completed and which ones had not. This turned out to be very readily accomplished with a flag set in the first record written, which is turned off as the last action of the transaction. However, this flag is not a key value, and we have to run a serial read on this file before turning up the backup system. It only takes about two minutes, but if not for this, switchover could be accomplished in a few seconds.

There is also the question of getting back to the "A" machine after it has returned on-line. It would have been possible to shadow back from the "B" machine to the "A" machine. However, this seemed to be pointless and foolish. If the "A" machine fails, then it is likely that the data on it is in an undesirable and unknown state. Under these circumstances, it is better simply to store the data from the "B" machine and restore it on the "A" machine. Jobs were set up to carry out this process, as well as the reverse one of synchronizing the data on the "B" system with that on "A". This implies that, when an emergency occurs, operation will continue on the backup machine until the close of business gives an opportunity to switch back.

Experience

The actual operation of the system satisfied our highest expectations. The load placed on the "A" system was minimal and unnoticeable. The "B" machine is usually updated so quickly that is almost impossible to measure any delay. The only possible difficulty is the complexity of keeping track of identical jobs initiated on one machine, but running on two different ones.

We ran into one or two minor bugs in the software, principally associated with KSAM files. Quest Software were able to resolve the problem in an amazingly short time, and provided an updated version of the software in less than 24 hours.

How does it work in an emergency? The equipment has been installed now for about 18 months, and, as expected, we haven't had any hardware problems. We have had to switch machines on a couple of occasions as a result of software or operational problems. The "B" machine has always taken over without a problem.

Other Benefits

We gained other benefits from this system that we had not anticipated.

First of all, the second machine is used for software development, thereby taking a load off the production machine. We have also set up the shadowing so that program files are shadowed in the opposite direction to data, namely from the "B" system to the "A" system. In this way, when a program is updated on the "B" machine, the update is automatically reflected on to the "A" machine without actually copying it. Since we did not buy the Network File Access portion of Netbase, copying the file would necessitate either storing to tape and restoring on the "A" machine or using a PC to download and upload the program.

However, this trick has a problem. When a program file is replaced it is usual to purge the old program file before copying the new one into place. The MPE "purge" command is not intercepted by the Netbase software, and so the program file on the "A" machine does not get deleted. "Rename" has the same problem.

A special purge was therefore written that imitates the standard MPE command, opening a file and closing it with a disposition of "delete". This command can be intercepted by Netbase and shadowed across the network. By setting up UDCs for "purge" and "rename" using this program, the programs can be properly shadowed from one machine to the other.

A second benefit is that, since the data is identical on both machines and all the software development is on the "B" machine, we only need to back up the "B" machine. The tape drive is therefore usually connected to the "B" machine and used for backups, which can be carried out during normal operating hours.

The spooling feature can be used generally to submit jobs on the "A" machine and have them run on the "B" machine. Furthermore the database on the "B" machine can be set up to allow reading, and so reports can be run on that machine without adding any load to the "A" machine. At present, this is only used for producing one report, a large report that is spooled to tape and put on microfiche.

We only have one printer and this is on the "A" machine. Consequently, whenever we want to print something from the "B" machine it must be first moved over to the "A" machine. This can be accomplished very easily. The "B" system has a ficticious printer configured with two device classes that are both spooled. There is a second continuously-running batch job that picks up output sent to one of our fictitious device classes and ships it over the link.

Conclusions

Our experience with a "hot backup" of this kind indicates that it is very easy to set up with the tools available today. The operation is very reliable and can give benefits over and above those to be expected from a simple shadowing operation.

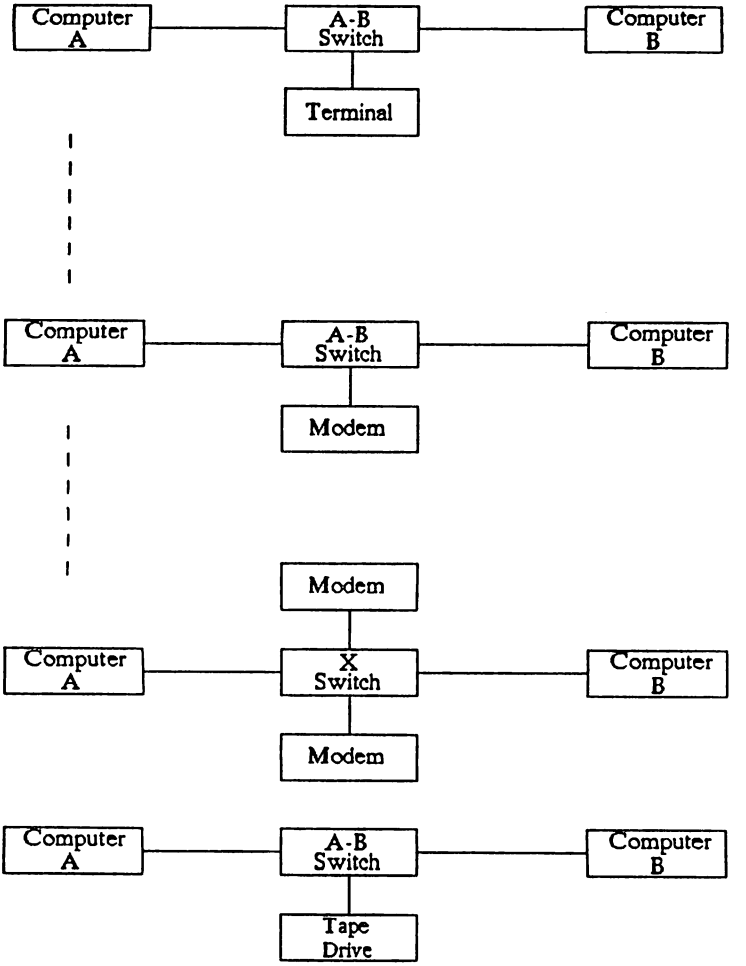

Computer A
A-B Switch
Computer B
Terminal
Computer A
A-B Switch
Computer B
Modem
Modem
Computer A
X Switch
Computer B
Modem
Computer A
A-B Switch
Computer B
Tape Drive

Fig. 1

MPE from a VMS Perspective

Paper #3110

By Robert S. Dobis and Steven M. Elsten

Crowe, Chizek and Company
2100 Market Tower
10 West Market Street
Indianapolis, IN 46204-2976
(317) 632-8989

The objective of this presentation is to highlight some of the similarities and differences, from the application developers perspective, between MPE on the HP3000 and VMS on the Digital VAX. Our intent is to provide a general, introductory cross reference between both environments. With this reference, users familiar with one of the two operating systems will have a starting point when beginning to develop programs in the other operating system. The general areas to be covered in this presentation are:

File Names

System Commands

Command Procedures

File Structures

Databases

Query Languages

Programming Languages

Forms Handlers

Application Development Tools

VMS vs MPE FILE NAMES

VMS	MPE
DEVICE NAME	NOT USED
DIRECTORY PATH	GROUP/ACCOUNT
UNLIMITED LEVELS	TWO LEVELS
39 CHARACTERS IN LENGTH	8 CHARACTERS IN LENGTH

EXAMPLES:

(VMS)

```
DUA1:[ROOT.LEVEL1.LEVEL2]FILE_NAME.DAT
```

(MPE)

```
FILENAME.GRPNAME.ACCTNAME
```

VMS COMMANDS AND THEIR MPE EQUIVALENTS

VMS	MPE
BACKUP	STORE/RESTORE
COPY	COPY
CREATE	BUILD
DEASSIGN	RESET
DEFINE/ASSIGN	FILE
DELETE	PURGE
DELETE/ENTRY	ABORTJOB #J/DELETESPOOLFILE
DIFFERENCE	Third party software
DIRECTORY	LISTF
EDIT	Editor
INQUIRE	INPUT
LINK	LINKEDIT
LOGOUT	BYE
MAIL	Third party software
PRINT	FCOPY/Editor
RENAME	RENAME
RUN	RUN
SEARCH	Editor
SET DEFAULT	CHGROUP
SHOW DEVICE	SHOWDEV
SHOW LOGICAL	LISTEQ
SHOW PROCESS	SHOWME
SHOW QUEUE (batch)	SHOWJOB
SHOW QUEUE (print)	SHOWOUT/LISTSPF

VMS COMMANDS AND THEIR MPE EQUIVALENTS (continued)

VMS	MPE
SHOW SYMBOL	SHOWVAR
SHOW USERS	SHOWJOB
STOP/ID	ABORTJOB #S
SUBMIT	STREAM
Symbol Assign	SETVAR/SETJCW
TYPE	PRINT/FCOPY

VMS vs MPE COMMAND PROCEDURES

VMS	MPE
SYMBOLS	VARIABLES
LOGICALS	FILE EQUATIONS
IF...THEN...ELSE	IF...THEN...ELSE
GOTO	GOTO/WHILE
WRITE SYS$OUTPUT	ECHO
INQUIRE	INPUT
@<*FILE NAME*>	<*FILE NAME*>
PARAMETER PASSING	PARAMETER PASSING

EXAMPLES:

(VMS)

```
$ INQUIRE/PROMPT="ENTER PROGRAM NUMBER " PROG_NAME
$ IF PROG_NAME .EQS. "PROG1"
$ THEN
$     DEFINE DATA_FILE DUA0:[DATA]DATA_FILE.DAT
$     RUN TEST_PROGRAM
$ ELSE
$     DEFINE DATA_FILE DUA1:[DATA]DATA_FILE.DAT
$     RUN TEST_PROGRAM
$ ENDIF
$ DEASSIGN DATA_FILE

$ @SAMPLE_PROC
      PROG1
```

(MPE)

```
INPUT PROGNAME;PROMPT="ENTER PROGRAM NUMBER "
IF !PROGNAME = "PROG1" THEN
      FILE DATAFILE=DATAFILE.GROUP1
      RUN TESTPROG.RUNGROUP
ELSE
      FILE DATAFILE=DATAFILE.GROUP2
      RUN TESTPROG.RUNGROUP
ENDIF
RESET DATAFILE

: TESTPROC
      PROG2
```

VMS vs MPE FILE STRUCTURES

VMS	MPE
RMS (Sequential)	MPE (Flat file)
RMS (Indexed)	KSAM
File Definition Language/ EDIT/FDL	KSAMUTIL
CREATE/FDL	BUILD (KSAMUTIL)
MULTIPLE KEYS	MULTIPLE KEYS
1 FILE	2 FILES (KEY FILE/DATA FILE)

VMS vs MPE DATABASES

VMS	MPE
RDB (RELATIONAL)	IMAGE (HIERARCHICAL) ALLBASE (RELATIONAL)
ORACLE	ORACLE
INGRES	INGRES
SMARTSTAR	
SYBASE	

VMS vs MPE QUERY LANGUAGES

DATATRIEVE	QUERY
RMS AND DATABASE FILES	DATABASES ONLY
READY	DEFINE
FIND	FIND
PRINT/LIST	REPORT
MODIFY	UPDATE
ERASE	DELETE
PROCEDURES	PROCEDURES
RECORD DEFINITIONS	UNSUPPORTED
DOMAIN DEFINITIONS	UNSUPPORTED
@<*PROCEDURE NAME*>	XEQ <*PROCEDURE NAME*>

EXAMPLES:

(DATATRIEVE)

```
READY DOMAIN1
FIND DOMAIN1 WITH KEY_FIELD = "ABCD"
PRINT ALL DATA1, DATA2, DATA3
MODIFY ALL DATA4, DATA5
   ABC
   123
FINISH
```

(QUERY)

```
DEFINE
  DBNAME1
  PASSWORD
  1
  DSETNAME1

FIND KEY_FIELD = "ABCD"
REPORT
  D1,DATA1,20
  D2,DATA2,40
  D3,DATA3,60
  END
UPDATE REPLACE,DATA4="XYZ";DATA5="123";END
EXIT
```

VMS vs MPE PROGRAMMING LANGUAGES

VMS	MPE
BASIC	BASIC
C	C
COBOL	COBOL
DIBOL	UNSUPPORTED
FORTRAN	FORTRAN
PASCAL	PASCAL
PL/1	PL/1
UNSUPPORTED	SPL

VMS vs MPE FORMS HANDLERS

VMS		MPE
FMS	**TDMS**	**VIEW**
Form Files/Libraries	Forms Stored in CDD	Form Libraries
FMS/EDIT	FDU	FORMSPEC
Screen Layout Editor	Screen Layout Editor	Screen Layout Editor
FMS/LIB	NOT SUPPORTED	FORMSPEC
Datatype Edits in form	Datatype Edits in form	Datatype Edits in form
No Advanced Validation	Range/List Validation	Processing Specs
External Procedure Calls	External Calls to Requests	External Intrinsic Calls
Char or Block Mode	Char or Block Mode	Block Mode

VMS vs MPE APPLICATION DEVELOPMENT TOOLS

VMS	MPE
COPY/INCLUDE FILES	COPYLIBS - COBEDIT
TEXT EDITORS (EDT, TPU,...)	EDIT3000, QEDIT, TDP,...
BATCH LISTING/PRINT FILES	NATIVE MODE SPOOLER
COMMAND PROCEDURES	COMMAND PROCEDURES
SYMBOLS	COMMAND PROCEDURE NAME HPPATH
LOGICAL	FILE EQUATIONS
DATATRIEVE	QUERY
CDD	SYSTEM DICTIONARY

RAIL SYSTEMS FOR TOMORROW

DATA COMMUNICATION TRAINS.....

By: Aldo Falossi
Cable Management Systems
3200 West Warner Ave.
Santa Ana, Ca. 92704
(714) 662-0664

Paper Number 3111

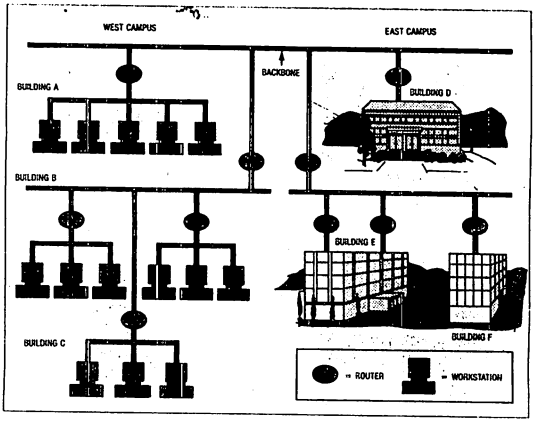
RAIL SYSTEMS FOR TOMORROW

DATA COMMUNICATION TRAINS.....

By Aldo Falossi

Cable Management Systems, Inc.

Any way we look at it, the standard work being conducted under the joint auspices of the Electronics Industry Association (EIA), the Institute of Electrical & Electronic Engineering (IEEE), and the Telecommunications Industry Association (TIA), leads towards a structured wiring based on fiber and copper media. The F.D.D.I. (Fiber Distributed Data Interface) standard has been given a prominent position as a backbone network scheme in the wiring now being drafted and scheduled for completion in 1992.



The movement towards "STANDARDS" wiring schemes began building momentum after the break-up of AT&T in the early 1980's. Before deregulation, building wiring being done for Voice and Data were in the majority of cases done by the telephone companies using voice (analog) components and installation technique based on telephone expertise gained by AT&T over the years.

Even before the divestiture of AT&T the market was being flooded with new communications and computing systems. Each vendor had its own solution and network scheme (Ethernet, Novel, Arcnet, RS232 and now even Token-Ring). Each supplier specified its own unique distances, medium, and connectors, leaving the user with little choice other than re-cable their facilities every time they added a new system or moved to a different transport protocol.

Visualize a Fortune 1000 company which had a Wang V/S system for word processing, an IBM system 38 for administration and an HP 3000 for manufacturing. Although the backbone could be wired for Ethernet (thick or thin coax) the premise wiring bringing

communication to the various offices would require dual coax for Wang, Twin-ax for the IBM and multiple twisted pair for the HP. The user had few choices.....either dedicate a user location to a specific system or wire three different sets of cable to each user location.

The problem faced by the user became more complex as personal computers invaded his company.....how to network these intelligent stations to his main frames and what transport medium must he use.

Over the years several vendors have offered their own cabling systems, but users, (more often than not), found themselves locked into a proprietary medium and systems scheme which provided transmission and flexibility limitation.

Standards like RS232 will become outdated as users will move to RS422/23 to increase the effective transmission speed between computer and terminals as computing power provided by faster main frame and personal computers require faster data transfer. Standards such as

10BaseT have been quickly developed and approved as interim stop-gap to take advantage of the billions of feet of wire which AT&T has placed in industrial, commercial and residential facilities. Companies have started , become successful and some have disappeared all in the effort to take advantage of the network market opportunity which promises faster speeds, longer distances, flexibility, easy management and maintenance.

The U.S. rail industry promised the same when millions of feet of railroad tracks were put down across the country.

I can hear the rail executive's conversations.....we will offer the public movement from the east to the west, north and south. We'll be able to move freight, troops for our defense, transport from the smallest to the largest animals, have wagon, for fuel, milk, restaurant to feed passengers and beds for those on long trips. They forgot one major parameter....."SPEED"! The U.S. rail system cannot support trains at over 100 miles per

hour while we can now manufacture locomotives which can achieve speeds of over 250 miles per hour.

The telephone industry did the same when the promised panecea facilities with 25 pair cables of 24 gauge (24 AWG) unshielded.

I can hear the phone man talking.....we don't need to worry about bandwidth since voice it's typically between 300 and 7500 herts. Let's not worry about loss on the line, we will put amplifiers when needed and if the signal is too low to hear, people will scream louder on the phone. Crosstalk will make us more money since people will stay longer on the phone to hear the conversation on the other line.

To prevent problems such as the ones faced by the railroad industry, The Electronic Industry Association (EIA) stepped in and tried to create standards in media that would accommodate several different vendors.

The task proved to be a major undertaking even with the help of the Telephone Industry Association (TIA), the

Building Industry Service International (BICSI), the Building Owner and Manager Association (BOEMA), and the Construction Specifications Institute (CSI).

The EIA continued its work with computer manufacturers, communication vendors as well as the major users such as HMO's, Insurance firms and Banks.....the outcome may be worth waiting.....meanwhile, what is a user supposed to plan for his network needs of today with an eye towards the future.....

"RAIL SYSTEMS FOR TOMORROW

DATA COMMUNICATION TRAINS....."

This writer feels that we must look at the problem faced by our industry (network) from a different prospective. Somewhat the same way "MA" BELL wired the building years ago, or the utility companies distributes power throughout our facilities today.

Before the building walls are closed, outlets for phone and power are wired at strategic locations, after the walls are closed the terminating connectors are

installed and tested regardless if a toaster made by G.E. or Emerson is plugged in or a phone made by AT&T or NEC will be used into the respective power and phone outlets.

The computer/communication industry can learn a great deal from these approaches, especially when industry forecasts are projecting that by the year 2000 there will be a Terminal (or PC's) in every modern office location where there is a phone being used.

The solution to our dilemma is somewhat more complicated than the one faced by the power or telephone industry.

How can a building or facility be wired intelligently for computer data communication when vendors of equipments have designed their products to run on proprietary hardware, connectors, media, software, etc., etc.

To better understand our dilemma, we need to take a look at our task.

There are four basic elements that make up Local Area Networks (LAN's).

1) The Topology
2) The Proto-Call
3) The Access Method
4) The Medium

While Topology, Proto-Call and Access Method change from vendor to vendor it appears that the Medium - The Cabling System - is finally headed towards standards which are transparent to the hardware/software manufacturers.

In the past, the questions posed by the Data Communications Manager was based on....."When is it better to use what medium"?....."What are the advantages and disadvantages of each type od cable"?

TWISTED PAIR, UNSHIELDED

The Good:

* It's the most flexible and the easiest to move and install in most situations.
* It's easier to install than the other types of copper cables and it's familiar to most people so they feel comfortable with it.
* It's the least expensive type of copper cable.

The Bad:

* It carries the least amount of information at limited data rate.
* It's very vulnerable to lightning, the elements and interference.
* Emits an electromagnetic field which can cause interference with other copper cables like crosstalk or data errors.

TWISTED PAIR, SHIELDED

The Good:

* Less likely to cause interference than unshielded cable. Less likely to be affected by nearby

equipment

or wires.

* Less vulnerable to the elements than unshielded.
* It's quick and easy to install. Everybody has a lot

of experience with it.

* It has higher data rate than unshielded.
* It has low bandwidth compared to coax and fiber optics.

COAX

The Good:

* Carries more information than twisted pair, shielded or unshielded.
* There is a healthy supply and it's easy to install.

The Bad:

* It's bulkier than twisted pair and difficult to move.

Coax cable is quite rigid and requires special tools

to get it where you want it.

* It's hard to move coax and Networks using this Medium are costly.

FIBER OPTICS

The Good:

* Carries information at undreamed-of speeds.
* Neither creates nor is susceptible to most interference because it doesn't have an electromagnetic field.
* The cost of fiber has dropped by 75% in the last few years.

The Bad:

* It's still expensive to convert optical to electrical energy.

* Most people don't know how to install it.
* A small imperfection in the cable can mess up all the signals.

Present day medium (cable) standards are predicated on the best known and most often referred Local Area Network (LAN) specifications.

For example: The Ethernet electrical specifications are intended for LAN systems with data rates that do not exceed 10 million bits per second (Mbps).

Ethernet (as a standard) has been around for over 10 years and as computer systems have progressed, the LAN's capacity needs have increased. Other factors driving up the capacity of existing networks include CAD/CAM work station performance, the movement towards distributed applications and the increased number of multiple sub/networks increasing the load on the backbone.

"If history has taught us anything is that computer systems will increase in through-put power.....and humans will be hard pressed to run 1 mile below 3 minutes....."

What that means is that anytime we have processors

(intelligent devices) communicating with each other without the interaction of a human, we can expect increased through-put which can only be limited by the rail road tracks (medium) handling the movement of date from one point to the other. While if there's human interaction, the best we can expect is how much faster we can read, type, and/or handle human transitions.

It's quite improbable for a human to increase its through-put by an order of magnitude while from Ethernet to F.D.D.I. (Fiber Distributed Data Interface) an order of magnitude increase in through-put is a reality today.

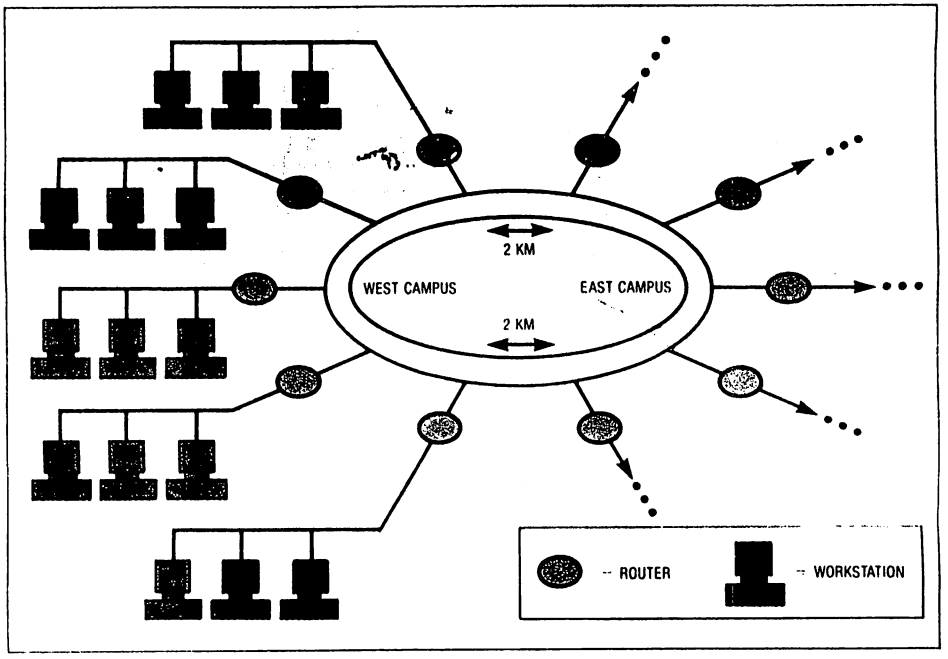
Understanding the limitations of humans and the part they play in the computer data communication scenario, "our goal is to design" a Network Cabling System to make it simple, yet manageable.

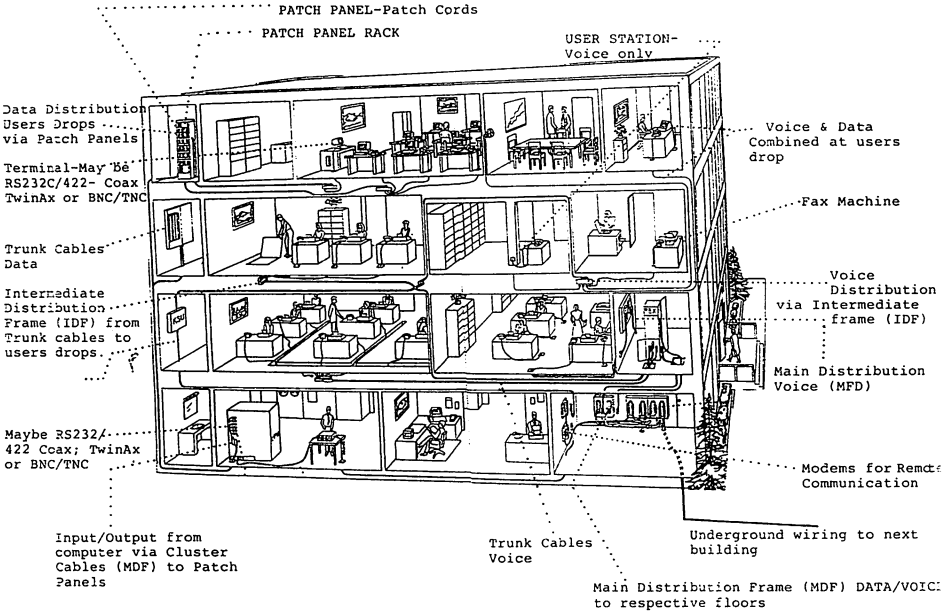
Backbone Cabling Solution

How are the communications and facilities managers going to install Fiber Optic Cabling to support todays need and future requirements ?.....The

answer is simpler than might be expected.

Network cabling design is straight forward with the understanding of the basic building blocks and the facilities layout.





First: The selection of 62.5/125 micron (m) fiber has emerged as the standard for both the computer OEM and building wiring environment, thus making the selection of fiber size easier. Shielded and unshielded twisted pair cable, while relatively inexpensive and already a large installed base, is limited in use for a backbone LAN by distance and bandwidth. In the last two years, optical fiber has begun the migration from the outside plant environment, up the building riser, up to the distribution closets.

Second: How is fiber backbone configured ? Fiber can be implemented like coax while it provides a high bandwidth, it can be used for transmission link among computers and peripheral equipment. A fiber network is typically configured as a dual, counter-rotating ring with branch and tree links joined to the ring at concentrators. The dual ring, (which requires running multiple fibers to each station in the ring), is self-healing when a fault occurs in one fiber link or at a cross bridge location. This protects the system from downtime when a failure occurs.

Fiber provides flexibility in designing networks for a number of different applications, including the following:

Data Center: The high speed backbone within the data center should be wired as in a campus or building environment. A physical star topology is recommended. Many devices can be served with one or more intermediate distribution frame (IDF), connecting CPU's, storage controllers, other peripherals and communication servers.

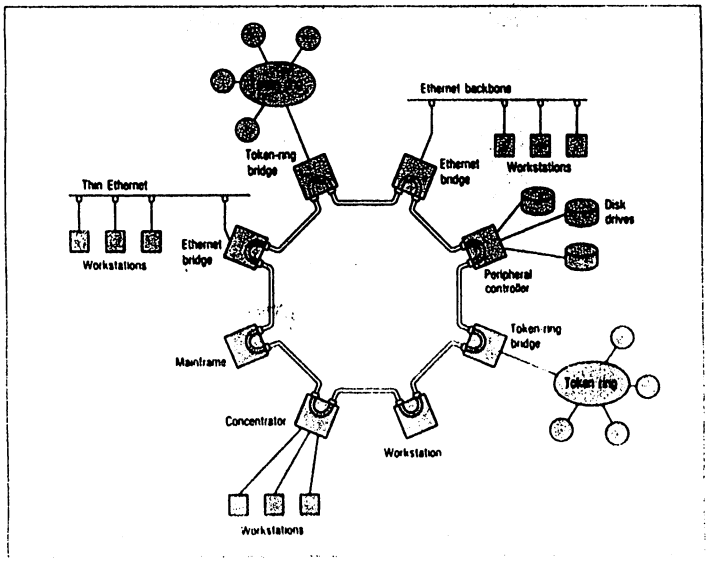
Since the data center is typically small, (less than 100 Ft.), using a patch panel system approach to connect the devices offers flexibility. Also because the cables are short, each device can be easily re-routed or removed if damaged.

The patch panel system becomes the central point for connecting all the devices using jumper cables provides additional flexibility is gained since the equipment may be easily connected in a point-to-point, star or ring configuration. The user can quickly change topologies by rearranging the cross connect jumper cables at the patch panels.

Fiber cables in the data center is normally a duplex cable of rugged design, to withstand installation force and the environment under a data center raised floor.

Inside Building: In a building environment, the riser fiber cable is the backbone connecting the equipment room (data center) and the individual floor (or distribution closets). Cables in a rotating dual ring

run back to the data center, and on each floor, using distribution panels are connected to the Local Area Networks, (LAN's), PC's, and via terminals via routers, bridges, or directly to peripheral servers.



Depending upon the individual application a cable fiber count may require from 6 to 36 fibers to satisfy current and future network needs.

Intrabuilding cable should meet the 1990 National Electric Code (NEC) requirements designated by Underwriter laboratories, Inc. The code specifies the

level of safety that each cable must meet; plenum, riser or general purpose.

The general contractor installing the cable should also be familiar with local building codes to select the appropriate cable and methods of installation.

Users Stations: Once the fiber cable backbone, main distribution frame (MDF) and the intermediate distribution frame (IDF) has been designed and installed for the unattended equipments from the data center to the distribution closets (LAN's) the final portion of the wiring system is implemented to bring data outlets to each (present and future) location which will have a terminal or peripheral.

Since most office locations are occupied by a human, the recommended media for this task is shielded twisted pair copper cable capable of supporting data transmissions of 10 - 20 Mbps.

Each user location should be wired with a cable consisting of four (4) pair terminated to a multiple

outlet which can be custom configured for the specific devices which will eventually connect to it.

In the distribution closet (twisted-pair) Servers, MAU's. Patch Panels, etc., provide for topology conversion in addition to station expansion and rearrangement.

Growth of the network can be modular and non-disruptive, addition and deletions can be accomplished by merely plugging and unplugging jumper cables.

Conclusion: Whether installing coax or twisted-pair backbone in a building (or campus) environment today, the user must plan for the eventual migration of fiber into his network. Future cost and time in adding, maintaining and managing the flow of data from the data center to the user can be drastically reduced by using standard components and good (planned) design practices.

PAPER #3113
DISAPPEARING DIAL-UP

JAMES D. HAM

SOUTHEASTERN PUBLIC SERVICE AUTHORITY
OF VIRGINIA

723 WOODLAKE DRIVE
P. O. BOX 1346
CHESAPEAKE, VIRGINIA 23320
804-399-8924

DISAPPEARING DIAL-UP

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, isolater/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 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 administration 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 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 environmental 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 uploaded 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 new lines 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 effort 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 confirm 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 communications 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 liens 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 communication 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.

MPE/XL Internals & Performance

by

Michael C. Hornsby

Beechglen Development Inc.
2026 Beechglen Ct.
Cincinnati, Ohio 45233

(513) 922-0509

Overview

- The HP3000 Product Line
--> Price, Performance, Market Forces
- Applications and Data Bases
--> Spaces, Turbo Image/XL, Special Caps
- MPE/XL Tables
--> KSO's, Process Tables, Memory Mgmt
- MPE/XL File Management
--> Directory, XM, Mapped Files
- Common Performance Problems
--> Development, Dispatching, Configuration

The HP3000 Product Line

- Price versus Performance
--> Life Cycles, Pricing, and Futures
- Processor & BUS Differences
--> TLB, Cache, and Memory Sizes
- Program Modes of Operation
--> Compatibility, Translated, and Native
- Market and Technical Forces
--> Competition, UNIX, Applications

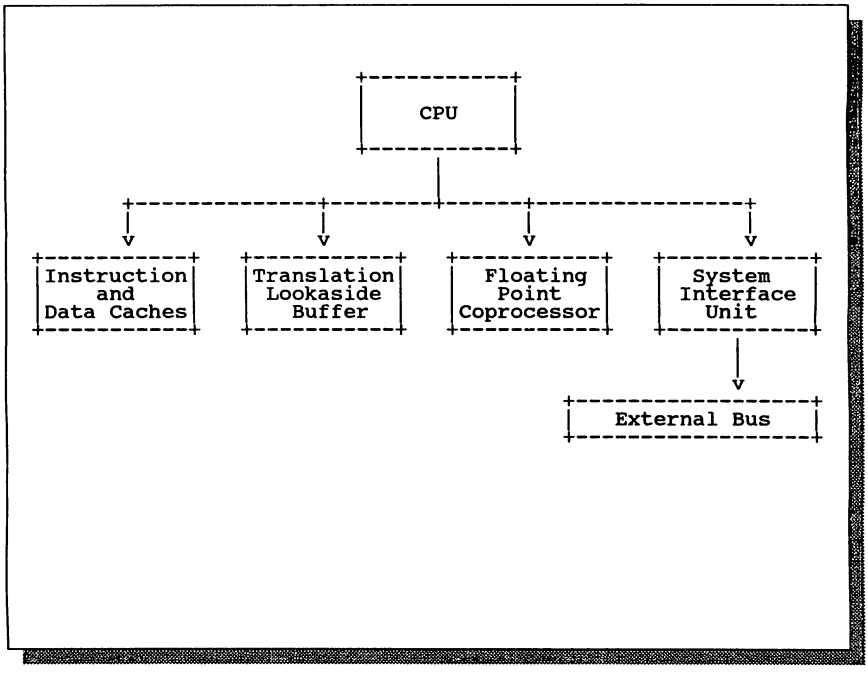
The HP3000 Product Line

Model	Max Users	Base Memory	CPU Cache	TLB	REL PERF	$'S
920*	20	24Mb	64Kb	64E	1.0	26K
922*	152	32Mb	32Kb	4K	1.7	75K
932*	250	32Mb	128Kb	4K	2.8	100K
948*	400	64Mb	1Mb	8K	6.0	190K
955	600	96Mb	256Kb	16K	5.4	385K
958*	600	96Mb	1Mb	8K	8.6	310K
960	600	128Mb	1Mb	16K	7.5	485K
980.100	600	192Mb	1Mb	128E	13.0	675K
980.200	600	256Mb	2Mb	256E	20.0	1050K

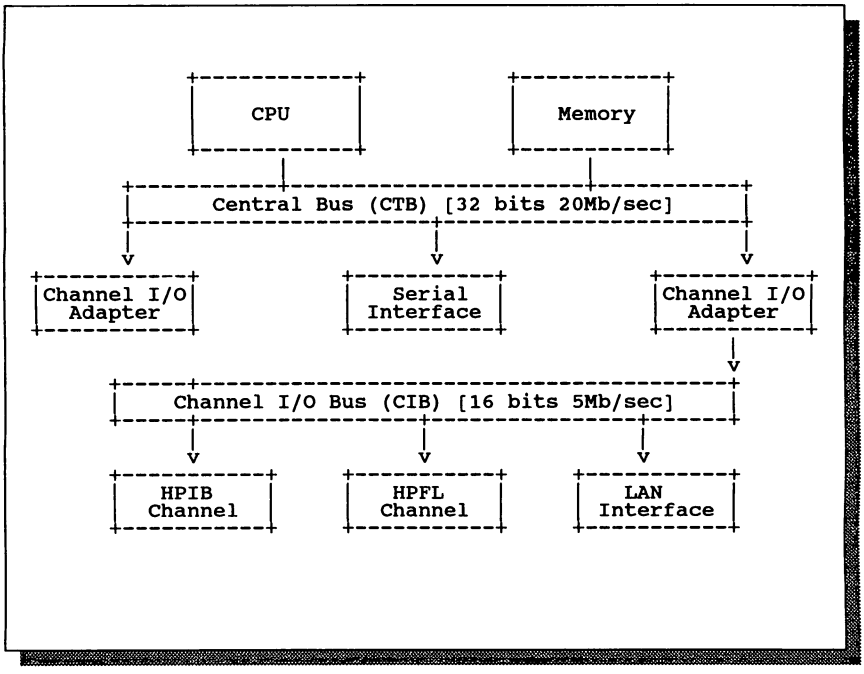
* - Package price includes disc & tape

MPE/XL Price versus Performance

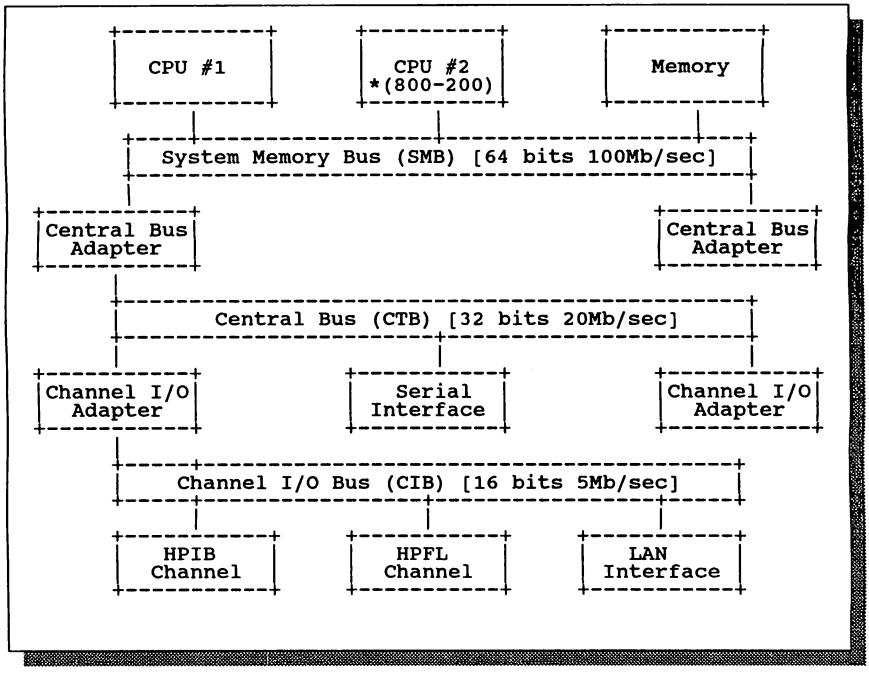
HPPA Processor Architecture



HP3000 HPPA Two Bus Architecture



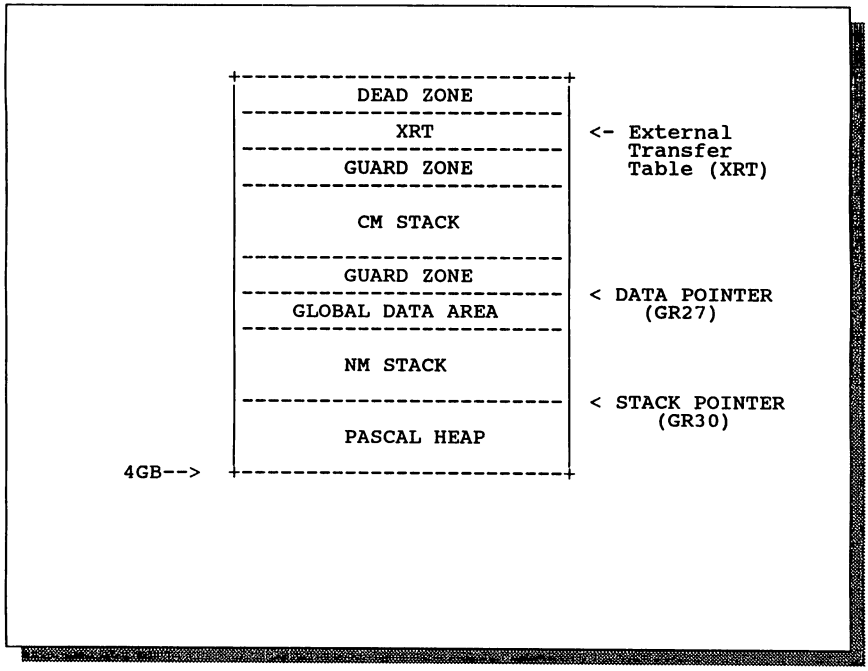
HP3000 HPPA Three Bus Architecture



Applications and Data Bases

- Process Spaces
--> Address Methods, Paging, Content
- Turbo Image/XL
--> Caste, Performance, Evolution
- Special Capabilities
--> Process Handling, Segments, Priv Code
- KSAM and Message Files
--> Niches, Problems, Opportunities

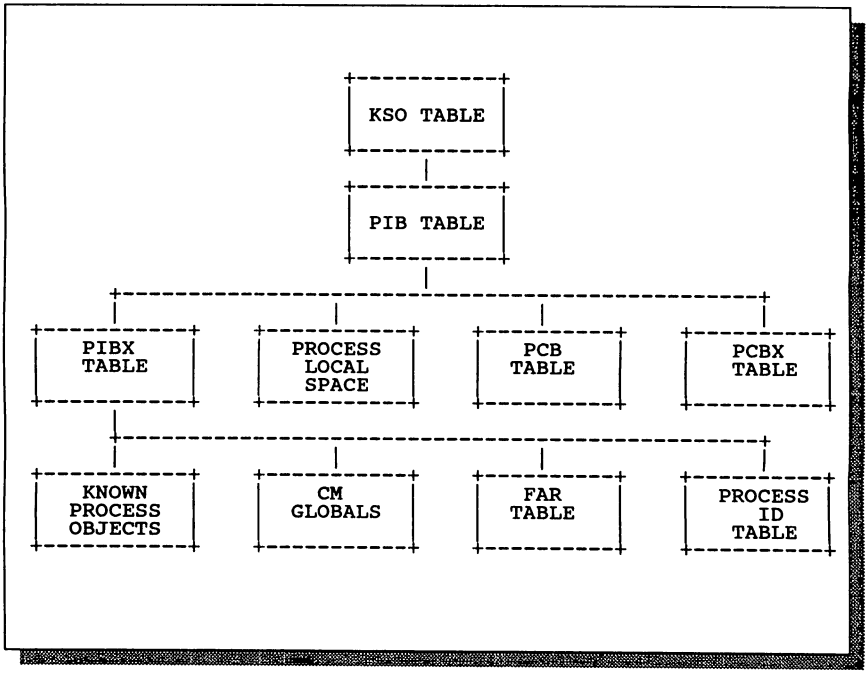
MPE/XL Process Local Space



MPE/XL Tables

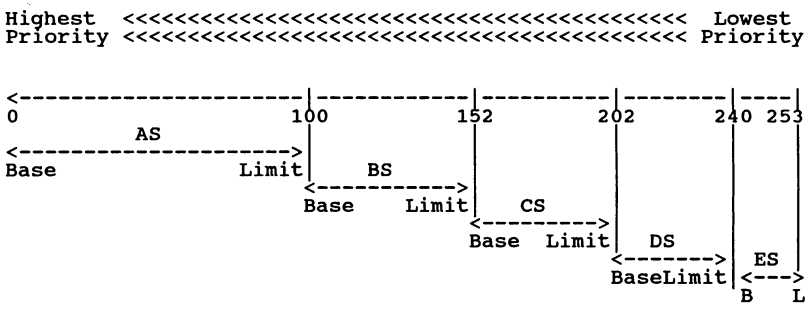
- Known System Objects
--> Structures, KSO's
- Process Control Structures
--> PIB, PCB, PCBX
- Dispatching Tables
--> Globals, TCB, Locking
- Memory Management
--> MIB, PDIR, PDIRX

MPE/XL PROCESS CONTROL TABLES

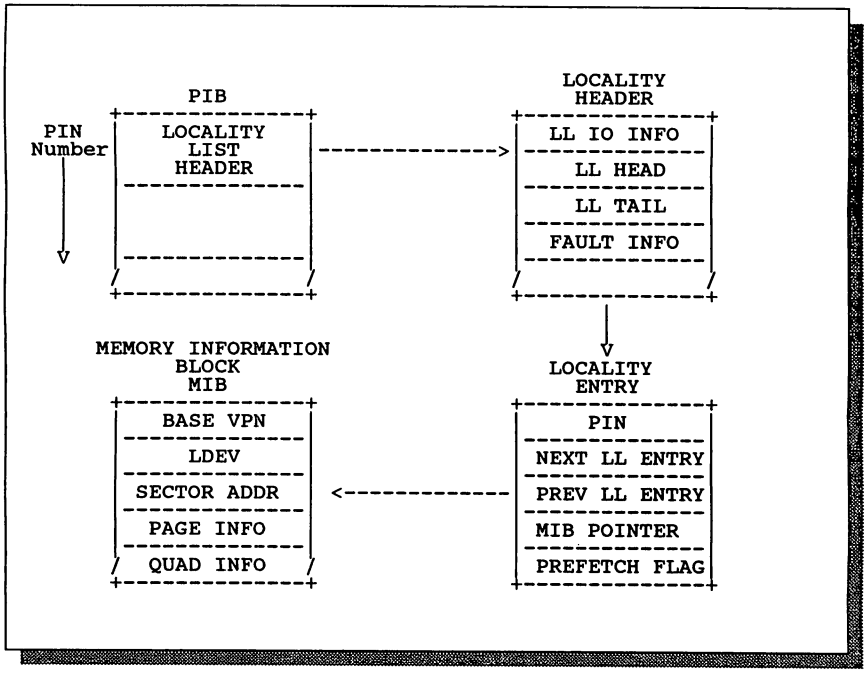


MPE/XL Dispatcher Queue Assignments

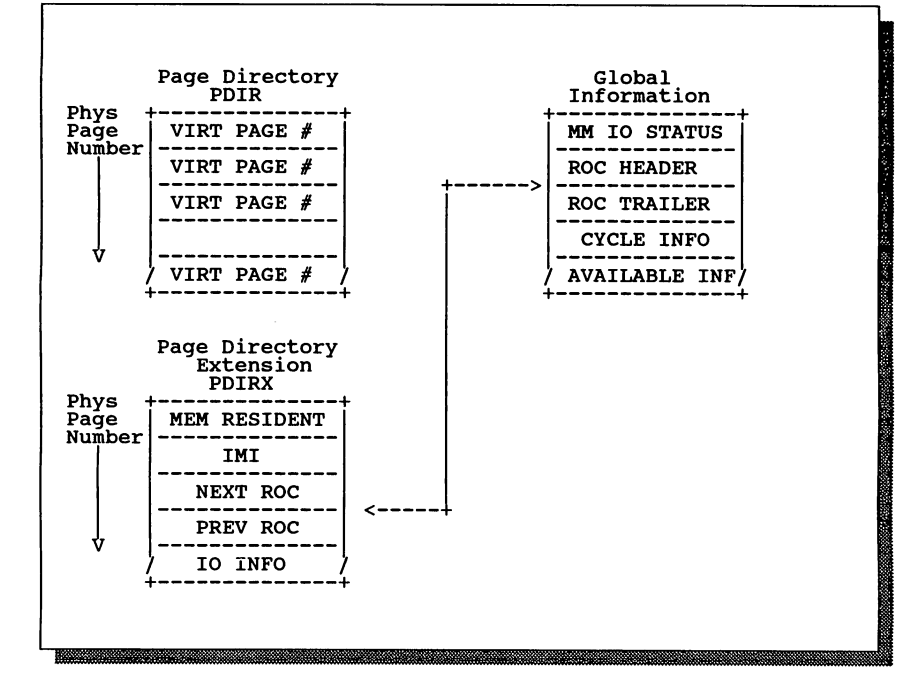
Subqueue	Base Pri	Limit Pri	Min Time	Max Time	Queue Type	Process Type
AS	0	99	-	-	Linear	MPE/XL
BS	100	149	-	-	Linear	MPE/XL
CS	152	200	200	2000	Circular	Sessions
DS	202	238	2000	2000	Circular	Jobs
ES	240	253	2000	2000	Circular	Jobs



MPE/XL MEMORY LOCALITY TABLES



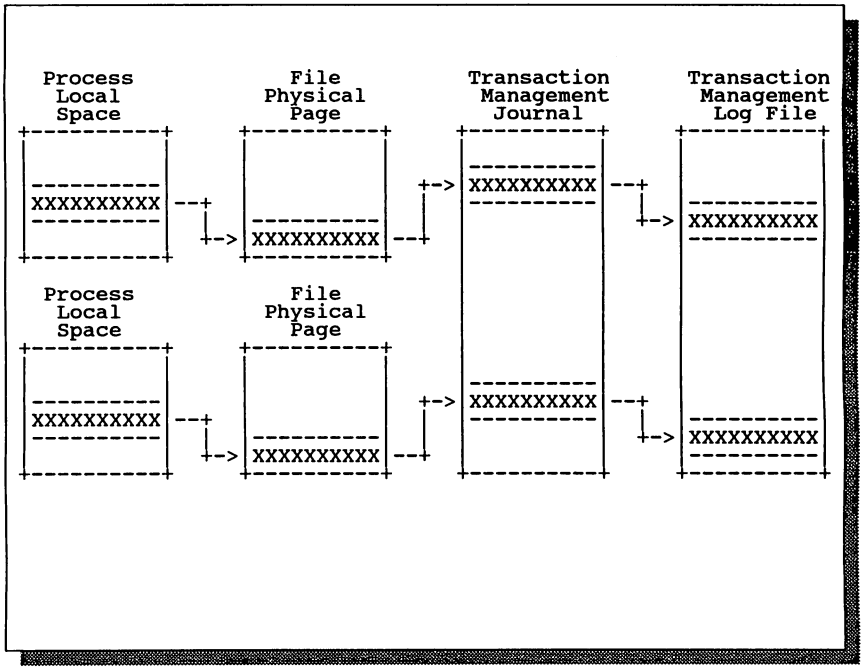
MPE/XL MEMORY MANAGEMENT TABLES



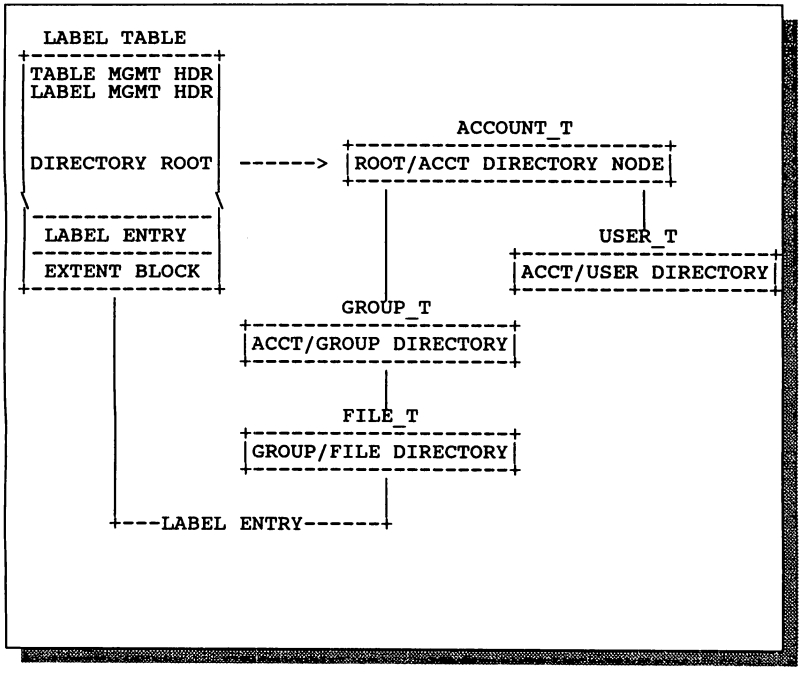
MPE/XL File Management

- Transaction Management
--> XMBuffers, XMlogging
- Directory Structures
--> Label Table, Extent Blocks
- Mapped Files
--> Compilers, File System Bypass, Pointers
- Volume Management
--> Recovery, Installation Management

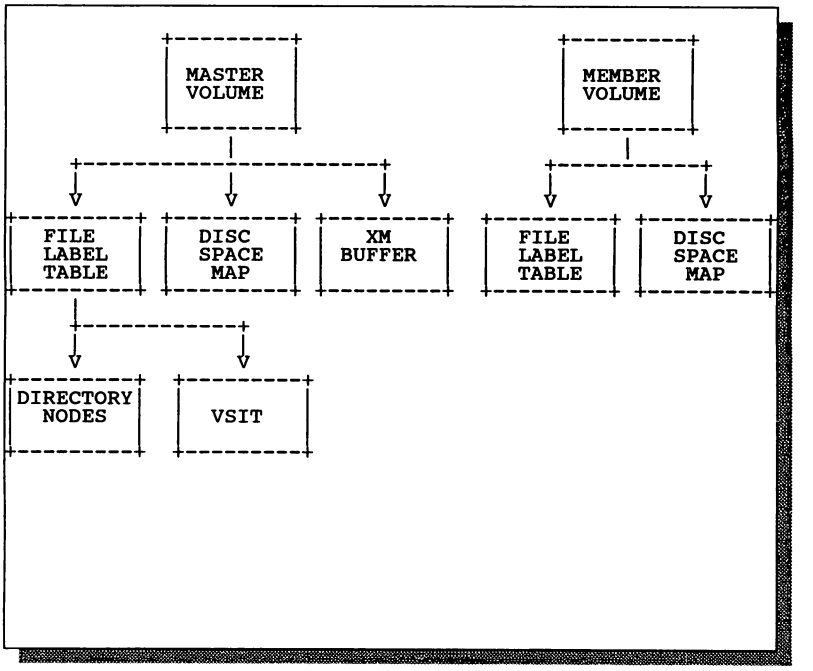
MPE/XL Transaction Management



MPE/XL FILE SYSTEM DIRECTORY



MPE/XL Volume Management



Common Performance Problems

- Development Activities
--> Compiles, Reports, Testing
- Short Transactions
--> Data Entry, File Transfers
- Process Handling
--> ASK, COGNOS, HPDESK
- Resource Sharing
--> Servers, Cooperative Processing

HP 3000 Capacity Planning in the Trenches

Robert A. Lund
Paper number 3115
Lund Performance Solutions
34130 Parkwoods Dr. NE
Albany, Oregon 97321
(503) 327-3800 FAX (503) 327-3276

Abstract

This paper will focus on capacity planning in the trenches. I will outline an HP 3000 capacity planning scheme that will insure pro-active performance management. I will stay as practical as possible and cover some of the best (and not so best) ways that others have dealt with the problem of planning for future application growth, and how best to meet that growth with an upgraded hardware configuration.

What is Capacity Planning?

The art of capacity planning involves a proper marriage of hardware to application load and function. It helps answer the question, "How much CPU horsepower, memory, disc drives, etc. do we need to provide adequate service for our data processing demands?"

Planning ahead for your data processing needs for two, three or even five years down the road is one aspect of an initial system sizing plan. But what about mid-stream when your accounting department "must" have a new payroll application added? Will the system handle it? Demands for system processing power are shifting sands indeed. This is one of the unknowns that DP Managers are faced with periodically.

Simply stated, capacity is usually thought of in terms of the horsepower of computer hardware, and the ability of that hardware to adequately fulfill the transaction requirements of the user community. Capacity planning, therefore, involves implementing a schedule to accommodate future growth. Accurately procuring hardware to drive the increase in load, within a timely fashion (read: no budget busting!), is what capacity planners are faced with.

Small companies often times have one person who wears the "hat" of many job functions. A system manager can be programmer, analyst, network specialist, capacity planner, and part time accounting supervisor, all in one! I will attempt to provide help for not only folks that are versed in performance management, but also for the poor multi-role souls.

Why Perform Capacity Planning?

Planning for future growth in system processing is just plain good sense. Just as in any aspect of business planning, you must also give some thought to computer hardware needs in the future.

By considering how you are going to match hardware upgrades with new loads, you will benefit in the following ways:

1) Your management will have the confidence that you are on top of future planning

2) You will not miss opportunities to extend the life of your system because you will be on top of monitoring and properly attending to housekeeping chores

3) You will minimize budget surprises for additional hardware

4) You will have the confidence that user service levels will be covered because you are continually monitoring them and taking action to insure their adequacy

5) You will see the early warning signs that point to an impending hardware upgrade

6) You will spend less time in evenings and weekends fretting about system throughput (overtime, job over-runs, user complaints, etc.)

7) Bottom line: You will not be flying blind into the future!

Let's get a bit more practical. Let's assume that you are the one responsible (perhaps by default) for the capacity planning effort. What are the demands placed on you? You are paid to deliver results and answer difficult questions. Some of the more down-to-earth questions that come up in DP meetings, or worse yet, in board of director meetings are as follows:

o "How do we accurately size an upgrade to our current system?"

o "When is improved performance not likely when migrating to a larger system?"

- "If we increase our accounts payable transactions from 50 per day to 100, what will be the net impact on the system?"

- "Do we have to go native mode if we migrate to MPE XL?"

- "When should a faster CPU be installed, and how long can we expect it to last?"

- "What will happen to response times when we add 25% more workstations next month?"

- "What is the most cost effective way to implement a permanent capacity planning strategy?"

Though we won't be able to answer all these questions in this paper, we will address some of the ways to deal with the subject of capacity planning.

Start Here: Profiling Your System's Performance

First things first, however. Consider this analogy. Before you can conclude whether your car is capable of pulling a new boat, you need to assess its current condition. If the engine is not tuned up, you will be making an assumption, perhaps falsely, that the car will not be able to handle the new load. You might then think that you need a larger one with more horsepower. A new (or larger) boat may mean a new car; this means money.

In the trenches, it is important to consider this analogy in light of your computer's current condition; this is the first logical step in implementing and maintaining a viable capacity planning effort.

It is critical that you stay on top of your system's "health". This is just plain wisdom, but more than that, it is a necessary prelude to implementing a real world capacity plan. A profile of your system answers the question, "What is my system's current utilization and remaining capacity?"

Much has been written about monitoring and tuning systems. I will provide a brief synopsis of a few things to keep in mind:

1) Be sure that your system has enough memory. In the course of our consulting and training classes we find an amazing number of memory starved systems. It is very difficult to forecast future capacity if a system is deficient in memory. This is due to the fact that increased memory management means an increase in CPU overhead. It generally also means that disk I/O activity will also go up. Your forecast will be inaccurate if you do not have an adequate amount to begin with.

2) Take your system's "pulse" in regard to the major resource areas: CPU, Memory, and I/O. If any one of these resource indicators look to be excessive, then your system might be quite "ill". How do you define excessive? Your software support representative or the vendor who supplies your performance monitoring tool should be able to help you. Research has quantified various thresholds for things like the amount of CPU busy on overhead, pause for disk, data locality, memory page faults, etc. I would be glad to discuss these with anyone reading this paper.

3) Check the condition of your databases in particular. You will find that some of the disk I/O indicators (#2 above) point indirectly to database inefficiency. If you are trying to forecast future hardware needs for a system that houses very "dirty" databases, then you may be overbuying on hardware. For example, a very large memory configuration on an MPE XL system can hide an I/O problem. If you are not opposed to increase the amount of hardware you need to make up for poor housekeeping, then so be it. Most companies cannot afford to do that. Unfortunately, if you do not perform some kind of diagnostics against your databases, you will not know what kind of shape they are in. This too will cloud your forecast. Some of the tools we see people using are DBLOADNG, HOWMESSY, Adager, DBGENERAL, and Flexibase.

Once you know where your system stands, then you can proceed to the next step: Characterize your workloads.

Workload Characterization

A workload is basically an aggregate of users and programs that constitute a meaningful measure of business activity on the system. A workload might be all the programs that are run in the Finance account. If you knew how much CPU, disk activity, etc. was attributed to this workload, then you would be better equipped to forecast the future. So, if this workload were to increase by 50%, then you could roughly figure the net impact of this workload's growth on the system as a whole.

In order to gather workload information, you will have to utilize a performance monitoring tool. Some tools have workload characterization built in. If you have one that does not support this feature, you might be able to write a program that would go through all the processes for a given interval and sum up key resource utilization. You will want to group these processes so that none fall through the "crack"; all activity, including system overhead, should be accounted for in a workload.

You'll also need to gather data for your system's usage during a representative time. It is surprising, but true, that you really only need a small amount of data during of a typical busy time if you are going to use the modelling method of capacity planning. However, if you use the forecast method, you will need at least a few months.

Having characterized workload information, the next step is to load that data into a spreadsheet for simple modelling.

Workload data, either taken from your performance tool, or from your extract file may then be loaded into a spreadsheet or statistical package. Then you could apply some growth factors and come up with a decent forecast of future CPU needs, for example. Figure 1.1 (later on) will illustrate how workloads can be used to forecast future CPU requirements.

The above steps illustrate a very simple overview of one method of capacity planning. I will now cover a number of different methodologies that are used to deal with future forecasting. The above method described the modelling method, which will be discussed in a bit more detail.

Method Number One: Educated Guessing...also known as "at the mercy of the salesrep"

This is by far the most common way we have seen HP 3000 shops perform capacity planning. The scenario goes something like this:

1) The system manager notes an exponential increase in user response complaints along with near infinite batch completion times.

2) The complaints get increasingly worse; middle managers use the open door policy and "go over" the system manager to voice their complaints of system sluggishness.

3) The top manager calls in the system manager. "What meaneth these complaints? What do you have in mind to improve the situation?"

4) The system manager staggers out of the boss's office, somewhat despairing, thinking to himself, "which floppy disk did I put my resume on?"

5) Reality hits the next morning and the system manager fires up the monitoring tool and produces a report that shows the system 90% busy for most of the day. His conclusion: "This is a no-brainer, we need a bigger CPU". He proudly struts into the MIS director's office and reports the news, "We need a bigger CPU".

After all, nobody can argue with the facts. The salesman further convinces them that a 9-something-or-other will solve their problems.

6) A few months later the requisition order is signed, costing a couple hundred thousand for the upgrade.

This scene is not uncommon. A couple problems with this method of capacity planning are as follows:

- It is moderateiy re-active and certainly not pro-active...bad management

- It does not take into account some other areas of performance management like,

  "Which programs are costing us the most CPU over time?"

  "Which applications could be run at a lower priority and thereby spread the CPU 'nuggets' in a way that robs Peter to pay Paul because Peter doesn't need as much as Paul?"

  "What percentage of the CPU busy time was attributed to overhead and memory management; are there consequently areas that could be tuned, or applications that could be improved to save CPU?"

- It does not quantify the amount of CPU necessary to sustain a given level of workload growth over the next few years; it merely hopes, prays, and assumes that this system will carry the company three or so years.

- By the way, where are we going to get two hundred grand?

- Or, in the case of folks who do not have a performance tool, what if they think a bigger CPU will solve the problem, but they really need more memory or a less serialized database?

- Which workloads on the system are most responsible for CPU erosion? Is it the finance department, manufacturing, or order entry?

- What percentage of the 100% busy is really due to our high priority applications like order entry, customer lookups, etc. The fact of the matter is that even a simple batch job will generally puff up the CPU utilization to 100%. One hundred percent busy alone is a very "sandy" foundation on which to build an upgrade case. What about the CPU queue lengths as well as other signs of true CPU shortage like process preemptions, etc.

This kind of capacity planning is dangerous and expensive. It is not the kind of heads-up management that is to be espoused by shops that are taking their jobs seriously...except those which have large amounts of money to "burn"! Educated guessing is very common, however. It is not the method that I recommend.

Method Number Two: Benchmarking

This method of capacity planning does not easily take into account future growth, but can be very accurate in showing you what size system can help you now.

Benchmarking involves re-creating your environment on a hardware configuration that mirrors the kind you envision upgrading to. To really do this right (re-creating your environment EXACTLY), as you might see, can be very involved. You must take as much of your interactive and batch environment as possible and transport them to another system, presumably at a different site.

The most we have seen done of this is folks taking some of their most bothersome batch jobs and running them on a larger 9xx system and walking away drooling at the results. If you are unable to re-create reality as it will be, interactive users, datacomm, et al, then this method could give one a false sense of security.

Sometimes benchmarking can be done in a synthetic way by writing some programs that re-create the environment you are trying to forecast. This is sometimes referred to as load simulation. We use this method in our lab to try the limits of a particular hardware configuration. You simply run your simulated environment, and then increase the number of users (or whatever) for a workload and re-run, taking appropriate response and transaction measurements.

Method Number Three: Trend extropolation/statistical forecasting

This method is described by the ICCM Capacity Management Handbook [1] as follows:

> "It simply involves keeping historical data on capacity utilization, by application, and using this as the basis for predicting future activity. Statistical techniques such as linear regression can be used to develop formulas for computing future loads, or simple arithmetic percentages can be calculated and used to make such projections."

The most common way to perform this method of capacity planning is to first collect data. This should be taught in "HP 3000 101" class; always collect and log performance data on the basis of workload CPU utilization. Without a sizable data trend, it will make future projection more difficult.

Then, the data is presented graphically. This is the best way to visualize patterns of system usage over time. Various peaks and valleys can be observed in the past. Then an extrapolation line can be drawn out into the future. This growth line would take into account a certain growth rate, either from what the average slope of the past trend has been, or modified by you to increase or decrease the rate of growth. Where that line intercepts a critical resource saturation point is the approximate date that you better have money budgeted to deal will application improvement or buy more hardware.

One major downside of this method of capacity planning is that it is difficult, if not impossible, to foretell the effect of additional applications or hardware changes. One writer likened this to navigating a twisting mountain road by monitoring the rear view mirror!

Method Number Four: Analytical Modelling

This method involves collecting individual workload consumption of various resources, specifically disk and CPU, utilizing a set of formulas to apply growth rates, and then come up with key performance indicators. Some of these indicators are transaction throughput, CPU and disk utilizations, and user and batch response times.

Be forewarned that this method is not for the faint hearted, especially if you are looking for tremendous accuracy. The math involved in analytical modelling can range from simple to tough; sometimes the equations look the same upside down as they do right side up! But you really do not need to be a math giant to utilize these formulas.

If you wish to experiment with this method, it is recommended that you obtain a standard book on the subject, "Quantitative System Performance" (2).

A very simple model is a good place to start. Let's say you have measured the CPU utilization of the following workloads on your system (Figure 1.1). The BASE column is the CPU utilization as reported by your performance tool. You load this data into a spreadsheet and apply some simple growth rates which provides the subsequent columns of utilizations. It is easy to see that sometime around May the CPU would be projected to reach saturation (90% for MPE XL, 75% for MPE V).

One could even get trickier and add an application, change growth rates for workloads at differing times, etc. This simple example does not take into account things like memory, disk I/O, file/resource lock contention, etc. It is very simplistic.

However, we have found in our studies, if your application is primarily CPU intensive, this method can be quite accurate. At any rate, it is a good place to start.

	% CPU Utilization							
	BASE	JAN	FEB	MAR	APR	MAY	JUN	JUL
FINANCE	11	12	13	14	15	16	17	18
MANUFACTURING	45	47	49	51	53	55	57	58
TELEMARKETING	8	9	10	11	12	13	14	15
SYSTEM OVERHEAD	3	4	5	6	7	8	9	10
	67%	72%	77%	82%	87%	92%	97%	102%

Figure 1

Keep in mind that this method does not take into account disk I/O and memory constraints. For example, in April you might run out of memory though the system is only 87% utilized.

We have also seen a number of systems that reached saturation in their application's ability to process transactions. The serialization of a data structure or an application can have just as negative effect on performance as can a resource shortage.

A good example of this involves process handling. One application I have seen involves a controlling process that acts as the traffic cop. There may be 20 users submitting their transactions through this one process and life may be rosy. But what happens when the transaction volume increases considerably? A queue of user requests begins to form. Each user then waits (and suffers) accordingly. The application vendor eventually went to a multi-threaded controlling process which could easily handle the increase in transaction traffic.

One customer that we performed consulting for had about one half of their series 960 virtually paralyzed due to application/data locality constraints. Upgrading to even an 980 series 200 would have done little or no good for them.

These limitations have to be kept in mind when using simple modelling for capacity planning.

Steps for Capacity Planning

Here are a few steps to use as an outline for a capacity plan:

1) Get appropriate backing from management. You'll want to address the need to free an appropriate staff member up to deal with the issue. This means time and money.

2) Interview key managers and users to get historical data. This involves taking some human and computer "pulse" points with regards to how the system has been performing in the past.

3) Collect resource usage by application workload. Characterize workloads (aggregates of users and processes) so that they are meaningful from a business perspective and not just from a data processing view.

4) Determine growth rates by asking key management. It is important to factor any new or greatly modified workloads that will enter into the picture sometime down the road.

5) Forecast resource usage. Use modelling or forecasting methods described previously.

6) Validate your forecast by comparing the forecast to actual resource usage.

7) Re-adjust your techniques and keep dialogue open with appropriate managers/users.

It is true that capacity planning often times ends up being an educated hunch. But it doesn't have to be. As you can see from this discussion, if you want to avoid the educated guess method, you will have to obtain a commitment from management (and yourself!) to put forth some time, energy, and money to find a method that will be suitable to your needs.

References

1. The IS Capacity Management Handbook Series, "Workload Forecasting," Volume I.
2. Lazowska et al "Quantitative System Performance: Computer System Analysis using Queueing Network Models". Prentice-Hall, 1984.
3. Watson, Cheryl "Capacity Planning", Watson & Walker, 1989
4. Lund, Robert "The Perils of Flying Blind or Why it is imperative to Have a Performance Monitoring Gameplan", Interact Magazine July, 1991

INTEREX '91 - Paper #3116

MPE XL Performance Considerations in the 90s

by
Stan Sieler

Allegro Consultants, Inc.
2101 Woodside Road
Redwood City, CA 94062
Voice: (415) 369-2303
Fax: (415) 369-2304

0. Introduction

With the release of MPE XL 3.0 and the HP 3000/980-200, MPE XL has come of age. This paper investigates performance considerations in various aspects of MPE XL. Included are: memory access considerations, page fault costs (and avoidance strategies), file system characteristics, and (perhaps most importantly) techniques to take advantage of multiple CPUs (as in the 980-200).

1. Memory Access Considerations

This section deals with two views of memory access: the micro view and the macro view. In the micro viewpoint, we will be looking at what happens when a single word of memory is touched. In the macro view, we will consider how we can take advantage of machines with up to one billion bytes of RAM memory.

1.1 Memory Access: Micro View

This section will look at the work done by the computer to accomplish a simple load from memory.

On PA RISC computers, such as the HP 3000/9xx, HP9000/8xx, and HP9000/7xx, memory is accessed only via Stores and Loads.

(PA means Precision Architecture, Hewlett-Packard's name for their RISC design. It was originally called HP High Precision Architecture, then HP Precision Architecture, and is now referred to simply as Precision Architecture. RISC means Reduced Instruction Set Computer, and CISC means Complex Instruction Set Computer.)

The PA RISC philosophy of restricting memory access to only the load/store instructions is in sharp contrast to most CISC computers. The Classic HP 3000 had a number of instructions that accessed memory in addition to performing other work.

The primary reason behind this strict interface with memory is that fetching data from memory is very slow, compared to the speed of ordinary instruction execution.

Note: parts of this section are greatly expanded in the book "Beyond Risc".

1.1.1 Code Fragment

The following description of what the PA RISC hardware does to implement a single LDW instruction will lay the groundwork for the rest of this micro view section. (The LDW instruction loads 32-bits of data from the location specified by a virtual address into one of 32 general purpose registers.)

For this example, assume that the code is fetching the value of "old_ktr" from the following Pascal/XL code fragment:

```
var
   ktr      : integer;
   old_ktr  : integer;
...
ktr := old_ktr;
```

The code emitted for the above statement would be:

```
LDW      12(0,27),1
STW      1,8(0,27)
```

The LDW instruction is read: load 32 bits from virtual memory (at the address which is the sum of #12 and the value in General Register 27 (r27)) into General Register 1 (r1).

(Note: for the rest of this paper, the phrase "General Register" will be abbreviated to "r", as in: r1)

Thus, this LDW instruction already causes the hardware to do some arithmetic in the process of generating the final address from which to load. The value (decimal 12, or #12) that is added to the base register (r27) is known as the displacement. This form of addressing (displacement plus base register) is one of several available in PA RISC, and is probably the most common in code generated by the Pascal/XL compiler.

For this example, assume that the final 64-bit virtual address specified by the "LDW 12(0,27)" instruction is: $5f1.$40331014

1.1.2 Virtual Addresses

PA RISC is generally described as having 64-bit virtual addresses. In the LDW instruction above, the base virtual address is specified by the "(0,27)" portion of the instruction. The two numbers in parenthesis are the "s-field" and the "General Register" (r), respectively. The s-field can have values in the range 0 to 3, and the General Register has values in the range 0 to 31.

Note: the computer has eight space registers (sr0, sr1, ..., sr7), but sr0 and the last four (sr4..sr7) cannot be directly used in load/store instructions.

The general form for a simple LDW instruction is:

```
LDW  d (s, b), t
```

Where:

```
d       = displacement (-8192..8191)
s       = s-field (either a "0", which is treated special,
          or 1, 2, or 3 (selecting space register sr1,
          sr2, or sr3))
b       = base register (0..31 for r0..r31)
t       = target register (0..31 for r0..r31)
```

Note: when an instruction is "viewed", most tools do not bother showing the "sr" and "r" characters. They are implied by their position in the instruction. Debug/XL and Pascal/XL omit the "sr" and "r". Avatar (from Software Research Northwest) omits them by default, but has a mode to include them, if desired.

When a load/store instruction specifies sr1, sr2, or sr3, the final 64-bit virtual address is assembled as:

```
[sr1 (or sr2 or sr3)] . [gr##]  +  d
```

Or, in other words, the upper 32 bits (called the "space id") is the value of the specified Space Register, and the bottom 32 bits (called the "offset") is the value of the specified General Register plus the offset value ("d"). (There are some addressing modes where "d" is considered to be a register instead of an immediate value, but this paper will not be going into these.)

Whenever a load/store instruction specifies Space Register 0 (sr0), the hardware handles the building of the final 64-bit address quite differently: the bottom 32 bits are calculated the same as above (i.e.: gr## + d), but the top 32 bits are calculated as follows:

1) Take the upper two bits of the value in the specified General Register. (This can only be binary values 00, 01, 10, or 11.)

2) Add the value 4 to the value found in step 1. (This can only result in 4, 5, 6, or 7).

3) Use the value of sr4, sr5, sr6, or sr7 for the upper 32 bits (if the sum was 4, use sr4; if the sum was 7, use sr7).

This form (0, gr##) of a virtual address is called a "short" virtual address. The other form (1/2/3, gr#) is called a "long" virtual address. In the debugger, Debug/XL, addresses that are specified with a dot (.) in the middle are long addresses (e.g.: $a.c0000000), and addresses specified with no dot are short addresses (e.g.: $c0000000).

Short virtual addresses may seem complicated, but they provide a method of accessing portions of different spaces in an efficient manner. In most native mode programs, the vast majority of all addresses used are short addresses.

1.1.3 Finding the Data

Once the final 64-bit virtual address has been constructed by the hardware, it has to determine several more things:

1) Is the data in memory?

2) Are you authorized to access it?

3) (for a store) Should you be interrupted because of a data breakpoint?

Memory (physical and virtual) is broken into uniform sized chunks called "pages". The size of a page is 4,096 bytes. (The hardware thinks of pages as being only 2,048 bytes, but we can ignore that for this discussion.)

A page is the basic unit of memory that is handled by the memory management software. If a page of virtual memory is "present", then it occupies a page of physical memory (RAM). If a page of virtual memory is "absent", then the operating system will have to read it from disc if we want to access even a single byte in it.

Given the concept of a "page", the "is the data in memory" question can be rephrased:

1) Determine what virtual page the address resides in.

2) Is that page in memory?

I have discussed the mechanics of mapping virtual addresses to physical addresses in various papers, as well as in "Beyond RISC", and do not want to be redundant here. So, let's assume that the page is in memory, starting at physical address $40000, and that we are authorized to access it.

Once the physical address of the page has been determined, the hardware adds to that base address the bottom 11 bits of the virtual address. Since the final 64-bit virtual address for our LDW is $5f1.$40331014, the bottom 11 bits would be: $014. The sum of hexadecimal $14 and $40000 is $40014. The hardware now has the physical byte address of the data.

1.1.4 Cache

If the hardware was to attempt to fetch the data directly from memory at this point, the following would happen:

1) The CPU marks the target register (r1 for this example) as "busy" (this will prevent subsequent instructions from trying to access it until the data is actually ready).

2) The address ($40014) would be put on the "bus".

3) One of the memory controllers would (hopefully!) recognize the address as being in the memory attached to it. (If no controller, or more than one controller, recognized the address, the computer would probably die.)

4) The controller would fetch the 32-bit word and put the value on the "bus".

5) The CPU would take the value from the bus and put it into the target register (r1 in this example) and marks it as "not busy".

In addition to being complicated, the above process is slow. If the next instruction after the LDW tried to access the target register (r1), then it would be blocked until the data arrived in the register. This blocking is called "register interlock".

The time required to actually fetch data from memory does not seem to have been published by HP. My timing tests on a 925 and 935 show that fetching a word from memory takes 32 cycles.

In order to solve the memory access performance problem, most computers have added a "cache", which is sort of a table of recently accessed memory addresses and their values. PA RISC has done the same. When the CPU is about to fetch a word of memory, it checks the cache to see if the data is in it. If it is, then the value remembered there will be used. If it is not, then it proceeds as described above. When data is in the cache, it takes two cycles to be brought into the target register. This reflect one (or more) orders of magnitude of performance difference between cache and the memory controller!

All PA RISC models have two caches: the Data Cache and the Instruction Cache. Data (information accessed with loads/stores) is handled by the Data Cache. Instructions (code) are handled by the Instruction Cache. On a most models, these two caches are in separate areas. On a few models, they are combined into one.

Caches are very expensive, compared to the cost of RAM chips. As a result, hardware designers must compromise between the desire for a big cache and an affordable computer. The size of the cache varies between the various 9xx computers. The following table shows the cache sizes for most of the known models:

CPU Model

Cache Type	920	922	925	930	932	935	948	949	950	955	958	960	980-100
Data	64	+	+	64	+	+	512	128	+	128	512	512	512
Code	64	+	+	64	+	+	512	512	+	128	512	512	512
totals:	128	32	16	128	128	128	1024	640	128	256	1024	1024	1024

Note: all cache sizes are in units of Kilobytes.

Note: "+" signifies a combined Data and Instruction cache.

Note: the 980-200, since it contains two 980-100 CPU boards, could be thought of as having 1024k/1024k cache sizes, but this would be misleading because it does not provide that much better a chance that your data will be in cache, as a single instruction executing on one of the processors will ONLY search the cache for that processor, not for both processors)

The bigger the cache, the better the chance that data you want will be in it.

One technique that keeps the price of the cache down, and also helps increase the efficiency of typical programs, is that the cache is organized in "lines", not in "words".

Consider a cache that is implemented as a table like the following, which is a paired list of 64-bit virtual addresses and 32-bit data:

64-bit Address $xxxxxxxx.$xxxxxxxx	32-bit Data
$000005f1.$40331014	$12345678
$000025f1.$12300000	$00000000
$00000323.$403f0014	$10002038
$000105f3.$60f31014	$20202020

In the above example, it would require 96 bits for each word of data stored (64 bits for the address plus 32 bits for the data). This represents a ratio of 96/32 bits, or 3. (I.e.: it costs 3 bits for every 1 bit of data.)

Instead, PA RISC cache is organized like the following table:

60-bit Address $xxxxxxxx.$xxxxxxx0	16 bytes of data			
$000005f1.$40331010	$00000278	$12345678	$23900fff	$44409aaa
$000025f1.$12300000	$00000000	$00000000	$00000000	$00000000
$00000323.$403f0010	$10002037	$10002038	$10002039	$1000203a
$000105f3.$60f31010	$5354414e	$20202020	$5349454c	$45522020

Note the "0" in the last (right-most) nibble of the address. Every cache line starts at a multiple of 16. The word our LDW is trying to load is the second word of the cache line. Whenever the CPU tries to access a byte that is not in cache, 16 bytes will be fetched by the memory controller and put on the bus. The CPU will put all 16 bytes in the cache. This means that for every 128 bits of data (8 * 16), the cache has 60 bits of address, or 188 bits per line. This is a ratio of 188/128, or 1.468, which is much better than the first type of cache organization.

Most models of PA RISC have a 32-byte cache line, but 16 bytes seems to be the minimum size (and is the 930's size), all larger cache line sizes are a multiple of 16. *(Note: there are other aspects to cache organization, but this model will suffice.)*

It is the concept of the cache line that is responsible for one of the more annoying characteristics of PA RISC: words (32-bit data) can be loaded/stored only into addresses that are a multiple of 4. Half words (16-bits) can be loaded/stored only into addresses that are a multiple of 2. Double words (64-bits) can be loaded/stored only into addresses that are multiples of 8. (Bytes can be loaded/stored anywhere, since every byte address is a multiple of one!)

The basic reason: an element being loaded/stored is not allowed to cross a cache line. (I.e.: be partially in one cache line and partially in another.) Because cache lines start at multiples of 16 (or 32 or 64), every address that is a multiple of 4 is guaranteed to never cross a cache line. (And similarly for multiples of 2 and 8).

Whenever the "cost" for an instruction is discussed, the issue of the CPU's pipeline arises. Although the PA RISC computers have an instruction pipeline that is from three to five deep, in general one instruction completes every cycle. So, for the purpose of discussing the cost of an instruction, we can usually ignore the pipeline and think of the CPU as executing one ordinary instruction per cycle.

My timings show the following "cost" for a few instructions and events (times derived on a 3000/925 and 3000/935):

#Cycles	Instruction/Context
1	NOP & most instructions that do not have register interlocks.
1.5	LDW (and other Loads), ignoring the location of the actual data. This is the minimum cost.

+1 Register interlock. The "+1" means that one extra cycle is used getting the data (requested by the previous instruction) from the cache and into the register.

The following sequence

```
    ; assuming cache line for r27 is in cache.
    ; instruction       ; should take:
LDW  0(0,27), 31        ; 1.5 cycles
ADDI 1, 31, 31          ; 1
                        ;-------------
                        ; 2.5 cycles total
```

Actually takes 3.5 cycles, where the extra cycle is the register interlock. *(Note: assumes that the cache line for the address in r27 is already in cache.)*

32 Cache miss penalty.
This penalty is paid only when an instruction tries to access a register that was loaded into and the data is not yet in memory, as show below.

```
    ;assuming cache line for r27 is NOT in cache.
    ; instruction      ; should take:
LDW  0(0,27), 31       ; 1.5 cycles
ADDI 1, 31, 31         ; 1   cycle
```

Actually 34.5 cycles. The extra here is the 32 cycle penalty for a cache miss.

730 TLB miss penalty. If a virtual address is not found in the cache, the hardware checks the Translation Lookaside Buffer (TLB) in an attempt to quickly determine the physical memory address that the data resides in. If there is no entry in the TLB for the virtual page that the address is a part of, a "Data TLB Miss Fault" (DTLB miss) is generated, and the operating system is invoked to determine where (and if!) the virtual page resides in physical memory.

806900 Page fault penalty. If the virtual address is part of a virtual page that is not currently in memory, the hardware traps to the operating system, and asks for the page to be read from disc (or allocated and filled with blanks or nulls), then the instruction is re-executed. (On a 935, 806900 is about 53 milliseconds.)

Note that an LDW (or LDH or LDB) will not always "cost" 32 extra cycles. Depending on how well the compiler emitted the code, you might see an LDW costing no extra cycles at all! The following example demonstrates how this can be:

```
    ;assuming cache line for r27 is NOT in cache.
    ;and assuming no one has recently loaded into
    ;r20 (or r31).
    ;instruction      ; should take:
LDW  0(0,27), 31      ; 1.5 cycles
ADDI 1, 20, 20        ; 1   cycle
ADDI 1, 20, 20        ; 1   cycle
ADDI 1, 20, 20        ; 1   cycle
ADDI 1, 20, 20        ; 1   cycle
ADDI 1, 20, 20        ; 1   cycle
ADDI 1, 20, 20        ; 1   cycle
```

```
ADDI 1, 20, 20      ; 1   cycle
ADDI 1, 20, 20      ; 1   cycle
ADDI 1, 20, 20      ; 1   cycle
ADDI 1, 20, 20      ; 1   cycle
ADDI 1, 31, 31      ; 1   cycle
                    ; should be 12.5 or 44.5???
```

This actually takes 34.5 cycles.

The above example shows how the CPU will continue to work, as long as no instruction tries to touch a register that has been loaded into. The optimizer for most Native Mode languages tries to put as much of this kind of work as possible between a "load" instruction and the next instruction that will use the loaded register.

1.1.5 Data Alignment

The previous section mentioned why PA RISC computers cannot load a 32-bit word from an arbitrary byte address. Instead, such loads require addresses that are 32-bit aligned (i.e.: a multiple of 4).

Most of the Native Mode compilers have a means of letting the user specify that certain 32-bit variables are not aligned nicely, and they can emit lengthier code to access the variables safely.

In migrating programs from the Classic HP 3000 (which did not have this restriction), some users have told the Pascal/XL or COBOL/XL or FORTRAN/XL compilers to use "$HP3000_16" alignment, which causes the compiler to emit three instructions instead of one for every load/store.

1.1.6 Short Signed Integers

Many Native Mode languages support the concept of a 16-bit signed integer data type (e.g.: shortint in Pascal/XL). Users who are trying to conserve memory by using shortints instead of integers may not be aware of the performance considerations. Loading a shortint into a register typically requires two instructions: one for the 16-bit load, and one to extend the sign (usually an EXTRS instruction). The sign extension converts the 16-bit number into a signed 32-bit number, which is necessary before doing any arithmetic with it.

1.1.7 Putting It All Together

The two words that best characterize what can be done about memory performance are: locality and alignment.

The CPU pays a performance penalty when it accesses memory. It pays a much bigger penalty if the new data is not in the same cache line as previously used data. Finally, it pays a tremendous penalty if the data is on a page that is not in memory. Thus, the locality (degree of closeness) of your memory accesses is critical. You can take these steps to group together data that is likely to be accessed at the same time:

1) Arrange global variables so that commonly used data is grouped together. (Of course, this contradicts one of the guidelines of readable programs ... sigh)

2) When defining a record (or struct) in Pascal (or C), group together the fields that are used together.

3) When declaring a matrix (doubly dimensioned arrays) in C, FORTRAN, or Pascal, consider how you will be typically accessing them: along a row or along a column? If possible, make the common path be along a row, as a matrix is stored one row after another in memory (for C and Pascal). FORTRAN stores a matrix the opposite, in column-major order, so your FORTRAN program should have the common path be along the column, not the row.

4) In an ordinary Pascal/XL record (and in all C/XL structs), the compiler will silently waste bytes between fields in order to keep the fields aligned on the most efficient boundaries. One way to override this is to use the Pascal/XL extension "crunched record". If any of your code has crunched records, examine them closely to determine if any 32-bit fields cross 32-bit boundaries.

Data that is aligned nicely loads quicker (a three to one ratio). If you have any 32-bit variables (or types) that have been declared as having 8-bit or 16-bit alignment (as in $HP3000_16), examine them and do either:

1) Remove the special alignment qualification, so the variables will be treated as 32-bit aligned only. *(Note: if you remove the qualification, and then attempt to access a 32-bit variable that is not 32-bit aligned, your program will abort with an "invalid address alignment" error.)*

2) If a procedure/function/subroutine has a reference parameter which is a non-32-bit aligned 32-bit variable, copy it into a local 32-bit variable (which is a normal 32-bit aligned variable).

Variables that are 16-bits in size (e.g.: shortint in Pascal/XL) have similar alignment problems as 32-bit variables, with the difference being that their addresses should be 16-bit aligned (i.e.: an even byte address).

1.2 Memory Access: Macro View

The HP 3000/980 hardware (all models) will support up to one gigabyte of memory (1,073,741,824 bytes). The second question this should raise is: how can I take advantage of this much memory?

The first question is: how much does one gigabyte cost? Luckily, there is more than one manufacturer of memory boards for the HP 3000/950, /960, and /980 (unlike the smaller models). This leads to price (and warranty) competition that benefits the users. HP sells 980-100 memory at $1,500 per megabyte; Kelly Computer Systems is less than half the HP price. Thus, the cost of one gigabyte of memory ranges from $750,000 to $1,500,000!

Regardless of the total amount of memory on your HP 3000/9xx, its cost, or who manufactured it, one thing should be clear: there is a lot more memory available today on any current HP 3000/9xx model than on any Classic HP 3000. (Except, of course, for a Classic HP 3000/70, which could be configured with a Kelly RAM Disc of up to 120 megabytes.)

So, if we are now using computers with so much memory, how do we take advantage of it?

On the Classic, programs were limited to 64KB of directly addressable memory. With cleverness (and with a speed penalty), extra data segments could raise this limit up to (in theory) 65 megabytes.

On MPE XL, Native Mode programs can easily access up to one gigabyte of memory directly, two gigabytes with very slight programming changes, and dozens of gigabytes with clever programming (by opening dozens of large mapped files).

When programs are being migrated from MPE V to MPE XL, the following steps are usually followed:

1) Run the program in Compatibility Mode;

2) Recompile the program in a Native Mode language, making the minimal changes necessary.

Unfortunately, this is where most users stop. As my partner, Steve Cooper, has said: a third migration is necessary to fully take advantage of the new architecture (and MPE XL).

Look at your programs closely, and try to determine if there are any places where Classic memory constraints severely affected the design. Examples include: disc-based lookup tables and iterative calculations instead of table lookups.

2. Page Faults

Virtual memory is wonderful. It allows the program (and programmer) to think that the computer has much more memory than it actually does. However, this is not without a cost. Section 1 briefly discussed the cost of a page fault (~800000 cycles). That cost should not be taken as gospel, as many different factors can influence the actual cost. These include:

1) Does the page reside on disc (or was it never allocated)? The former requires a disc read, while the latter does not, and instead a page of memory will be allocated and filled with blanks or nulls.

2) Has the system been getting a lot of page faults? If it has, it may not service your process's fault for awhile.

3) Where on disc is the page? Fetching the page will involve contention for the disc, controller, and bus. At the disc, rotational latency, seek time, and data transfer rate all affect the cost.

4) Where should the page be put in memory? On a busy system, the memory manager will be slower in deciding. Worse yet, it may throw out one of your soon-to-be-accessed pages to make room for the current page.

The best strategy in minimizing page fault costs is to avoid them. Some of the same techniques used to optimize cache line hits (section 1) can be used in a more global scale here. In cache optimization, you are trying to keep items in the same 16 (or 32) byte cache line. In virtual memory optimization, you are trying to keep data that is used "together" within a 4,096 byte virtual page.

We have one page fault avoidance strategy available that has no cache-line equivalent: prefetching. Prefetching is the act of asking MPE XL to bring one (or more) virtual pages from disc into memory before we actually need them. Even the

Classic HP 3000 had some degree of programmer controlled prefetching (i.e.: the FREADSEEK intrinsic).

On MPE XL, there are three methods of prefetching: automatic prefetching (done by the file system), the FREADSEEK intrinsic (used by the programmer), and the internal "prefetch" procedure (callable by the aggressive programmer using privileged mode).

As my previous papers have shown, the automatic prefetch done for us by the file system seems to bring somewhere between 1 and 8 virtual pages of our file at a time. However, we have no control whatsoever over this function. (Even choosing random versus sequential reads has no impact.)

The internal routine, prefetch, is the most efficient method of requesting MPE XL to bring in one (or more) pages of data before you actually need it. However, HP has chosen to cloak the internals of MPE XL in secrecy, rendering this routine difficult for the average programmer to access. For now, we'll leave it with a warning: if you determine how to call the routine, please be cautious: attempting to prefetch very large amounts of data (e.g.: a megabyte) can result in system failures.

The FREADSEEK intrinsic is a somewhat slower method of requesting prefetching. It is also much more awkward to use. To prefetch using FREADSEEK, decide if you are going to be thinking of the data in terms of byte offsets within a mapped file or as records.

The most important fact to remember when adding either kind of prefetching to a program is this: ask for the data sufficiently ahead of time so that MPE XL has time to bring it in to memory before you actually need it. If you don't ask for the data soon enough, your program might actually run slower: in addition to the time required to initiate the prefetching, you would be spending time with page faults.

Let's consider the byte offsets viewpoint first. If we assume that your program is processing data (from a mapped file) at the rate of bytes_processed_per_millisec, and that requesting a quantum (a "chunk") of disc data of fetch_quantum_bytes (say, 100,000) takes millisecs_to_fetch_quantum, then we have the information necessary to determine how far ahead of time the prefetch should be issued. We can calculate the time required to process one quantum of data (once it is in memory) as follows:

```
millisecs_to_process_quantum := fetch_quantum_bytes div
                                bytes_processed_per_millisec;
```

We will need to prefetch a "quantum" at the larger of the above value and millisecs_to_fetch_quantum before we need it. If the millisecs_to_process_quantum value is larger than millisecs_to_fetch_quantum, then we are CPU bound. On the other hand, if millisecs_to_process_quantum is less than millisecs_to_fetch_quantum, then we are I/O bound. If the two values are the same, then we are CPU and I/O bound.

If we are CPU bound (or both), then we can simply prefetch one quantum ahead of where we are now. (E.g.: prefetch from byte 100,000 when we are about to start processing byte 0. Prefetch from byte 200,000 when we are about to start processing byte 100,000.)

If we are I/O bound, then we will have to prefetch several quantums ahead of time. The actual value is:

```
bytes_ahead := bytes_processed_per_millisec
               * millisecs_to_fetch_quantum;
```

Remember that the bytes_processed_per millisec is subject to large changes from the low-end of 3000/9xx computers to the high-end.

As a final guideline, I would suggest a limit of one million for both the quantum and for the "bytes_ahead" value. When either number becomes much larger, you run the risk of the memory manager undoing some of your prefetch efforts.

The record viewpoint is similar, with the calculations being done in terms of records rather than bytes. The record viewpoint makes the actual prefetching easier. Using FREADSEEK, the prefetching of "n" records starting at record "d" would be:

```
records_per_freadseek:=(16384 + record_size_in_bytes - 1)div
                                record_size_in_bytes;
   while n > 0 do
      begin
      freadseek (file#, d);
      if ccode <> cce then
         complain (or ignore, error possibly due to
                   prefetching past the end of the file);
      d := d + records_per_freadseek;
      n := n - records_per_freadseek;
      end;
```

The above algorithm attempts to avoid calling the FREADSEEK intrinsic for every record. After all, MPE XL will typically bring in about four virtual pages for each FREADSEEK. Since four pages is 16,384 bytes, it means that (for most files) several records have been prefetched. The actual number of pages definitely varies, but tests show that four virtual pages is a reasonable average.

Of course, the above analysis is concerned with the operation of a single process, excluding all other processes on the system. If our program is running on an I/O bound system, then is will usually take more time to fetch "quantum" bytes from disc, so we should be asking for the data even earlier.

Another consideration for prefetching is that the earlier we ask for the data (particularly on a multi-user machine), the more chance there is of the memory manager deciding to throw out some of our own prefetched pages to make room for another fetch request (ours or from another process).

Lastly, if our program is to be running on a system that is considered "memory bound", then we probably don't want it to be prefetching, as there is no memory to spare.

3. File System

MPE XL provides Native Mode programs with three methods of opening existing disc files: the FOPEN intrinsic, and two variations of the HPFOPEN intrinsic, as shown:

1) FOPEN, using the file name
2) HPFOPEN, using the file name
3) HPFOPEN, using the ufid (Unique File IDentifier) of the file.

Some publications have touted the third method as being the most efficient, so I thought it should be tested in real life.

First, an explanation of how the three methods work is needed.

FOPEN calls an internal MPE XL routine called "fopen_nm", which calls another internal routine called "open_old". Open_old searches the directory for the filename passed into FOPEN. If the filename is found in the directory, its entry will contain a ufid. A ufid is a 20-byte data structure that can be thought of as a "pointer" into another data structure called the "label table". It is the label table that keeps track of where the various portions of the file's data reside on disc.

HPFOPEN (with a filename) calls open_old directly (without calling fopen_nm), and the file open proceeds as above.

The third method (HPFOPEN with a ufid) also calls open_old, but completely bypasses the directory search. The ufid is treated as a pointer into the label table. It is checked for validity (to prevent random generated ufids from causing problems), and is then used just like the ufid that the other two methods fetched from the directory.

In principle, the third method should be quicker than the first two methods because the directory search phase is eliminated. Unfortunately, timings don't bear this out.

The following table shows the number of instructions required for each of the three methods to open a permanent file in the logon group (specifying only a file part, not a group name or account name). Two scenarios were tested for each method. The first was to open a file which was not currently open by anyone else in the system. The second was to open a file that another process already had open.

File: foo	# Instructions	
Method	(when not in use)	(already in use)
FOPEN	28824	25829
HPFOPEN-title	28855	25824
HPFOPEN-ufid	29551	13383

Note: the "# instructions" is not the same as "# cycles". As discussed in section 1, an instruction can take more than one cycle. The actual time required by any method will vary, depending on the cache miss ratio and number of page faults (among others). The instruction count does not reflect the cost of any page faults, TLB misses, or other interruptions. Further, the instruction count varies about 4% across multiple runs (perhaps partially due to system logging).

The above results surprised me, as I expected the HPFOPEN-ufid method to be much faster. I can only conjecture that the intrinsic is subjecting the user-supplied ufid to an expensive validity check, which is not applied to ufids found in the directory search. The low cost of the "already in use" HPFOPEN-ufid could, perhaps, be because the open_old finds an instance of the already-open file's ufid in a table somewhere, and can bypass the validity checking.

I increased the cost of the directory search by testing the opening of a file in another account.

I did one last test, comparing opening permanent files to temporary files:

File: acctjobs.pub.sys	# Instructions	
Method	(when not in use)	(already in use)
FOPEN	33246	36180
HPFOPEN-title	34508	37442
HPFOPEN-ufid	33010	13704

Note that the number of instructions for the HPFOPEN-ufid (when not in use) is now the smallest of the three methods, although just barely.

I cannot explain why the "already in use" open now takes more time for the FOPEN and HPFOPEN-title methods, when it took less time with a non-fully-qualified filename. I repeated the tests and obtained similar results.

File: temp	# Instructions	
Method	(when not in use)	(already in use)
FOPEN	20317	20194
HPFOPEN	21641	21516
HPFOPEN	22141	17972

The results of this test taught me one lesson: if my program needs to pass data to another program (in the same job/session), a temporary file is cheaper to open, by up to 30%.

4. Multiple CPUs

Multiple CPUs are not new. Commercial computers with more than one tightly coupled CPU have been available since the mid 1960s. (I learned ALGOL using a dual CPU Burroughs B6700 in 1970.) Even the venerable HP 3000, model II, had some support for multiple CPUs (although it was never released in a two CPU model).

With the advent of the HP 3000/980-200 (a dual processor system), we can now start taking advantage of multi-processing. When looked at from the viewpoint of speeding up a single process, a multiple CPU machine is a waste. (In fact, it costs a small amount of performance, compared to a single CPU machine with a single-CPU operating system.) However, many programs today can be restructured to take advantage of multiple CPUs.

Consider a night-time batch program that runs in, essentially, stand-alone mode. (Perhaps it is sequentially reading a TurboIMAGE master dataset, looking for account numbers. For each account number found, it follows a detail chain, looking for unpaid invoices. As unpaid invoices are found, they are written to a report.) This typical program will run for a short time and then block, waiting for data from disc. When the data is ready, the process resumes running. This type of program is easy to recognize using any performance tools the CPU time is substantially less than the elapsed time, even when run stand-alone.

If the program can be broken into two parts to run in parallel, the second one can usefully utilize the I/O wait time of the first one, and vice versa. Taking our earlier example, if the first program was tasked with reading the first half of the master dataset, and the second program was tasked with reading the second half, then the set of two programs will probably finish much sooner than the original one. Given a program that ran, by itself, in 3600 seconds of elapsed time, using 1200 seconds of CPU time, about 2400 seconds were spent waiting for I/O.

If the program is split in two, on a single CPU machine, the best-case estimate for the elapsed time would be about 3000 seconds. (If the two were run in succession, they would each take 1800 seconds elapsed and 600 CPU, each waiting for 1200 seconds. The best-case assumption is that, when running in parallel, all of the CPU for the second process is done while the first process was waiting for I/O, representing a savings of 600 seconds.) With a dual CPU machine, the best case assumption drops to 1800 seconds. (Of course, this isn't a very likely result, as the I/O requests will overlap, causing contention for I/O resources.)

On any MPE XL machine, a program that is not CPU bound is a potential candidate for splitting into multiple programs. When looked at as a stand-alone application, the major drawbacks of splitting a program is increased MPE XL overhead (dispatcher), and increased program complexity. As part of a busy machine, splitting a single program can have potentially hurt overall system performance (because there is now one more hungry mouth to eat CPU time).

The programs that will benefit the most from a second CPU are those that are CPU bound.

5. Summary

The aspect of memory usage that has the most impact on performance is locality. Performance gains, both small and large, can be achieved by keeping locality in mind when designing algorithms and data structures.

When compared to the Classic HP 3000s, memory on an MPE XL machine is far cheaper and much more plentiful than ever before. We should keep this in mind when designing new algorithms.

Multiple CPUs are, finally, a reality on the HP 3000. I look forward to writing, and seeing others write, programs that can truly take advantage of the new power this will offer.

Data Integrity and Recovery: The IMAGE/Adager approach.

F. Alfredo Rego

Adager Lab Manager

Adager
Sun Valley, Idaho
83353-0030
U.S.A.

Telephone +1 (208) 726-9100 Fax +1 (208) 726-8191

Hewlett-Packard's IMAGE Database Management System has many features designed to enhance its integrity. Unfortunately, despite our best efforts, IMAGE databases may suffer structural damage. Adager Corporation's Adager (The Adapter/Manager for IMAGE Databases) has many features designed to quickly detect, diagnose and correct (if possible) database failures.

Reliability, speed and ease-of-use are extremely important when dealing with faulty database structures. We know what we want: our databases should remain *correct* and *available* despite failures. This requires a lot of work. Keeping our guard up means several things. First of all, we should try to *avoid* problems. Unfortunately, we may fail despite our best efforts. If so, we should *locate* and *fix* the errors and we should *remove* the causes.

As Fred White says, "Your choice of method of repair is governed by a) the nature and extent of the damage, b) your knowledge of IMAGE, c) your knowledge of the data in the database and d) the tools at your disposal together with your ability to apply them."

Adager provides a wealth of database-therapy functionality. But, before reviewing Adager's specific methodology, let's review the general subject of database therapy.

To be ON or not to be ON: that is the BIT question!

When a bit stands by itself, without any relationships whatsoever with other bits, this "on/off" question has no *correct* answer. But when a bit is part of a redundant conglomerate, the question's chance of having a correct answer increases (as a function of the number of redundant "colleagues" in the conglomerate). For example, consider "parity bits". With one parity bit, our only hope is to detect a one-bit mischief; with more parity bits, we can hope to also point our fingers to the offending bit and (if we are really smart and if we want to pay the price in terms of extra bits) to correct it!

IMAGE databases have built-in redundancies (involving critical fields, chain pointers, chain

counts, entry counts and "presence" bits). It is mathematically very simple to detect inconsistencies among these redundant IMAGE elements. The hardware (any flavor of HP3000 computer) that supports IMAGE databases has all kinds of error-detecting and error-correcting components (such as memory, discs, tape drives, and so on).

Keeping your guard up

IMAGE is extremely *reliable* (since it has few bugs of its own) and *resilient* (since it shows "compassion" toward the faults of its underlying hardware and operating system). IMAGE, by design, contains spare resources (for instance, the redundancy between pointers and search fields). Nevertheless, IMAGE structures do fail. Such failures may go undetected or they may show up on a user's screen or report. Depending on how your application programs deal (or fail to deal) with database problems, you may end up with a nicely-logged set of diagnostics that pinpoint specific chains or you may find yourself with aborted jobs that say nothing at all. In any case, you must always be on your toes!

Prevention

A bit of prevention can save many megabytes worth of reloading. As an example of an elementary precaution, consider the electrical power that makes our computers tick. Since my early days in Guatemala, back in the seventies, I always have insisted on a healthy standard for my HP3000 computers: uninterruptible power supplies (UPSs) backed up by diesel generators. I am dismayed to see how many big companies (with vast resources) choose not to invest in these elementary measures. I am also very pleased to see that some smart organizations (notably Hewlett-Packard's response centers) have opted for outstanding batteries of UPSs and powerful auxiliary generators.

Fault tolerant technologies typically carry performance penalties. This applies to fault tolerance toward lousy electrical power as well as to fault tolerance toward lousy systems or applications programming (not to mention lousy hardware). We must decide the relative worth of the various options (for instance, consider the extremes of never backing up at all versus backing up every five minutes).

Backup and (hopefully) recovery

Backup is one thing; recovery is another thing altogether. I quote from my *Practitioner's Experiences* essay:

> "People can store things on the wrong set of tapes, clobbering whatever data was on the tapes! And people can restore from the wrong set of tapes, clobbering whatever data was on disc! A well-organized tape-library system is a good investment, especially if it is itself computerized (after all, you are trying, precisely, to protect yourself from sloppy operators...)
>
> People can physically keep your backup tapes in a hostile environment, thereby rendering them useless. I recommend professional handling of your off-site tape storage. And I recommend that you periodically check the validity of the tapes kept both on-site and off-site. What would happen if you had to restore some file from

some tape that was physically impossible to read [or that did not quite contain what it was supposed to have, due to a bug in the backup software]?

People can fail to backup the system at all. Whether they do it intentionally or innocently is irrelevant: the catastrophic consequences are the same. How do you know that the tapes that are supposed to contain your full sysdumps really contain them? Backing things up is a chore. How do you know that your people are not taking shortcuts? I know a user who has an HP3000 machine dedicated to just a single purpose: a reload from the sysdump tapes produced by other computers. If any reload has any difficulties whatsoever, he takes immediate action to correct the problem while it is *important* but not *urgent*. Most people wait until a problem is *important*, *urgent*, and *impossible* to solve within the given time/resource constraints."

What if you lowered your guard and you now face database damage?

Broken chains are painful. Why should we add insult to injury using primitive methods to detect and fix broken chains? When we are forced to embark on a fixing mission, we are naturally nervous. Particularly if we are not *gurus*. Even bit pushers should feel uncomfortable dealing with cryptic listings in octal (or hexadecimal), since a minor human mistake may have disastrous consequences. All of us have experienced, at one time or another, the frustration that unfriendly systems generate. We all deserve better human interfaces. There is no reason to compound an already tense situation with temperamental database therapy software!

The first step is to detect the errors and to diagnose the faults. Then, we must devise a strategy to recover the database. One approach is to restore the whole database from the latest backup (assuming that the fault happened after such backup) and then use DBRECOV to apply all logged transactions. If this is unfeasible (due to the unbearably long times involved, or to the lack of backup, or to the lack of log files), then we must treat the database in an OnLine manner. Regardless of our choice of methodology, our objective is to put the database back into a stable state. Ideally, we should also find the culprit and remove it so we don't go through the whole thing over and over again.

One of the toughest challenges facing the database therapist is the correction of discrepancies between critical fields and their associated pointer structures. Fortunately, redundancy comes to your help (if the pointers are broken but the SearchField is not, you can reconstruct the pointers; if the SearchField is broken but the pointers are fine, you can reconstruct the SearchField -- a little trickier in the case of master datasets, though). If the pointers are broken and the SearchField is also broken, then a combination of "amputation" and "transplanting" may be necessary.

Typically, diagnostics will identify faulty entries by entry numbers and/or by the values of the offending search fields. In either case, we must eventually define some search field value that will guide us in trying to find the ChainHead. The fact that we find (or fail to find) the ChainHead does not really tell us much if we have not *exhaustively* tested its master dataset beforehand. For instance, even when we find the ChainHead, the involved synonym chain may have been broken in the past and there might be other "floating" duplicate master ChainHeads with the same search field value. Not finding the ChainHead is an even surer symptom of a faulty master dataset. The conclusion is that a complete certification of the involved master datasets is a pre-requisite for doing any kind of chain therapy.

When starting with an entry identified by its entry number, we should not assume that the "key" value of the entry at that location is valid.

When starting with a search field value, we must remember that decimal data formats (IMAGE types "Z" and "P") present a special challenge, since non-negative values may be *signed* or *unsigned*. We must try both, treating them as *separate* requests.

After any fixing, a serial scan of the involved datasets is necessary, to pick up any pieces that may have been left scattered around. This applies to entries as well as to the global dataset HighWater mark, FreeEntry count and detail FreeEntry list.

The post-fixing dataset certification should make sure that there are neither duplicate master ChainHeads nor orphan detail entries.

For checkup (auditing) and treatment (fixing), the technique of zooming is very convenient, since we can select the level of involvement (and the amount of time involved in the process): All or some of the paths in a dataset? All or some of the chains in a path? Consistency between the global counts and pointers of a dataset and the actual entries in the dataset (detail FreeEntry lists, HighWater marks, FreeEntry counts)? Consistency between the root-file tables and the actual datasets as they exist on disc?

A special case of zooming is "windowing" (or "paging", from an MPE viewpoint). IMAGE entries are grouped into MPE blocks. Sometimes, entire MPE blocks are shifted (typically by one byte) as a consequence of hardware or software failures. At other times, entire MPE blocks are over-written by the spooler or other software. If we just look at IMAGE entries, we will get some bizarre diagnostics. If we look at them from the perspective of MPE blocks, we can usually see a more reasonable pattern. Under extreme cases of clobbered blocks, the only reasonable thing to do is to "amputate" (zero out) the offending blocks, perhaps "transplanting" information substitutes from some backup medium and "bridging" the pointers to rebuild the involved chains.

Beware of simplistic solutions

I saw a case where, due to an unlucky coincidence (involving two master chainheads with different search field values pointing to the same detail entry), two chains got hopelessly intermingled. In this case, the search field values were correct but the pointers were incorrect. One (incorrect) way of "fixing" the problem relied on the simplistic approach of changing the "offending" search field values (which were just fine) to match a given chainhead's value. The correct way is to disentangle the pointers to reconnect all the appropriate chain members.

This whole problem could have been eliminated by plugging one of the few holes in IMAGE (the lack of checking for strict correspondence between the search field values for all members of a given chain). If you are serious about improving the quality of IMAGE, please write a note to your Hewlett-Packard software support engineer stating this enhancement request.

Subtleties

The most powerful methods can also be the most dangerous, in the wrong hands. Strong medicines (such as privileged DEBUG or DISKEDIT) need prescription and administration, as opposed to "over the counter" solutions.

Adager's Database-Therapy Functions

With Adager's expert guidance, you can conveniently examine and repair all kinds of database bits and bytes without cumbersome octal or hexadecimal displays.

Why do you need database therapy?

You need database therapy because your database has been, somehow, damaged. It is great to fix the symptoms, but you *must* still address the causes. Please implement policies and procedures that will avoid an encore.

If you need database therapy, something has happened that needs your attention. Here is a list of things that you may want to investigate:

- *partial* database stores,
- *incomplete* database stores,
- *partial* database restores,
- *incomplete* database restores,
- privileged programs,
- power failures or brownouts,
- system failures,
- hardware failures,
- new hardware,
- new software.

Is database therapy guaranteed?

No. The possibilities for database errors range from "very simple to fix" (such as one bad bit or a couple of broken chain links) to "practically impossible to fix" (such as the results of an incorrectly-handled partial database store/restore).

It is (mathematically) simple to *detect* inconsistencies between redundant IMAGE elements. It may be (practically) difficult and tedious to *fix* the inconsistencies.

In the best case, you will repair your database problems in a matter of minutes.

In the worst case, you may have to restore an earlier version of your database and roll-forward its transactions.

Examine Chain

Adager examines a specific chain (detail or synonym). Adager reports to you (and logs internally) any inconsistencies it finds. You may subsequently use **Fix Chain** to repair individual broken chains or **Fix Path** to rebuild the path's damaged chains.

While you examine chains, other processes may access the database in compatible read-only modes.

Examine Path

To examine a path is to examine all of its chains.

If you specify a *master* (manual or automatic) dataset, Adager will examine the master dataset's *synonym* chains.

If you specify a *detail* dataset, Adager will examine the detail dataset's paths (which include the detail *ChainLinks* as well as the related master *ChainHeads*).

Adager reports to you (and logs internally) any inconsistencies it finds. You may subsequently use **Fix Chain** to repair individual broken chains or **Fix Path** to rebuild the path's damaged chains.

While you examine paths, other processes may access the database in compatible read-only modes.

Fix Chain

If you request a *master* (manual or automatic) dataset, Adager will deal with specific *synonym* chains.

If you request a *detail* dataset, Adager will deal with specific *path* chains (which include the detail *ChainLinks* as well as the related master *ChainHeads*).

If a detail dataset has more than one path, Adager takes a *very* methodical approach. Instead of attempting to deal with *all* paths simultaneously (which would drive you to distraction), Adager deals with *one* path at a time.

If you specify a search field *value*, Adager will use hashing to locate the "last" and "first" pointers in the master ChainHead.

As an alternative, you can specify an entry *number*. Adager will still hash to locate the master ChainHead but, in addition, Adager will use the given entry's "previous" and "next" pointers to look for its neighbors (if any).

As still another alternative, you can deal with the *privileged* portions of specific entries (in-use bits, counts, pointers and search field values), as well as the global dataset counts and pointers (kept in the file's user label).

You may specify entries on-line, one at a time, or through an Adager-produced log file.

If a path has too many defective chains, you might prefer to use **Fix Path**, which rebuilds the chains in the path from scratch, based on search field values.

Fix Dataset

Fix Dataset deals with *individual* dataset structures whereas **Fix Path** and **Fix Chain** deal with *path relationships* between datasets.

For a master dataset, **Fix Dataset** re-hashes its entries, rebuilding all synonym chains in the process. Adager deletes duplicate entries and *automatic master* entries whose detail ChainHeads are all zeroes. Adager updates the dataset's entry count. (To repair *only* specific

synonym chains, please use **Fix Chain**.)

For a detail dataset, **Fix Dataset** rebuilds the delete chain, if one exists, and updates the dataset's entry count. (To repair broken detail chains, please use **Fix Chain** or **Fix Path**.)

Fix Path

Adager rebuilds the chains on a detail's path according to the values of the detail's search field (and according to the values of the detail's sort field, if the path is sorted).

Before attempting to fix a detail's path, you should apply the **Examine Path** function to the related *master* dataset's synonym chains. If you find any problems in the synonym chains, please fix them *before* continuing (you may use **Fix Dataset** or **Fix Chain**).

Fix Path is meant for *detail paths*. To rebuild all of the synonym chains on a master dataset, please see **Fix Dataset** above.

Replace Dataset

If you have lost the file for a dataset, Adager creates a brand-new *empty* file, with the appropriate structure.

If the dataset is a *master with paths*, you should subsequently do a **Fix Path** on each path (through the related detail datasets). This adds the master entries (with the appropriate search field values) required to "parent" the orphan detail chains, if any.

If the dataset is a *detail with paths*, you should subsequently do a **Fix Path** on each path to initialize the master ChainHeads.

Conclusion: "success" and "failure"

There are two possible outcomes once we finish our database therapy operation: "success" and "failure". We should not become too complacent when we think we were successful, since there are still some tough questions (*how* did we get into the mess?, *why* did it happen at all?, *what* can we do to avoid an encore?) And even when we face an apparent failure, we should keep things in perspective and learn as much as possible in the process of going down fighting for our database's life.

Paper# 3118

INCREASED SYSTEM AVAILABILITY WITH OPTIMAL BACKUPS

Sue Coatney (408)447-5588
Jim Nissen (408)447-5720

Hewlett-Packard
Commercial Systems Division
19447 Pruneridge Ave
Cupertino, CA 95014

1. Introduction

System Backup is a daily chore; whether you do full backups or partial backups. As your systems get larger and support more users, completing your backup becomes even more difficult. You will have to carefully schedule operations in order to maximize availability of your systems and minimize the interruption of user processing. You may have to require your operators to work off-hours just to mount tapes for your backup.

You will ensure that your backup gets done, because you backup your systems for many reasons. These may include archival, disaster recovery, operator error, user error, and even disk space management. While a brute force backup can meet all these requirements, structuring the backup to meet the specific requirements of your shop can significantly increase the availability of your system.

The release of several new products like HP 1300H Digital Data Storage (DDS), HP C1700A Optical Library Systems, and TurboSTORE/XL II gives you many new options. By using these new technologies in an optimal fashion you can greatly improve your backup performance and recovery performance. These solutions will help you meet your changing business requirements.

As your business environment continues to change, the need for your systems to be available longer for business processing is constantly increasing: you may even be moving to, or planning to move to, a 24 hour a day/ 7 days a week/52 weeks a year availability schedule. Planning for this type of environment requires new applications and system utilities to maximize system availability and to reduce or eliminate "planned" downtime.

1.1 Backup requirements

When asked what is required in a "Backup" product, many of you would respond that you need a FAST, UNATTENDED, and ONLINE backup. These requirements are characterized by the following:

- FAST backup, usually by dedicating some system resources to accomplish this.
- UNATTENDED backup, which means the backup can continue automatically without requiring someone to be there to change media. Depending on system storage requirements, this may also be accomplished by taking advantage of newer technologies for media.

- ONLINE backup, which means users and jobs can be actively running while the backup is occurring.

Your shop may require one or more of these. For example, you may require both unattended and online backup. On the other hand, you may only require unattended backup. Addressing these three requirements can be accomplished in a number of ways, also. Today, a number of backup options are available to address one or more of these requirements.

For example, unattended backup can be accomplished by using high density media like DDS or optical disk. You can further extend the "level" of unattendeness by using high density media with software backup products which provide software data compression. The ultimate unattendeness can be provided when a media management facility is coupled with an optical library system. For example, the media management facility within TurboSTORE/XL II will allow you to do multiple backups to your Optical Library System without operator intervention. It will also provide for selective file retrieval without any operator intervention.

Suppose you have 5 GB of storage to backup and it currently requires 36 reels of 1/2" tape and takes 5 hours. If your objective is to get critical your applications up within 2 hours instead of 5 hours, you can combine features/options like high density software data compression with DDS media. If you don't want to be down at all, plus you want your backup to be unattended, you can combine TurboSTORE/XL II with online backup, with fast data compression, and with multiple DDS drives or the Optical Library System.

1.2 Backup trade-offs

Addressing your backup requirements require different trade-offs. The use of features like online backup and software data compression will use system resources.

Online backup for example will add overhead to the CPU resource as well as the disk subsystem. Use of software data compression will use even more CPU resources. You will need to trade-off media usage against CPU usage. For example if your system is lightly loaded during online backup, you may choose the high density data compression. If it is heavily loaded you would probably choose no data compression. In any case you would not choose to do an online backup during peak hours.

You may want to consider online backup even if you don't have a 24 hour requirement. For example using online backup to ease the race to daylight may be the right choice for your shop. Being able to finish the backup while your morning work load ramps up may be just what your shop needs.

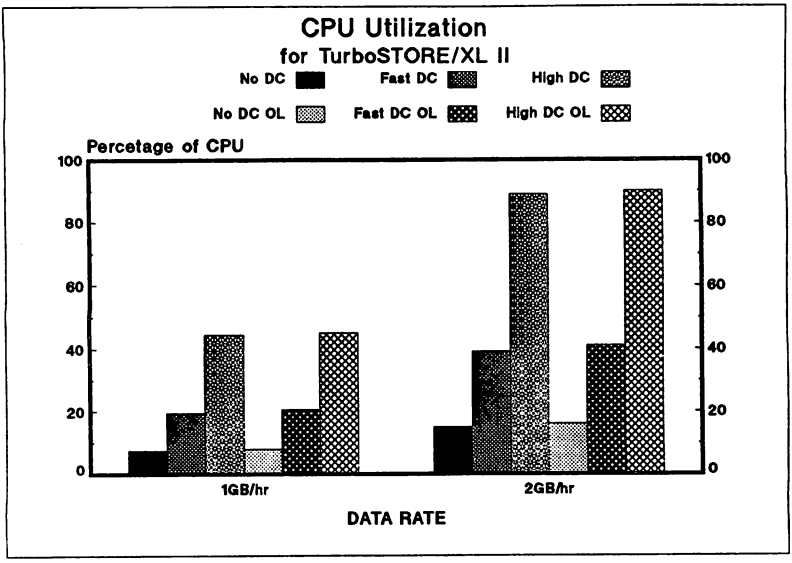
You should also consider unattended backup solutions such as TurboSTORE/XL with DDS or Optical Library Systems, even if you have a 24 hour staff. By reducing the requirement for operator intervention, you also reduce the probability of operator errors.

Developing an optimal backup strategy for your shop will consist of matching your requirements to the available solutions. TurboSTORE/XL II has a wide variety of options which can be used to increase the availability of your system. First let us address online backup and its ramifications.

2. Online Backup

TurboSTORE/XL II provides an online backup option. The use of online backup raises new issues. A question you may ask is why have online backup when the dedicated backup already requires a large amount of the CPU. TurboSTORE/XL II is designed to get the backup done as fast as possible and therefore it will consume as much of system resources as necessary. In fact the ";INTER" option (which allows for interleaving of files) is what typically causes TurboSTORE/XL II to limit the IO rate to the disk subsystem. This is not what you want if your trying to get useful work done during the backup. When data compression is added to TurboSTORE/XL II it is possible for TurboSTORE II to limit on CPU cycles. The only other limits to process throughput is the IO rate to the backup device and the priority of the TurboSTORE/XL II process. Therefore, in the online mode, you must decide on which compression algorithm (if any) and how to use your backup devices (i.e. parallel or sequential) in addition to which devices to be used.

The backup and system performance will depend upon the IO rate of the disk subsystem, speed of the CPU, priority of the backup process, and IO rate of the backup devices. If it takes 2 hours to complete a dedicated backup of your system, it is unlikely that you would want to set up your online backup to complete in that time. You will want to set up the online backup to leave some system resources for the users. A strategy might be to use 50% of system resources for backup and allow the backup to take twice as long. Remember your critical application is up during the backup. Currently there are two mechanisms to throttle or slow down the backup process effectively. The first is running the backup process in a specific queue. The ";INTER" option will override this and put TurboSTORE/XL II at top of the C queue. The second is the IO rate which the backup device will accept data. If the backup device is a streaming tape drive you will want it to limit the process throughput. Figure 2 shows the effective overhead for given throughput rates on a HP3000/950.



The logging overhead incurred with online backup should also be considered. A shadow log which maintains the before image of each and every file is maintained until that file is committed to the backup media. The amount of CPU overhead that is incurred is dependent on the write activity to that file. Therefore the system throughput would be enhanced by making sure that the most active file sets are backed up first.

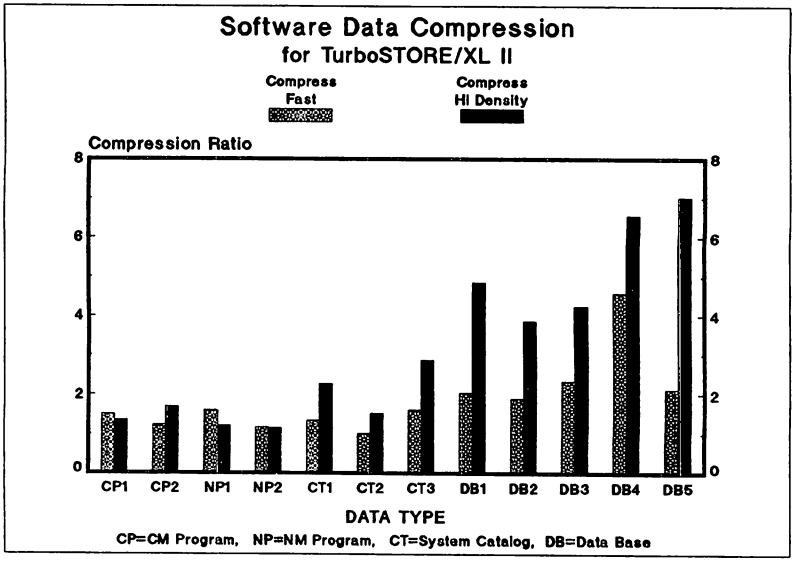
3. Data Compression

TurboSTORE/XL II provides two different software algorithms for data compression. The first is a Run Length Encoding which provides an average compression ratio of 2.1:1 on MPE XL. This algorithm basically removes repeated bytes or characters from the file. It is a simple and relatively fast algorithm. The second is the LZ encoding algorithm. It encodes data into a byte string dictionary. The dictionary resides in the data and compression occurs any time a byte string is repeated. This algorithm provides an average compression ratio of 3.6:1.

The average compression ratios stated above may or may not be realized in your shop. Some data is more redundant (and therefore more compressible) than other data. Figure 1 shows the compression ratios of various file sets. Note that both types of program files have very little redundancy. On the other hand, catalogs are more compressible and data bases are the most compressible. Although not shown here, attempting to compress a non-compressible file requires as much or more CPU than attempting to compress a compressible file.

Therefore, if your backups include a large quantity of program and library files you may not achieve high compression ratios. If the bulk of these files are application and system software, they probably never change between system updates. A simple change in backup procedures from two level to a three level partial strategy could greatly increase the efficiency of your weekly backups. For example whenever you update your system or add a major new

subsystem or application package do a full backup. Then modify your job streams for weekly backups to do partial backups from that date with the ":DATE >=" option in TurboSTORE/XL. If your daily backups are relative to the weekly then you will be able to find any file on one of three tape sets.



The key here is that if you separate executing code, static data, and dynamic data from each other you can maximize the efficiency of the data compression algorithms.

4. Data Bases

The 900 series HP3000s support two HP data bases, TurboImage and Allbase/SQL as well as several third party data bases. Currently DBSTORE is recommended to be used to backup TurboImage data bases. This will ensure the integrity of the data base for roll-back/roll-forward recovery. The STORE or STOREONLINE commands within SQLUtil are recommended for backup of Allbase/SQL data bases. The STOREONLINE command coupled with TurboSTORE/XL II with Online Backup will provide for no interruption of the Allbase/SQL applications.

5. Volume Sets

Another way to increase availability is to separate your applications onto separate Volume Sets. By doing this you can backup one application (using the ONVS option in TurboSTORE/XL II) while all others are still available. There is an additional advantage to partitioning your data onto separate volume sets. If in the unlikely event you have a head crash on one of your disks, you will only have to reload one volume set and not an entire system.

6. Archival

By using the "DATE<=" option with the "PURGE" option you can build a rudimentary Archival system. Using this option of store you can archive files which have not been accessed recently. For example you could archive all files which have not been accessed in the last week or month or whatever. Appendix 1 contains a command file and EDITOR use file which creates an online directory of all files which have been archived. This is done by using the filelist which is generated with the SHOW option.

Appendix 2 contains a command file that will restore a previously archived file. It will find the most recently archived version of the file. By using the ask option the user can find earlier versions of the file. A similar command file could be written to do an archive list function which would aid in finding archived files.

7. Device Organization

With TurboSTORE/XL II it is feasible to organize both your backup and Restore process to use multiple devices. Using multiple devices will decrease the amount of time your backup (and restore) process will take. There are several ways to organize your devices - sequential, parallel, and parallel device pools.

7.1 Sequential Devices

One method of using multiple devices is to copy files sequentially to a group of identical backup devices. As soon as the media on the first backup device is filled, TurboSTORE will immediately start writing to the second backup device. No time is lost while swapping media on the backup devices.

Using sequential devices is particularly useful when using tape devices, which require operator intervention and have long rewind times.

7.2 Parallel devices

Using parallel devices results in using multiple devices concurrently. Using parallel devices will use more system resources than one or sequential devices, but it will shorten the backup time.

If your using TurboSTORE/XL II with an optical library system, using parallel devices is the most efficient method of backing up your system.

7.3 Parallel Device Pools

Parallel device pools provide the advantages of both parallel and sequential devices. Basically, your copying multiple file sets concurrently and have additional backup devices ready when the media on the first set of devices is filled.

8. Backup Device Support

TurboSTORE/XL II supports a variety of backup devices. They include 1/2 inch tape, DDS tape drives, and Optical Library Systems. Each of the device groups have specific characteristics - certain ones will meet your backup needs better than others. Devices which are supported (as of Release 3.0) are:

- HP7974 1/2 Inch Tape Drive
- HP7978 A/B 1/2 Inch Tape Drive
- HP7979 1/2 Inch Tape Drive
- HP7980 1/2 Inch Tape Drive
- HP7980XC 1/2 Inch Tape Drive
- HP1300H DDS Tape Drive
- HPC1700A Optical Library System

Each of these devices have varying characteristics and capacities. We will now discuss these different device characteristics.

8.1 1/2 Inch Tape

8.1.1 Capacity

Half-inch tape devices have several different densities available (1600, 6250, and 6250 with Data Compression). The density used will determine how many bits of data are distributed on an inch of magnetic tape. Assuming a 2400 foot reel of magnetic tape, a tape stored at a density of 1600 bpi will contain approximately 42 MB of data. A tape stored using a density of 6250 bpi will contain 140 MB of data. The hardware compression on the HP7980XC drive will result with a 3-5X compression ration. Therefore, a tape stored using hardware data compression could result in 420 to 700 MB of data being stored per tape.
NOTE: Compression ratios vary depending on the data being compressed. Depending upon the data it is feasible to have less than a 3 times data compression, or greater than 5 times using XC hardware data compression.

As mentioned earlier, it is feasible to use the data compression which TurboSTORE/XL II provides on the 1/2 inch tape devices which have data compression support themselves. Testing has not shown any significant increase of capacity when both hardware & software data compression are used. Overall, if you have a 7980XC tape drive you will achieve better compression and CPU performance using the hardware compression it provides.

8.1.2 Transfer Rates

The effective TurboSTORE/XL transfer rate for 1/2 inch tape devices is very dependent upon which device you are using as well as the number of devices connected to each HP-IB Device Adapter card.

Device transfer rates are as follows:

1/2 Inch Tape Transfer Rates

Device Name	Transfer Rate
HP7974	149 KB/sec
HP7978A/B	395 KB/sec
HP7979	186 KB/sec
HP7980	695 KB/sec
HP7980XC	790 KB/sec

Since the HP-IB card has a limited transfer rate of 1 MB/sec, transfer rates will be limited if too many devices are attached to a single HPIB Device Adapter Card. The combined transfer rate of all devices connected to a single HPIB Device Adapter card is limited to the transfer rate of the card, 1 MB/sec.

8.1.3 Operator Intervention Needed

A significant amount of operator intervention is needed for every reel change. After TurboSTORE/XL has completed writing a tape, an operator needs to swap tapes before TurboSTORE/XL can continue. If an operator is available to service the tape drive, in a best case scenario it will take approximately 3.5 minutes before TurboSTORE/XL can continue with the backup.

To reduce the amount of time your system sits idle waiting for operator intervention, multiple tape drives can be used with TurboSTORE/XL. TurboSTORE/XL offers sequential (when one drive completes, the next drive starts) and parallel (uses several devices concurrently). Creating parallel pools of sequential devices will provide the greatest benefit by maximizing backup data throughput and minimizing the impact of operator intervention.

8.2 DDS Tape Devices

8.2.1 Capacity

The DDS tape devices have only one density (or way of writing data to the tape). A 60m tape can contain up to 1.2 GB of uncompressed data. With the data compression functionality available in TurboSTORE/XL II, up to 5 GB of compressed data can be placed onto one tape

cartridge.

The DDS tape cartridges are also much smaller than the 1/2" reels of tape. The DDS cartridges are approximately the size of a credit card, with a depth of 1/2".

8.2.2 Data Transfer Rates

The DDS device transfer rate is 175 KB/sec. The DDS device is capable of writing 1.2 GB in 2 hours. Using TurboSTORE/XL II high compression, up to 5 GB of data can be written to the same physical tape in approximately 2 hours.

The data compression provided with TurboSTORE/XL II and the large data capacity of DDS tapes results in significantly less operator intervention needed during your backup process.

8.3 Optical Library Systems

8.3.1 Capacity

The data capacity of an entire optical disk which is used on MPE XL is currently a little less than 600 MB (300 MB/side)of uncompressed data. The Optical Library System contains 32 storage slots. Each of the 32 storage slots can hold an optical disk. A fully loaded Optical Library System can contain approximately 19 GB of uncompressed backup data.

The TurboSTORE/XL II data compression capabilities can also be used with the Optical Library System. Using the TurboSTORE/XL II high data compression, each piece of media can contain approximately 2.1 GB of backup data. Therefore, a fully loaded Optical Library System, utilizing high TurboSTORE/XL II data compression can contain up to 70 GB of data.

8.3.2 Transfer Rates

Utilizing both of the imbedded disk drives in the Optical Library System results in a write transfer rate of 500 KB/sec of uncompressed data. With TurboSTORE/XL II's low data compression, you will be able to store 7.5 GB per hour to a single Optical Library System. Since reading data from an optical library system is twice as fast as writing, Restore performance will be significantly better than Store.

The data transfer rates when using the TurboSTORE/XL II data compression are very dependent upon CPU availability and speed.

8.3.3 Operator Intervention

The Optical Library System also contains an autochanger mechanism, which is controlled by the TurboSTORE/XL II with Optical Support software. When TurboSTORE/XL II needs the next piece of media to continue writing or reading the backup, no operator intervention is needed to swap the media. As long as enough Scratch media is kept inside of the Optical Library System, no operator intervention will be needed to complete the backup. The converse is also true, if all the necessary disks are placed inside the Optical Disk Library System, operator intervention won't be needed when restoring files onto your MPE XL system.

9. Access Time for Restore

Access time for Restoring files is dependent upon several factors:

- Type of backup device (1/2 inch tape, DDS, or Optical).
- Number of files being restored.
- Where your files are located on your backup media.
- CPU availability (potentially).

The length of time needed to restore a large amount of files is very dependent upon the transfer rate of your backup device. When restoring a small number of files, other factors influence the restore speed. Those factors will be discussed in further detail.

9.1 Selective File Restore with Tape

Restoring a specific file or set of files from tape (either 1/2 inch or DDS) involves the following:

- Issue the Restore command.
- Mounting the last volume of a multi-volume set.
- Mounting the volume which TurboSTORE/XL directs you to.
- TurboSTORE/XL skips over other files on tape to the requested file.
- The file(s) will be restored from the tape volume(s).

Once the correct tape volume is mounted the amount of time spent locating the files is relatively short. TurboSTORE/XL II uses "File Marks" on the 1/2 inch tape drives and "Save Set Marks" on the DDS devices to skip to the file requested. Using these technologies, TurboSTORE/XL II is able to locate your file within 1-2 minutes. Once the file is found, the actual data transfer happens relatively quickly - although it does depend upon file size and device transfer rates.

9.2 Selective File Restore with Optical Library Systems

Restoring a specific file or set of files from an Optical Library System involves the following actions:

- Issue the Restore command with the appropriate backup name specified.
- TurboSTORE II will mount optical disk volumes which match the backup name specified in the Restore command, looking for the requested file(s).
- If the necessary media is not within the Optical Library System, the operator will be prompted on the console to place the correct media in the Optical Library System mailslot.

If the necessary backup media is within the Optical Library System, files will be restored within a matter of minutes. The actual time necessary depends on how many optical disks need to be mounted before the file(s) are found. It takes approximately 1 minute to get each disk mounted and into the correct state before files can be located.

10. Media Handling

Backups are done so that in case a disaster occurs, the data on your system can be retrieved. As a result it is important to keep media labelled with backup information (so you know what media to retrieve data from) and to use good quality media (so the data can be retrieved). Proper media handling and care will make your backup process smoother.

10.1 1/2 Inch Tape

- To prevent data loss, 1/2 inch tapes need to be rewound and reconditioned every 2-3 years.
- The heads on the tape drives need to be cleaned every 10 hours of use.
- Media and tape devices should not be exposed to temperature or humidity extremes.
- Good quality tapes should be used to prevent time-wasting retries.
- 1/2 tapes are large and require a large amount of floor space for storage. Tapes must be stored vertically.

10.2 DDS Tapes

- DDS cleaning cartridges need to be used after every 25 hours of use.
- Media and DDS devices should not be exposed to temperature or humidity extremes.
- DDS tapes are small & require a significantly smaller amount of floor space for storage.

10.3 Optical Library Systems

- Rewritable optical media is good for 10+ years.
- No media or optical head cleaning is needed.
- To eliminate operator intervention during backups enough SCRATCH media to complete your backup should be available in the Optical Library System.
- Media from previous backups can be left inside the Optical Library System so Restores can be performed without operator intervention.
- Media removed from the Optical Library System should be labelled with the backup name/date and returned to it's plastic case.
- Neither optical media or the Optical Library System should not be exposed to temperature or humidity extremes.

11. Conclusion

In planning your backup strategy you must first determine your requirements and then organize your procedures for optimal performance. Answering the following questions will help define your requirements:

- Do you need to keep your system up 24 hours a day or would online backup help in your race for daylight?
- How does online backup fit into your needs?
- Can moving to an unattended backup solution optimize your operator's productivity, or will it help alleviate operator errors?
- Does having unattended or online backup capability reduce your operational costs?
- How does your backup strategy help you provide better service to your users?
- Is storage space for media important to you?
- How important is media life and/or media cost?
- How important is unattended archival retrieval?
- What is the projected growth of your system?

When you understand the answers to these questions you will be able to select the peripherals and backup product(s) to meet your requirements. Once you have chosen your backup solution, then you can optimize your backup process.

Several approaches have been suggested to optimize your backups. The basic concept is to "Divide and Conquer." By separating your backups you can get optimal performance from the available backup products. You may be concerned that you have more backups to manage and this is true. However with tape management systems or the HPC1700A Optical Library System and the use of command files and/or job streams this effort can be easily managed.

Appendix 1

```
COMMAND FILE "STOREF"
ANYPARM instring=null
setvar comstring ups("!instring")
comment
comment   Remove any user specified show option and add option
comment   required for utility
comment
setvar comstring comstring-";SHOW"-",SHORT"-",LONG"-",DATES"
setvar comstring comstring-",SECURITY"-",OFFLINE"
setvar comstring comstring-"=SHORT"-"=LONG"-"=DATES"
setvar comstring comstring-"=SECURITY"-"=OFFLINE"+";SHOW=LONG"
continue
purge archdir1
continue
purge archdir2
comment
comment  Read last store number
comment
setvar cierror 0
continue
input asciinum<archdir.pub.sys
if cierror<> 0 then
  build archdir.pub.sys;disc=10;rec=60,16,f,ascii;nocctl
  file archdir=archdir.pub.sys,old;rec=60,16,f,ascii;nocctl
  echo RF000000>>*archdir
  echo NO DATE>>*archdir
  echo ########################################>>*archdir
  setvar asciinum "RF000000"
setvar cierror 0
endif
setvar asciinum str("!asciinum",3,6)
setvar num !asciinum
setvar num num+1000001
setvar asciinum "RF"+str("!num",2,6)
echo !asciinum>archdir2
comment
comment   Execute TurboSTORE
comment
continue
reset syslist
STORE !comstring>archdir1
save archdir1
save archdir2
comment
comment  Have EDITOR remove exteraneous informantion
comment  join it to the archive directory
comment
continue
purge archdir3
run editor.pub.sys <archedit.pub.sys
copy archdir3,archdir.pub.sys;yes
```

```
purge archdir3
reset syslist
reset archdir2
echo purge archdir1
echo purge archdir2
deletevar comstring
deletevar asciinum
deletevar num

EDITOR FILE  "ARCHEDIT.PUB.SYS"

SET TIME=3000
T ARCHDIR2
JQ ARCHDIR1,UNN
DQ 2,3,5
CQ 85/255 TO "*****" IN ALL
FQ FIRST
WHILE
  FQ "THE BACKUP TO DASS NAME IS"
  BEGIN
  CQ "THE BACKUP TO DASS NAME IS " TO "NAME="
  COPY * to 5
  END
FQ FIRST
WHILE
  FQ "FILENAME GROUP    ACCOUNT"
  BEGIN
  FQ *-1
  DQ 6/*
L *+1
  END
FQ FIRST
FQ "FILENAME GROUP    ACCOUNT"
CQ 28/74 TO "" IN */LAST
GATHER ALL
FQ 8
FQ "    ****"
DQ */LAST
CQ "*****" TO "" IN ALL
CQ 1/40 TO "****************************************" IN LAST
JQ ARCHDIR.PUB.SYS,UNN
SET FORMAT=DEFAULT
SET RIGHT=40
SET LENGTH=40
SET FIXED
K ARCHDIR3,UNN
EXIT
```

Appendix 2

```
COMMAND FILE "RESTOREF"

ANYPARM instring=NULL
SETVAR comstring UPS("!instring"-"        "-"    "-"  "-" "+";NO ")
SETVAR comstring comstring-"        "-"    "-"  "-" "
SETVAR comstring comstring-"        "-"    "-"  "-" "
SETVAR comstring comstring-"        "-"    "-"  "-" "
SETVAR comstring comstring-"        "-"    "-"  "-" "
SETVAR comstring comstring-"        "-"    "-"  "-" "
SETVAR comstring comstring-"        "-"    "-"  "-" "
SETVAR comstring comstring-"        "-"    "-"  "-" "
SETVAR comstring comstring-"        "-"    "-"  "-" "
SETVAR comstring comstring-"        "-"    "-"  "-" "
SETVAR comstring comstring-"@"-"@"-"@"-"@"-"@"-"@"-"@"-"@"
SETVAR pos POS(";",comstring)
SETVAR answer STR(comstring,pos+1,3)
SETVAR comstring LFT(comstring,pos-1)+"...
SETVAR pos POS(".",comstring)
SETVAR fname LFT(comstring,pos-1)
SETVAR comstring STR(comstring,pos+1,20)
SETVAR pos POS(".",comstring)
SETVAR gname LFT(comstring,pos-1)
SETVAR comstring STR(comstring,pos+1,20)
SETVAR pos POS(".",comstring)
SETVAR aname LFT(comstring,pos-1)
IF gname="" THEN
   SETVAR gname hpgroup
ENDIF
IF aname="" THEN
   SETVAR aname hpaccount
ENDIF
SETVAR fname LFT(fname+"        ",8)
SETVAR gname LFT(gname+"        ",8)
SETVAR aname LFT(aname+"        ",8)
SETVAR fname fname+"."+gname+"."+aname
FILE INFILE=$STDIN
ARCHFIND  <archdir.pub.sys
CONTINUE
DELETEVAR comstring
DELETEVAR pos
DELETEVAR answer
DELETEVAR fname
DELETEVAR gname
DELETEVAR aname

COMMAND FILE "ARCHFIND.PUB.SYS"

PARM a=a
```

```
SETVAR instring " "
WHILE lft(instring,3)<>"###"
  INPUT instring
  SETVAR instring "!instring"
  SETVAR tempstr lft("!instring",2)
  IF tempstr="RF" THEN
     SETVAR rfile lft("!instring",8)
     INPUT instring
  SETVAR instring "!instring"
     SETVAR date instring
     INPUT instring
  SETVAR instring "!instring"
     SETVAR bname ""
     SETVAR tempstr lft(instring,5)
     IF tempstr="NAME=" THEN
        SETVAR bname lft(instring,31)
     ENDIF
  ENDIF
  SETVAR tempstr lft(instring,26)
  IF lft(tempstr,3)<>"###" THEN
     WHILE lft(tempstr,3)<>"***"
       SETVAR answer2 "Y"
       IF fname=tempstr THEN
          SETVAR tempstr tempstr-"        "-"    "-"  "-" "
          SETVAR tempstr tempstr-"        "-"    "-"  "-" "
          SETVAR tempstr tempstr-"        "-"    "-"  "-" "
          IF answer="ASK" THEN
             SETVAR ansr "k"
             WHILE ansr<>"Y" and ansr<>"N"
                ECHO Do you want !tempstr  from !date?
                INPUT ansr ;prompt="(YES/NO)? "<*INFILE
                SETVAR ansr ups(lft("!ansr"-"    "-"   "-" ",1))
                SETVAR answer2 ansr
             ENDWHILE
          ENDIF
          IF answer2="Y"  THEN
             IF LFT(bname,5)="NAME=" THEN
                RESTORE ;!tempstr;MOSET=(Mo);!bname;SHOW
                SETVAR instring "######"
             ELSE
                SETVAR media str(instring,30,10)
                SETVAR set str(instring,27,2)
                CONTINUE
                PURGE !rfile
                BUILD !rfile;disc=10;rec=36,16,f,ascii;nocctl
                FILE temp=!rfile,old;rec=36,16,f,ascii;nocctl
                ECHO Mount backup from !date >>*temp
                ECHO Media !media  of Store Set !set>>*temp
                TELLOP Mount backup from !date
                TELLOP Media !media  of Store Set !set
                FILE !rfile;dev=tape
                CONTINUE
                RESTORE *!rfile;!tempstr;SHOW
                DELETEVAR media
```

```
            DELETEVAR set
            PURGE !rfile
            SETVAR instring "######"
          ENDIF
        ENDIF
      ENDIF
      INPUT tempstr
      SETVAR tempstr lft("!tempstr",26)
    ENDWHILE
  ELSE
    ECHO File !fname not found.
  ENDIF
ENDWHILE
DELETEVAR tempstr
DELETEVAR rfile
DELETEVAR date
DELETEVAR bname
DELETEVAR answer2
```

High Availability on the HP 3000

Jessy Hsu and Kendall Sutton
Commercial System Division
Hewlett-Packard Company
19447 Pruneridge Ave.
Cupertino, CA 95014

Abstract

In organizations that rely on online transaction processing (OLTP), whether it's business, manufacturing, or government, there is an increasing need for greater system up time. While the HP 3000 has a reputation for superior reliability, today's OLTP environments demand high availability features that reduce system downtime to an absolute minimum. HP has responded to this need with several high availability products. This paper will discuss the use of Mirrored Disk/XL, TurboStore II/XL (with online backup), SPU Switchover/XL (which allows data sets to be switched from one SPU to another), and how they fit into a high availability data center configuration. We will also discuss how these products form the building blocks for further advances in the area of high availability on the HP 3000.

Table of Contents

Introduction

For organizations that increasingly rely on online transaction processing (OLTP) -- whether it be business, industry, or government -- the need for greater system uptime goes in only one direction: up. In focusing on making the HP 3000 a leader in the OLTP market, HP has recognized and addressed this need.

To ensure a leadership position in the OLTP market, HP has produced several products to increase system availability. The goal is continuous availability -- eliminating the effects of unplanned and planned system downtime from being apparent to your end users. The strategy is to provide the appropriate system availability based on evolving market needs.

1. System Outage Characteristics

Downtime for a customer is the time when the system is not available to the end user. Hence, the system can be defined to be available if an end user can carry out a unit of work (e.g. transaction) against the data (e.g. database). The causes of customer outages or downtime can typically be classified as either unplanned or planned.

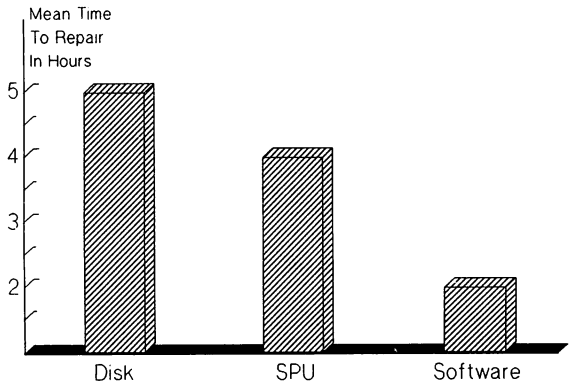
1.1 Unplanned outages

Unplanned downtime is the time when the system is not available due to software and/or hardware failures. The objective of highly available systems is to minimize this downtime, thereby maximizing a customer's return on their computer system investment.

A 1989 INTEREX survey (Figure 1) shows that the major components that contribute to unplanned system downtime are disk failure, SPU failure and software failure. It reflects components which cause extended unplanned downtime and demonstrates that even infrequent failures can lead to large amounts of downtime. Disk failures represent the area of greatest extended outages. Much of the sustained downtime is due to data recovery, which occurs upon replacement of the failed disk element.

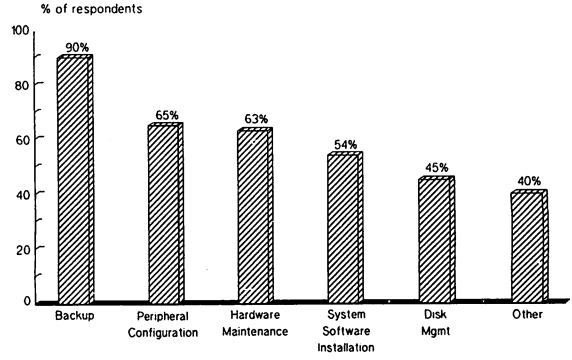
Today's HP 3000 PA-RISC systems and disk subsystem has very high MTBF (Mean Time Between Failures) and system outages caused by SPU or disk failure are infrequent relative to competitive systems. However, it is HP's objective to reduce the system outage impact of those infrequent failures even further. With high availability solutions available since 1990, outage time can be greatly reduced. This will be discussed in the 1990 Availability Plan section.

Software failures represent another major factor causing system outages. HP recognizes the problem and currently deploys quality methodologies to improve software reliability. Minimizing the impact of software failures will center on reducing the number of system failures and the time to recover.



Source: Interex Survey 1988

Figure 1. Causes of extended downtime



Source: Interex Survey 1989 (185 respondents)

Figure 2. Sources of planned system downtime

1.2 Planned downtime

Planned downtime is a result of system activities like backup, configuration management, etc. Figure 2 suggests that backup and configuration are the number one and number two critical area of planned downtime.

As we will discuss in upcoming sections, the focus of high availability offerings is to manage or eliminate these factors. As HP's high-end system configurations continue to grow and more disks, more memory, and more users are connected on a system, the management and control of these activities will be crucial to meeting the increasing demands for higher availability. HP recognizes this and our strategy is to seek ways to resolve these factors.

2. Increasing levels of system availability - HP's Strategy

2.1 Pre-1990 availability

With the introduction of PA-RISC (Precision Architecture--Reduced Instruction Set Computing), HP redefined quality commercial systems. Based on early warranty data, HP 3000 Series 950s achieved well over 55% increases in reliability over the industry-acknowledged highly reliable Series 70. PA-RISC has been a significant factor in a leading industry watcher, Datapro, annually rating HP's reliability superior to all others, including IBM and DEC.

The ability to simplify dramatically the number of components over traditional computer architectures is the primary reason for the increase in reliability. Additionally, other high reliability features are designed into PA-RISC based systems. These features range from Predictive Support/XL which can predict maintenance needs before a failure can happen, to HP PowerFail, which provides uninterrupted battery backup after a power failure has occurred.

2.2 1990/1991 high availability offering - CDAC

In 1990/1991, the strategy was to provide an integrated high availability architecture known as CDAC (Continuous Data Access Control). The architecture focused on minimizing the fundamental causes of unplanned and planned outages and maximizing system availability.. The first suite of products to address these needs include products to manage disk (Mirrored Disk/XL), processor (SPU Switchover/XL), and software failures (AutoRestart/XL). As for minimizing the need for scheduled outages, HP 3000 systems offer a variety of continuous operation products while performing various system activities, such as backup (TurboSTORE/XL II) and configuration management (OpenView DTC Manager, DTC/Terminal Access).

2.2.1 Unplanned outages

As can be seen in Figure 1, the major cause of unplanned downtime is hardware failures, primarily disk and SPU. With the increasing complexity of processor units and a larger number of disk drives per system, hardware failures have the potential to significantly increase their contribution to unplanned downtime. In particular, products such as Mirrored Disk/XL and SPU Switchover/XL have been introduced in order to minimize the impact of such hardware failures. Increased system reliability in conjunction with AutoRestart/XL helps to both reduce the frequency of downtime events as well as reduce the recovery time associated with a software failure.

2.2.1.1. Mirrored Disk/XL eliminates downtime due to disk failures

Mirrored Disk/XL eliminates vulnerability to disk drive failures by providing redundant or mirrored disk drives for user-selected critical application data. In the event of the failure of a disk that is mirrored, all I/O activity for the mirrored pair is automatically switched to the alternate disk. Repair and resynchronization of the failed disk are performed online with no interruption in application processing. Further, in using separate I/O interfaces for the mirrored disk partners, all single points of failure are eliminated in the disk subsystem.

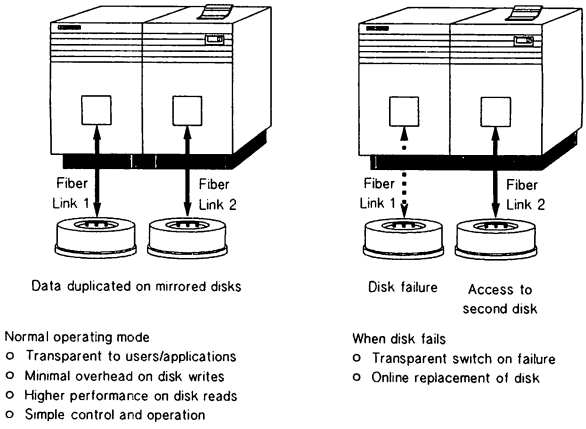
2.2.1.2. SPU Switchover/XL provides recovery due to SPU failure

SPU Switchover/XL automatically detects system failures and allows the system operator to initiate switchover between a pair of SPUs without interruption of the secondary system. Both SPUs can be running applications with the primary SPU typically running critical OLTP applications and the secondary SPU running less critical applications (such as application development). No additional processing overhead is required to support SPU Switchover/XL on either processor. Additionally, full data integrity is ensured as the failed (primary) SPU relinquishes control of the disk and transaction log to the secondary SPU.

SPU Switchover/XL is a new product available on MPE XL release 3.0. A more complete explanation along with an example of its use is presented in the Appendix A.

2.2.1.3 AutoRestart/XL minimizes downtime due to software failures

Normally, significant time can pass before a software system failure is recognized and an operator is able to respond by initiating a dump and manually restarting the system. Software failures can contribute significantly to the amount of extended downtime as shown in Figure 1. AutoRestart/XL minimizes this downtime by automatically and immediately saving the system state and initiating system restart. No operation intervention or action is necessary. Hence, system recovery time is minimized due to a software failure. Further, by saving the system state to disk rather than tape, the time required to save the system state is cut by at least 50% on high-end HP 3000 systems with larger memories.



Normal operating mode
- Transparent to users/applications
- Minimal overhead on disk writes
- Higher performance on disk reads
- Simple control and operation

When disk fails
- Transparent switch on failure
- Online replacement of disk

Figure 3. Mirrored Disk/XL operation

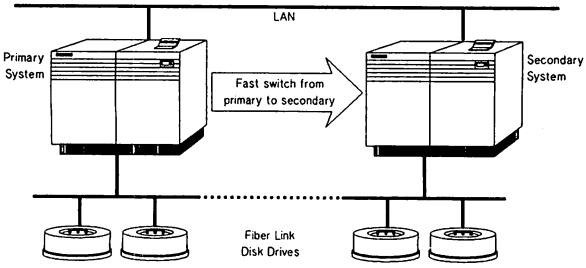


Figure 4. SPU Switchover/XL operation

2.2.1.4 Preventative Maintenance (PM) minimizes outages

In the event that the elimination of unplanned outages is not possible, solutions which can transform this downtime into scheduled maintenance or planned downtime can be crucial to increasing system availability.

Solutions like Predictive Support/XL provide this capability. Predictive Support/XL provides a proactive preventative maintenance (PM) approach to detecting and resolving problems within various subsystems, i.e. memory, tape, disk drives. Predictive Support/XL is a standard feature of MPE/XL's software support and can provide automatic reporting into HP's support centers. Hence, PM minimizes the chances of various system component outages.

2.2.2 Planned downtime

As demonstrated in Figure 2, managing and reducing planned downtime covers a broad area. To eliminate planned downtime, multiple solutions are necessary.

2.2.2.1 TurboSTORE/XL II virtually eliminates backup downtime

HP's approach in 1990/1991 has been to focus on areas that provide the greatest impact on system availability. TurboSTORE/XL II was developed to address online backup, an area identified in Figure 2 as a chief reason for planned downtime.

TurboSTORE/XL II can eliminate downtime due to backup requirements. With the online backup feature of TurboSTORE/XL II, around-the-clock data processing is supported while system backups occur. Though normally a 5 minute application interruption occurs to ensure complete data integrity, with ALLBASE/SQL, this interruption is eliminated. Further, the time and need to handle media (i.e. tape) during the backup process can be eliminated. Unattended backup is supported with up to 74 Gbytes of capacity when utilizing HP's Rewritable Optical Disk Library System. The impact on production is minimized. Further, increased performance using multiple backup devices in parallel (files are interleaved on the devices) can dramatically reduce backup time.

2.2.2.2. Mirrored Disk/XL provides online backup

Mirrored Disk/XL allows users to "split" the mirrored pair and backup files while continuing to process applications. Additionally, Mirrored Disk/XL eliminates disk failures as well as minimizing overhead due to system backup.

2.2.2.3 SPU Switchover/XL supports preventative maintenance

Although SPU Switchover/XL is targeted at reducing unplanned downtime, it provides an excellent mechanism to switch applications and data from one system to another for any reason. For example, if a system needs to be down for preventative maintenance (PM) or for deferred repair service, SPU Switchover/XL minimizes the associated application downtime.

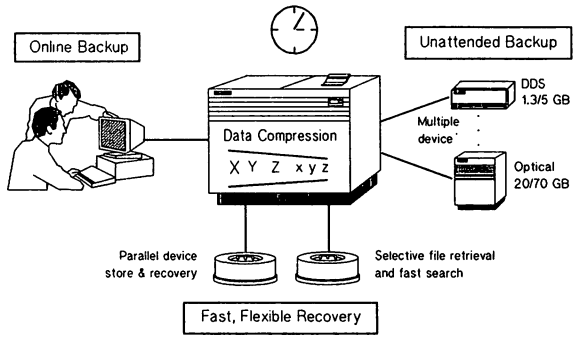


Figure 5. TurboSTORE/XL II operation

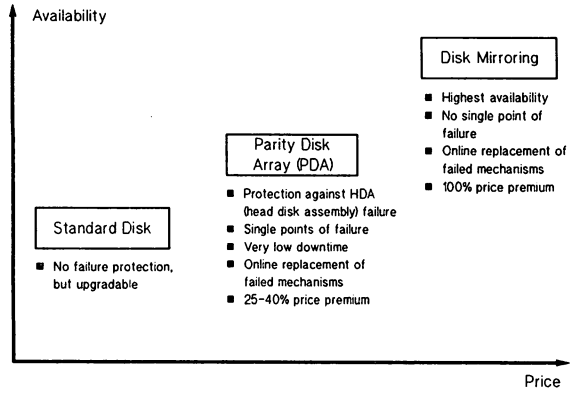


Figure 6. Disk storage product family

2.2.2.4 Dynamic reconfiguration eliminates significant downtime

Solutions like OpenView DTC Manager simplify configuring, installing, monitoring and diagnosing DTC related issues. Further, solutions like DTC/Terminal Access provide for dynamic reconfiguration (e.g. add/delete) of terminals without system interruption. DTC/Terminal Access also enhances system availability by its ability to dynamically switch from one system to another on the same LAN.

2.2.2.5 Powerpatch/XL minimizes the impact of software fixes

HP Powerpatch/XL is a proactive patch product designed to fix known software problems on customer systems before they experience them. Powerpatch/XL provides the ability to customize the patch application process for only those products (MPE/XL, data communications, etc.) actually installed on a customer's system. The patches supplied with Powerpatch/XL have been field tested for a minimum of 30 days.

Further, Powerpatch/XL minimizes planned downtime. The background installation of patches can continue as users continue to use the system. Only when final implementation of the patches occur is the system briefly impacted.

2.3 Future System Availability Technologies - SCA

For future system availability HP will focus on providing incremental improvements in high availability. We will also emphasize the development of more cost-effective high availability solutions, as indicated by industry trends, for 1992 and beyond through a concept called System Component Availability (SCA).

2.3.1 System Component Availability (SCA) Concept

The technology available today has provided for a multiple box solution to reduce unplanned outages. For example, SPU Switchover/XL requires two system processing units (SPUs) to provide redundancy in processors. The same is true for disk mirroring. The cost for redundancy is justified for ensuring high system availability.

Through System Component Availability (SCA) this cost premium can be lowered while providing somewhat greater availability than a standard system component, e.g. SPU, disk. SCA creates a family of high availability solutions. In many cases "filling" in between the standard and the totally redundant solutions, Figure 6 represents an example of this with disk. The SCA concept involves designing and building high availability into the actual components (e.g. SPU, disk).

For example, disk availability solutions will involve not only disk mirroring but also parity disk arrays (PDA) -- disk SCA. More details about PDA are described in the next section. Though providing greatly increased data availability, PDAs do not provide the redundancy in disk controllers, power supplies or interface adapters that mirrored disks allow. This positions PDAs as an availability solution between standard and mirrored disk, as shown in Figure 6. The objective is greater cost-effectiveness with a lower price premium.

In minimizing planned downtime, such as backup, HP may focus on process automation, flexibility in leveraging backup devices for multiple systems, and performance improvements. While increased functionality will occur, the primary focus will be on improving cost-effectiveness. Other areas of improved planned downtime management, e.g. configuration management, will follow in due course.

2.3.2 Parity disk arrays (PDAs) increase cost-effectiveness

Currently, the disk mirroring product for MPE/XL systems allows mirroring of data in user volume sets. The product provides protection against failure of disk mechanisms (HDA--head disk assembly), disk controllers and HP-FL interface adapters. Hence, disk mirroring provides complete redundancy of the disk subsystem.

An upcoming extension to full disk mirroring is the concept of parity disk arrays (PDAs). In concept, PDAs might involve five separate disk mechanisms connected to a single controller and single HP-FL interface adapter. In a parity configuration, four disk mechanisms serve to store data while the optional fifth mechanism stores the parity data of the other four. In the event of a HDA failure, all data is protected through the use of the parity data. PDAs allow for rapid and transparent reconstruction of any lost data, as well as easy, online servicing of the failed HDA mechanism.

The system component availability (SCA) feature of PDAs, provide a cost reduction over disk mirroring. The result is a more cost-effective approach in providing protection against HDA failures. Note, however, that a PDA architecture does not provide complete protection against disk failures as does disk mirroring. For instance, the PDA concept does not provide for redundancy in disk controllers, disk power supplies or HP-FL interface adapters. As such, PDAs actually "fill a hole" in providing a continuing array of solutions aimed at reducing outages due to disk failures.

2.3.3 Continuing towards software resilience

For 1992 and beyond, HP will focus on reducing system failures and ways of making the system more resilient to software and hardware failures. Some current system failures could be isolated to a single process which could be aborted instead of bringing the system down. HP will also emphasize on fault isolation strategy, such as the use of "footprinting" in the kernel in order to isolate and recover from error conditions without causing a system abort.

2.4 Summary of HP 3000's availability strategy

The strategy to achieve higher system availability summarized above is based on three fundamental elements:

- Provide exceptionally reliable system components
- Provide a product family of integrated continuous availability solutions
- Increase the cost-effectiveness of providing continuous availability

With these elements HP is setting a direction which will position itself as a leader in OLTP.

3. High Availability Data Center Recommendations

The individual high availability product offerings from HP can be combined to provide maximum availability for critical applications and data.

Proper planning is required to create the optimum hardware configuration and data placement strategy. We will examine a setup that uses the following products to achieve maximum data availability:

TurboStore II/XL with On-line Backup
Autorestart/XL
Mirrored Disk/XL
SPU Switchover/XL
Parity Disk Arrays (coming out with MPE XL release 4.0)
Open View DTC Manager

Figure 7 shows a conceptual view of a high availability data center. In this scheme all critical data and applications have been migrated to mirrored volume sets which are configured to be switchable between two SPUs.

Each system is equipped with an additional user volume set so Autorestart/XL can perform a fast dump to disk and automatically reboot the system in the event of a software failure.

The system volume set on each system is made up of parity disk arrays (PDAs). (Available '92) While they do not cover the complete disk subsystem with redundant components, they provide a high level of protection against failures.

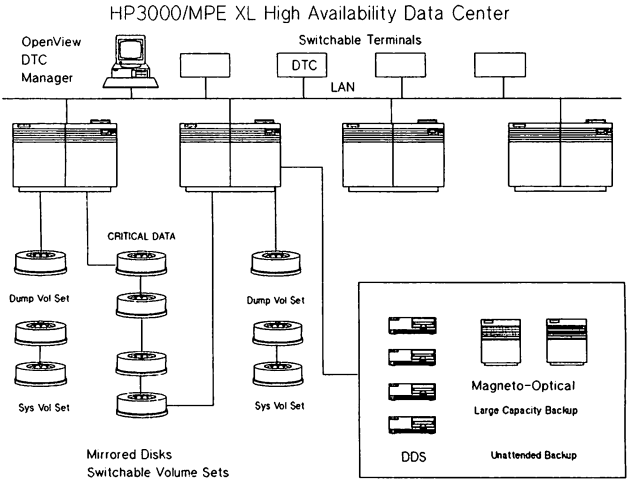
Open View DTC Manager is running on a PC connected to the LAN to provide terminal switchover and assist in network switchover.

For backup, TurboStore II/XL with on-line backup is used. It can be used to backup to a magneto-optical device for large capacity unattended backups. And with Allbase/SQL applications, the on-line backup takes place with zero application downtime. Other application data requires only a short quiescent period before the backup begins, then the applications can continue.

In this high availability configuration, only a limited number of events can cause unexpected downtime, and only very rare events (such as a site disaster or two mirrored disks failing at the same time) can cause an extended period where the data is not available.

Planned downtime is minimized with the use of TurboStore II/XL with on-line backup. Planned downtime is also reduced by scheduling a switchover in the event that a machine requires an extended period of down time for preventive maintenance or hardware/software updates or configuration changes.

Figure 7. HP3000/MPE XL High Availability Data Center



In summary...

HP's commitment to leadership in the OLTP market has led to a focus on enhancing system availability. HP will implement a phased approach to delivering increasing system availability which allows both the time to develop the appropriate technology as well as the time for customers to adopt.

HP believes that the solutions proposed here will dramatically reduce both scheduled and unscheduled system outages. Our goal is to provide continuous availability in your data processing environment and support the ever growing processing demands of your business. Today, HP 3000's high availability solutions are the foundation for this future.

Appendix A SPU Switchover/XL Product Feature Set and Implementation

Introduction

SPU Switchover/XL is a new product designed to increase data availability on HP 3000 Series 900 computer systems. In the event that a system must be down for a prolonged period of time (e.g. because of a hardware failure or preventive maintenance) mission critical data and applications can be switched to an alternate SPU so that processing can continue.

SPU Switchover/XL has the following features:

- Switchover of volume sets between a pair of systems - One or more user volume sets can be switched back and forth between a pair systems. Once configured, the volume sets can be switched from one system to the other whenever desired. The switchover could be initiated because of extended downtime on one system, or for load balancing.

Note: Switchable volume sets must consist of only HP-FL disks.

- Recovery time - Data recovery time is less than that associated with a reboot of a system.

- No reboot of alternate - The system that volume sets are being switched to can continue normal processing during and after the switchover. However, the performance implications of the additional load will need to be evaluated.

- Simple operator commands - All volume sets in the switchover configuration are switched from one system to the other with a single command.

- Switchback - When the system from which the volume sets were switched is once again operational, the sets can be switched back to their home system. This can be done without impact to processes (on either system) that are not using the data on the switchable sets.

Note: This product is not intended to assist in recovery from software failures. Even though switchover time is less than reboot time, the additional overhead of switching users over to the alternate system, and eventually back to the home system, would probably negate the advantage. This would need to be evaluated on a case by case basis.

Hardware Configuration

Figure 8 shows an overview of the hardware configuration for SPU Switchover/XL. The most notable characteristic is the fact that each group of switchable disk drives has a physical connection to both systems. (A more detailed picture of how this is accomplished is presented below.)

The drives are accessible by only one system at a time. During boot, the drives will mount for access on their home system as specified in the SPU Switchover configuration. On the alternate system, they will appear as volume type LOCKED and will not be available for access.

Figure 9 shows a more detailed view of the SPU Switchover configuration. Switchable volume sets are connected to two systems via the HP Fiber Link interface. There may be up to eight disk drives connected together with P-Bus cables from each HP-FL Device Adapter card. The minimum number of switchable disks is two, since one HP-FL DA card from each machine must connect to a disk drive in the chain.

Software Configuration

Using Sysgen

Sysgen is used to configure SPU Switchover. The purpose of the configuration is to define which volume sets may be switched, and the home and alternate systems for those volume sets. The configuration will be used to determine on which system a volume set will mount at boot time.

Commands

asystem - Used to give each system a name. This name is used in subsequent configuration commands.

apair - Used to specify a pair of systems, one of which is a home system (for some volume set(s)) and one of which is an alternate.

avolset - Used to identify a switchable volume set. The parameters of the command specify the home system and alternate systems for that volume set.

show - Used to display the current configuration.

dvolset - Used do delete a volume set from the configuration.

dpair - Used to delete a home/alternate system pair from the configuration.

dsystem - Used to delete the system name previously specified.

Working Example

Let's examine the steps necessary to set up an SPU Switchover/XL configuration on a pair of systems. For the purposes of this example, we will be sharing a two volume mirrored volume set (four disks total) called DATA_VOL_SET between two machines. The machines will be named XX and YY. XX will be the home system for the volume set and YY will be the alternate.

The hardware would be set up as shown in figure 9. Assume that the disks are brand new and have no data on them. Further assume that the normal sysgen configuration files have been established that designate the disks on one of the FL chains to be ldevs 31 and 32, and the disks on the other FL chain to be 41 and 42, on both systems. (Note that switchable disks do NOT need to be the same ldev on both systems for SPU Switchover/XL to function.)

Given this initial hardware and software configuration, we are ready to begin the SPU Switchover/XL software configuration. Before any switchover configuration is present, disks that are connected to two systems follow a simple rule during the bootup process. A disk will mount for access on the

system that "talks" to that disk first. It will mount as "LOCKED" (as displayed by the DSTAT command) on the other system. It will not be available for either access or initialization on the other system.

Therefore, in preparing for switchover configuration, we make sure that the system on which we plan to do the configuration (system XX) is booted before the other system (YY).

On system XX we would see the following from a DSTAT command:

```
 LDEV-TYPE     STATUS       VOLUME (VOLUME SET - GEN)
-----------  ----------   ----------------------------
30- 079371    MASTER-MD     MEMBER1          (DATA_VOL_SET-0)
31- 079371    MEMBER-MD     MEMBER2          (DATA_VOL_SET-0)
40- 079371    MASTER-MD     MEMBER1          (DATA_VOL_SET-0)
41- 079371    MEMBER-MD     MEMBER2          (DATA_VOL_SET-0)
```

This shows that the volume set is mounted for access by the system.

On system YY we would see the following from a DSTAT command:

```
 LDEV-TYPE     STATUS       VOLUME (VOLUME SET - GEN)
-----------  ----------   ----------------------------
30- 079371    LOCKED
31- 079371    LOCKED
40- 079371    LOCKED
41- 079371    LOCKED
```

The status is LOCKED because these disks are locked by another system and not available for access.

Although we plan to do the configuration on system XX, there is one part of it we must do on system YY. We must give system YY its switchover configuration name directly on the system itself. All other configuration work will be done on system XX.

After running SYSGEN on system YY, the following is typed at the sysgen prompt:

```
sysgen> spu
```

This will display a list of commands and give the spu prompt, at which the following is typed:

```
   spu> asystem YY
```

This is the only action taken on YY. The rest of the configuration is done on XX.

On system XX, sysgen is run and the spu command is entered. The following series of commands will:

1) name system XX for switchover configuration purposes

2) designate XX as the primary and YY as the alternate system

3) designate DATA_VOL_SET as a switchable volume set whose home system is XX

```
spu> asystem XX

spu> apair home=XX alt=YY

spu> avolset volset=DATA_VOL_SET home=XX
```

We could now use the show command to see what we have set up:

```
spu> show

Home                Alternate                Volume Sets
----                ---------                -----------
XX                  YY                       DATA_VOL_SET
```

At this point we are ready to save the configuration in the usual manner.

```
spu> hold
spu> exit

sysgen> keep
sysgen> exit
```

The switchover configuration is now ready to be activated. This is done using the new CI command SPUCONTROL.

After the initial switchover configuration (as well as after any changes are made to the switchover configuration) the SPUCONTROL SETUP command must be issued. This pushes the configuration file SPUINFOP over to the other system so that the two are in sync.

```
:SPUCONTROL SETUP
```

Now that the SPUINFOP files are in sync on the two systems, we are ready to complete the activation of the switchover environment with the following command:

```
:SPUCONTROL START
```

This begins the background communication between the two SPUs. At this point, the volume set DATA_VOL_SET can be switched from XX to YY when desired.

If the system XX were to suffer a failure, a repeating console message would appear on YY notifying the operator of that fact. The determination would then be made whether to switch the volume set over to YY or not. If it is determined that XX will be down for an extended period and a switchover is necessary, it can be effected from system YY with a single command:

```
:SWITCHOVER from=XX
```

This will mount all volume sets (in this case just DATA_VOL_SET) configured to be home to XX, on YY. They will go through the normal transaction management recovery process and then be available for access.

When XX has been repaired and we are ready to switch the volume set back to it, we must first close the set on YY.

```
:VSCLOSE DATA_VOL_SET
```

All files on the volume set must be closed before the VSCLOSE will succeed. Once the volume set is closed, we issue the following from XX:

```
:SWITCHBACK from=YY
```

This will mount the volume set on XX and it will be ready for immediate access. (There is no transaction management recovery here because the volume set was closed.)

Note that the volume set could have been switched from XX to YY at any time by simply issuing a VSCLOSE on the set from system XX, followed by the SWITCHOVER command on YY.

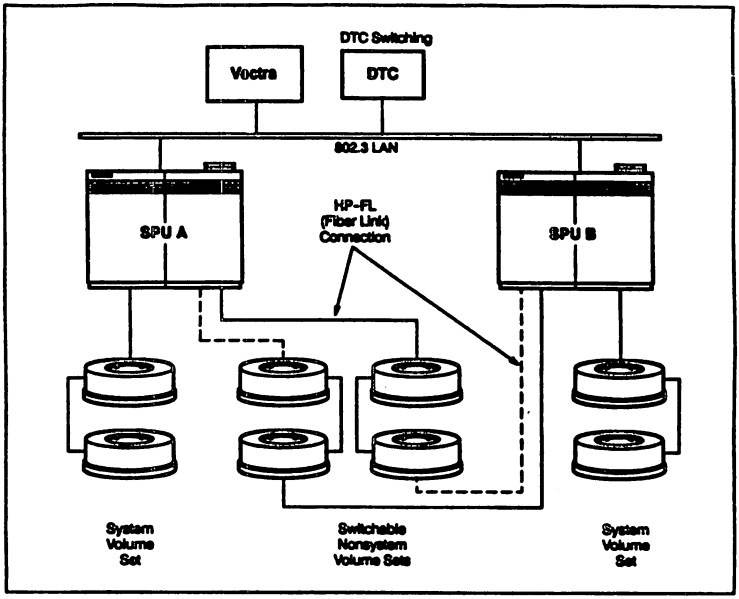


Figure 8. SPU Switchover/XL Configuration

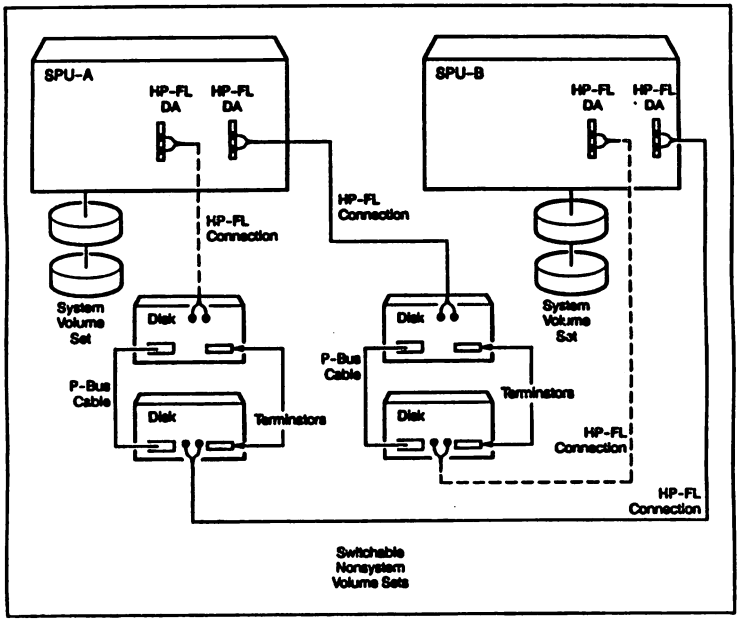


Figure 9. SPU Switchover/XL Hardware Configuration

HPVOLINFO - A NEW DISK MANAGEMENT INTRINSIC

by
Pat Alvarez
Lalitha Pejavar

Commercial Systems Division
Hewlett-Packard
19447 Pruneridge Ave
Cupertino, CA 95014
(408)725-8900

ABSTRACT

There has been a growing need among DP and MIS managers for a programmatic tool to track disk space usage. This would supplement the already available commands and utilities such as DSTAT, FREE5 and VOLUTIL which are essentially more interactive in nature. HPVOLINFO is a new intrinsic which has been added to the MPE intrinsic library (both MPE V and MPE/XL) designed to fill this need. It is intended to provide the user with the capability to programmatically extract disk space usage information. As the user's system becomes more complex due to increased number of disks with larger disk capacities, it becomes important to dynamically monitor the usage, especially when approaching the current maximum configuration limit of 64 spindles. Using the HPVOLINFO intrinsic to monitor disk space usage regularly, problems that are likely to impact performance and limit resource availability can be anticipated and timely preemptive action can be taken to avoid them. The HPVOLINFO intrinsic enables users to retrieve 45 different metrics related to disk management. Additionally, these metrics can be charted to produce a visual representation of usage trends. More importantly, the information generated by HPVOLINFO can be used to automatically trigger other actions specified by the user.

INTRODUCTION

On MPE, there are different methods to find out how much disk space is available or used through various commands and utilities - LISTF, REPORT, VINIT, FREE5, VOLUTIL, etc. But, currently, no intrinsics return this type of information and therefore, the process of gathering volume space information becomes a very manual process through the use of these commands and utilities. Since no volume intrinsics exist today, the customer must write privileged mode programs which are not supported by HP in order to retrieve volume information from the system tables. As a way to gather volume information more easily, the HPVOLINFO intrinsic has been added to the MPE intrinsic library on both MPE V and MPE/XL. Note: The terms "volume" and "disk" will be used interchangeably throughout this document.

HPVOLINFO is designed to provide the MIS and DP managers the capability to programmatically extract information regarding disk space usage from system and nonsystem volumes. The volume information that is returned can be used to track volume space usage. More specifically, the information that is returned describes how the space on volumes is allocated - i.e. how much is used for operating system purposes, how much space is free space, how much space is used for spool files, etc. In addition, HPVOLINFO provides information about the structure of the disks on a system, i.e. volume set names, volume class names,

volume names, etc.

The information that can be retrieved using HPVOLINFO includes:

- Drive type
- Capacity of a volume
- Sector size of a drive
- Amount of volume space used by MPE
- Spool file space usage
- Permanent file space usage
- Volume type
- Volume set name
- Number of volume classes
- List of volume classes
- Ldev that a volume is configured on
- Free space information
- Number of member volumes
- List of member volume names
- Number of volume sets
- List of volume sets

CHARACTERISTICS of HPVOLINFO

Using HPVOLINFO, up to 6 items of information can be retrieved for each call of the intrinsic for either a device (a volume), a set (one or more volumes logically grouped), or a class (logical groups in a volume set). The following is the syntax for an HPVOLINFO intrinsic call:

HPVOLINFO (status, volspecifiernum, volspecifier [,itemnum, item] [...]);

The parameter *volspecifiernum* is used to indicate the type of volume selection that is to be used to obtain information from HPVOLINFO. It can range from 0 to 5 depending upon whether the caller is specifying all volumes on a system, a specific logical device number, a volume set, a volume class, a volume name or a device class name. The default for *volspecifiernum* is 0, i.e. all volumes on the system, if the parameter is not specified. The next parameter, *volspecifier*, is used in conjunction with the *volspecifiernum* parameter and actually contains the information indicated in the former. For example, a *volspecifiernum* of 1 refers to a specific logical device number which is then passed as an integer in the *volspecifier* parameter. Similarly, a *volspecifiernum* of 2 indicates that the user is specifying a volume set name which then is passed in the *volspecifier* parameter as a character array. Since a *volspecifiernum* of 0 indicates all volumes, the *volspecifier* parameter is ignored. A detailed description of this usage is provided in the MPE Intrinsics Reference Manual.

The *itemnum* and *item* parameters (up to 6 pairs of itemnum parameters and associated item buffers) are for the actual characteristics that the user would query through this intrinsic. *Itemnum* is the cardinal number of the item desired (see summary of item numbers and items). For example, the user would use an itemnum of 5 to get the number of mounted volumes for a given set, and HPVOLINFO would return this integer value in the item parameter. Similarly, the user would use an itemnum of 7 to get the names of the volumes which are a part of a given set, and HPVOLINFO would return in the item parameter a character array consisting of volume names. A complete list of valid itemnums and the detailed information regarding the associated item buffers can be found in the MPE Intrinsics Reference Manual.

An optional *status* parameter is also provided that serves as a check as to whether the call was successful. However, it is strongly recommended that this *status* parameter is used in practice since failure to do so will cause the calling program to abort if the call was unsuccessful.

The following table summarizes the information returned by the HPVOLINFO intrinsic.

ITEM # 1: Reserved for system use.

ITEM # 2 and 3 : Number and List of Volume Sets

Item 2 returns the number of system and nonsystem volume sets configured on the system and Item 3 returns a list of all the system and nonsystem volume set names configured on the system. The user would typically want to use Item 2 in conjunction with Item 3 so that the value returned by Item 2 can be used to determine the size of the buffer to pass when calling HPVOLINFO with Item 3.

ITEM # 4 and 5: Number and List of Volume Classes

Item 4 returns the number of volume classes and Item 5 returns a list of volume class names associated with a volume or volume set. Since a volume can be associated with more than one volume class, Item 4 can be used to retrieve the number of volume classes belonging to a specific volume. Likewise, it can return the number of classes associated with a set. Similarly Item 5 can be used to return a list of volume classes that the volume is a member of or that is a subset of a set. The user would typically want to use Item 4 in conjunction with Item 5 so that the value returned by Item 4 can be used to determine the size of the buffer to pass when calling HPVOLINFO with Item 5.

ITEM # 6 and 7: Number and List of Member Volumes

Item 6 returns the number of member volumes and Item 7 returns the list of names of the member volumes in the specified volume set, volume class or device class. The user would typically want to use Item 6 in conjunction with Item 7 so that the value returned by Item 6 can be used to determine the size of the buffer to pass when calling HPVOLINFO with Item 7.

Table 1 - Summary of Item Numbers and Items (continued)

ITEM # 8 and 10: Drive Type and Volume Type

Item 8 returns the drive type and Item 10 returns the volume type of a given volume. The drive type refers to the name of the drive i.e. HP7935 or HP7937. Volume type will return whether a given volume belongs to either a system or nonsystem volume set.

ITEM # 9: Drive Sector Size

Item 9 returns the drives logical sector size. Currently, this logical size is always 256 bytes.

ITEM # 11 and 12: Volume Name and Volume Set Name

Item 11 returns the volume name and Item 12 returns the volume set name corresponding to the passed ldev. An ldev can be associated with only one volume set.

ITEM # 13: Logical Device Number

Item 13 returns the logical device number of the specified volume.

ITEM # 14 and 15: Volume Capacity

Item 14 and 15 return the volume capacity for a given volume or the total capacity of all the volumes for a given volume set or class.

ITEM # 16 through 29: MPE Overhead

Items 16 through 29 return MPE overhead or volume space that is used for operating system purposes. Item 16 and 17 return the total MPE overhead. This total consists of everything on a volume that is not set aside for file space use. On MPE V, the MPE overhead includes - volume label, virtual memory, directory, defective tracks/sector table, initial segments, disk cold load information table, volume table, free space map and channel programs; on MPE/XL - volume label, file label table, directory, volume set information table, free space map, transient space and transaction management overhead. A subset of the MPE overhead is returned through items 18 and 19 (MPE/XL transient space), 20 and 21 (MPE/XL configured transient space), 22 and 23 (MPE V Virtual memory), 24 and 25 (Directory), 26 and 27 (MPE/XL file label tables), and 28 and 29 (MPE/XL transaction management).

ITEM # 30 and 31: Spool File Disk Space Usage

Items 30 and 31 return spool file disk space usage. Spool file space consists of the volume space that is used by spool files. Spool files are files that are not a part of the permanent file space. This space can only be found on system volumes that are configured with the device class of SPOOL.

Table 1 - Summary of Item Numbers and Items (continued)

ITEM # 32 and 33: Disk Space Used by Permanent Files

Item 32 and 33 return the disk space used by permanent files.

ITEM # 36 through 43: Free Space Information

Items 36 through 43 return free space information for a volume or a group of volumes. By using Item 36 and 37, the user can pass an array that specifies a series of ranges, and the intrinsic will return the number of free areas whose size is within each of the ranges. Item 38 and 39 returns the free space distribution sectors per range. The user can pass an array that specifies a series of ranges, and the intrinsic will return, for each range specified in the array, the total free space for free areas found in that range. Item 40 and 41 return the total free space on a volume or a group of volumes. Item 42 and 43 return the largest contiguous free space area on a volume or a group of volumes.

Tabel 1 - Summary of Item Numbers and Items

INTRINSIC USAGE

This section describes how to obtain and use the information returned from the HPVOLINFO intrinsic.

Prior to the HPVOLINFO intrinsic, invoking MPE commands was the only way the user was able to get volume information. For example, the DSTAT command could be used to get the volume set name that a particular ldev is associated with, and the REPORT command could be used to get the amount of permanent file space used on the system and then the filespace sectors displayed by the REPORT command could be summed up. Using these commands the user could analyze the composition and management of his disks.

Consider, for example, trying to determine the disk space utilization of the system disks. For simplicity, we will assume that there are no user volumes (private volumes, on MPE V) on the system. The type of information the user would want to know includes: the amount of space used by MPE, i.e. directory space, virtual memory, volume label, etc., the amount of space being used for permanent files and spool files, and the amount of free space.

To obtain the space used for permanent files, spool files and free space, the user could do the following:

```
:DSTAT ALL                    Displays disk management information

:REPORT @.@                   Returns permanent file space

:SHOWOUT STATUS               Returns spool file space

:DISCFREE (on MPE V, FREE5)   Reports free space
```

Trying to determine the amount of directory space used or any disk space used by MPE becomes a little more difficult, if not impossible. In addition, trying to determine these figures for a particular disk as opposed to all the system disks, makes the task become even more difficult. Likewise, if this case was extended to include user volumes, gathering this type of information becomes even more complicated.

HPVOLINFO will now provide the user with a supported interface to the operating system to obtain volume information. HPVOLINFO allows the user to programmatically obtain all the information described above, along with some other features not mentioned in the example. It will return this information per disk, set, or class for both system and user volumes.

Some examples using HPVOLINFO include:

- Determine the amount of "lost disk space" on the system and therefore, predict when a "condense" (through VINIT) or "recover lost disk space" is needed
- Maintain a history of the disks on the system (i.e, fragmentation, allocation) in order to predict future disk requirements and forecast future disk space needs
- Customize the VOLUTIL "SHOW" commands

Determine "lost disk space"

On MPE V, any free unused space not referenced by the Disk Free Space Map (DFSM), is considered "lost disk space". For example, space results in this state when a system failure occurs due to temporary files being opened. This space can only be recovered while in INITIAL or through the CONDENSE command in VINIT. Because the time to recover this space is approximately 5 to 10 minutes for every 1000 files, this process can take a very long time for a very large system -- regardless of the amount of lost disk space.

In the past, typically, a "recover lost disk space" was done in a reactive type manner. For example, if users were experiencing "out of disk space" problems, this was a good sign that a "recover lost disk space" was needed and a decision had to be made as to whether recovery steps should be taken. Using HPVOLINFO, the amount of lost disk space on the system can be determined and prevent unneeded recovery from occurring. To determine lost disk space, each of the items below must be retrieved from HPVOLINFO and the following equation applied: (program illustrated in Example 1)

```
    capacity of all the disk drives   { Item 15 }
  - permanent file space              { Item 33 }
  - spool file space                  { Item 31 }
  - MPE overhead                      { Item 17 }
  - free space                        { Item 41 }
-------------------------------------------------------
  = lost disk space
```

```
program lost_disk_space (input, output);

const
    vol_set_specifier = 2;      { specifier is a volume set }

    max_items         = 6;      { 6 items of information can be retrieved }

    capacity_item     = 15;     { defines HPVOLINFO itemnum 15 }
    mpe_overhead_item = 17;     { defines HPVOLINFO itemnum 17 }
    spool_file_item   = 31;     { defines HPVOLINFO itemnum 31 }
    perm_file_item    = 33;     { defines HPVOLINFO itemnum 33 }
    free_space_item   = 41;     { defines HPVOLINFO itemnum 41 }

type
    int_type     = (int,sint);
    status_type  = record
                     case tag : int_type of
                       sint : (sint : packed array [1..2] of shortint);
                       int  : (int  : integer);
                   end;

    { volspecifier can be an integer or a character array }
    volspecifier_type = record
                          case integer of
                            0 : (pac : packed array [1..70] of char);
                            1 : (int : shortint);
                        end;

var
    status          : status_type; { returns information if error occurs }

    item_capacity,                 { returns HPVOLINFO item for capacity }
    item_mpe_overhead,             { returns HPVOLINFO item for overhead }
    item_spool_file,               { returns HPVOLINFO item for spool files }
    item_perm_file,                { returns HPVOLINFO item for perm files }
    item_free_space : longreal;    { returns HPVOLINFO item for free space }
    itemnum         : array [1..max_items] of shortint;

    volspecifiernum : shortint;         { defines HPVOLINFO volspecifiernum }
    volspecifier    : volspecifier_type;{ defines HPVOLINFO volspecifier }

    lost_space      : longreal;

procedure hpvolinfo; intrinsic;

begin

  status.int := 0;
```

Example 1 - Determining Lost Disk Space Using HPVOLINFO (continued)

```
  volspecifiernum := vol_set_specifier;          { volume specifier is system }
  volspecifier.pac := '%mpexl_system_volume_set%';{ volume set                }

  itemnum[1] := capacity_item;          { retrieve 5 items from HPVOLINFO }
  itemnum[2] := mpe_overhead_item;
  itemnum[3] := spool_file_item;
  itemnum[4] := perm_file_item;
  itemnum[5] := free_space_item;

  hpvolinfo(status.int, volspecifiernum, volspecifier,
            itemnum[1], item_capacity,
            itemnum[2], item_mpe_overhead,
            itemnum[3], item_spool_file,
            itemnum[4], item_perm_file,
            itemnum[5], item_free_space,);

  if status.int <> 0 then
    begin
      writeln('ERROR occurred in HPVOLINFO');
      writeln('Error     = ', status.sint[1]);
      writeln('Subsystem = ', status.sint[2]);
    end
  else
    begin
      { determine lost disk space and display values }
      lost_space := item_capacity - item_mpe_overhead -
                        item_spool_file - item_perm_file -
                        item_free_space;

      writeln('Capacity          =', item_capacity);
      writeln('Overhead          =', item_mpe_overhead);
      writeln('Spool file        =', item_spool_file);
      writeln('Perm file         =', item_perm_file);
      writeln('Free space        =', item_free_space);
      writeln('Lost Disk Space   =', lost_space);
    end;

end.
```

Example 1 - Determining Lost Disk Space Using HPVOLINFO

Tracking Disk Trends

Whether it is to determine how to allocate disk space for new projects or to determine when more disks are required, HPVOLINFO can be used to track disk space usage, and therefore, future disk needs can be predicted. This information can also be used for planning purposes. One way of doing this is to create a job that produces output reports based on the information returned by HPVOLINFO. This job can be scheduled at any specified time of the day on a routine basis. These output reports can be used for tracking disk space usage, and to

determine the appropriate steps required to assure efficient disk space use. Also, if desired, the output can be mapped to operator warning messages when deemed necessary. For example, warning messages can be used when it is determined that disk space limits are being reached. Likewise, warning messages can be used to let the operator know the space availability on particular disks.

Customize VOLUTIL "SHOW" commands

The SHOWSET, SHOWCLASS, and SHOWVOL commands in VOLUTIL display set, class and volume information. Since HPVOLINFO not only returns disk space information, but also returns formatting type information (volume set name, volume name, drive type), the user could use a combination of the information that HPVOLINFO returns to create his own customized VOLUTIL "SHOW" commands. For example, the output from the VOLUTIL SHOWSET STRUCT command for the system volumes on a system would look as follows:

```
volutil: showset mpexl_system_volume_set struct

Volumes in set: MPEXL_SYSTEM_VOLUME_SET

  MEMBER1
  MEMBER2
  MEMBER3

Classes in set: MPEXL_SYSTEM_VOLUME_SET

  DISC

Volumes in class: MPEXL_SYSTEM_VOLUME_SET:DISC

  MEMBER1
  MEMBER2
  MEMBER3
```

The same information can be returned by HPVOLINFO using items #6 and #7 (number of member volumes and list of member volumes) to return the "Volumes in set" and the "Volumes in class"; and items #4 and #5 (number of volume classes and list of volume classes) to return the "Classes in set".

LIMITATIONS

HPVOLINFO gives the user a snapshot of disk space at the time the call is executed. It does not give the user any indication of how the disk space is dynamically changing over time. Therefore, if a lot of activity is occurring on the system at the time the call to HPVOLINFO is made, some of the items returned may not reflect the expected result. For example, when HPVOLINFO is called to return the disk space used by permanent files, a value is returned. But, if immediately after the value is returned, a process on the system purges a file (FCLOSEs a file with disposition 4), the value will not reflect this difference. Also, a call to the intrinsic to retrieve certain items such as permanent file space will require considerable CPU resources and should be avoided at peak load times.

The HPVOLINFO intrinsic will return a "volume not mounted" error if the logical device number, volume set or volume class for which the information is requested, is not logically mounted. Logically mounted refers to invoking the commands LMOUNT or MOUNT on MPE V and VSRESERVESYS or VSRESERVE on MPE/XL. If a volume is taken offline while the process is accessing it through HPVOLINFO , the process will hang. This is because IOs cannot complete for that disk until the volume is back online.

On MPE/XL, since mirrored disks maintain identical copies of the same information on two disks, the values that are returned by this intrinsic reflect information from one of the volumes selected randomly in a mirrored disk pair.

CONCLUSION

The HPVOLINFO intrinsic fills the need for accessing information in a timely manner so that disk space management becomes more effective. The more complex systems get, the greater the need will be for accurate information extracting tools. For example, Volume Management in MPE/XL enables the user to control and manage his disk space allocation based on a functional and application based relationship. This is because of its well partitioned volumes, volume sets (System Volume Set and User Volume Set) and volume classes. HPVOLINFO takes advantage of this powerful Volume Management facility giving the user an efficient tool to manage his disk space. As a result, the user can better utilize the disk space as opposed to having to unnecessarily add more resources and thus, saving precious capital.

MPE XL Enhanced FOS Security

by
Rich Webber
MPE XL Support Engineer
Commercial Systems Division
Hewlett-Packard Company

INTRODUCTION

With the 3.0 release of MPE XL, Hewlett-Packard will introduce several new and enhanced system security features. Most of the information presented in this paper is available in the Fundamental Operating System manual set but it is spread across seven different manuals. It is provided here in an attempt to centralize it for easy reference.

The changes in MPE XL provided with Enhanced FOS Security address three primary areas:

- Discretionary Access Control
- Logon Access Security
- Security Auditing

Discretionary Access Control

File access on MPE XL can be made more secure by the implementation of a Discretionary Access Control (DAC) mechanism. This mechanism is discretionary, in that it is up to the owner of the object to grant access rights to whomever he wishes and access is based solely on this, not on the enforcement of some mandatory rules (e.g., a file access matrix).

Logon Access Security

Three new features have been added to improve logon access security:

- A new CI command for manipulating user passwords
- Enhancements to password prompting
- Job submitter banner

Security Auditing

The following security auditing enhancements have been implemented within the system logging facility:

- Logging of password changes
- Logging of system logging configuration
- Logging of restore
- Logging of printer access failure
- Logging of ACD changes
- Logging of stream initiation
- Logging of user logging
- Logging of process creation
- Auditability by named user

DISCRETIONARY ACCESS CONTROL

The C2 classification of the Trusted System Evaluation Criteria requires that a trusted system "shall define and control access between named users and named objects", and the enforcement mechanism "shall be capable of including or excluding access to the granularity of a single user". The DAC guideline from the National Computer Security Center recommends the use of Access Control Lists (ACLs) as the best way to meet the DAC requirements. Access Control Definitions will be the DAC mechanism on MPE XL. Access Control Definitions (ACDs) are extensions of Access Control Lists in the sense that an ACD not only contains a list of users who can access an object (e.g., a file or a device), but it may also contain restrictions such as the day and time at which a user can access the object. Each protected object can have an attached ACD which specifies who can or cannot access the object. At the time the object is initially accessed (e.g., file open), this function is evaluated to determine if the access attempt should be granted or denied.

Only a subset of the possible features of Access Control Definitions will be implemented on MPE XL. This implementation is fully compatible with the MPE V/E V-Delta-4 implementation of ACDs. Files and devices are the only objects protected by ACDs, and an ACD will only contain a list of users and the access modes each user has to the file or device.

ACD Overview

ACDs can be associated with files and devices. An ACD consists of a list of users and the access modes those users are granted for that file or device. Each entry in the list is termed the "userspec/mode pair."

As the creator of a file you may create an ACD for that file which could be defined as follows:

- ACD = (R : FARK.DOE ; W,A,L : @.DOE, @.PAYROLL)

or

- ACD = (NONE : RAY.DOE, @.CSSO ; R,W : @.@)

The first ACD would grant read access to FARK.DOE and write, append, and lock access to all users in the DOE and PAYROLL accounts. No other user can access the file except the owner (file creator, account manager, and system manager). The second example allows no access to RAY.DOE or any user in the CSSO account but read and write access to all other users on the system.

Each user specification (e.g., RAY.DOE or @.@) and associated modes make up an entry in the ACD. The maximum number of entries allowed in an ACD is forty (40), twice the maximum on MPE V/E.

ACD Security Policy

Only the owner of a file or device can create and change the ACD for that file or device. For a file, the owners are the file creator, the account manager of the account where the file resides, and the system manager. For devices, the owner of all devices is the system manager. The owners, by definition, have all access to the object.

When an ACD is attached to an object, it solely determines the access to the object. In other words, the ACD overrides existing file security mechanisms such as release/secure, file lockwords, and account, group and file level attributes. For devices, ACDs override ND capability check.

Having access to a file or device is different from having permission to access the ACD for that file or device. For example, if a user has read (R) access to a file or device, he may or may not have permission to read the ACD for that file, depending on if he is granted RACD permission.

The owner of an ACD has all permissions to the ACD. That is, the owner is the ONLY one who can create, delete, modify, list, and copy the ACD. However, the owner can authorize other users to access his ACD. By giving users "RACD" permission, the owner allows users to READ and COPY the ACD.

RACD is the only permission type allowed to be given to other users.

Where ACDs are Stored

An ACD for a file is stored in the file label extension for that file. An ACD for a device or class is stored in the file label extension of the appropriate file in the "3000devs" account.

How ACDs are Created and Maintained

File/device ACDs can be created and maintained through the :ALTSEC command and through the intrinsic HPACDPUT.

In addition, ACDs can also be created as the result of :COPY, :FCOPY, and :RESTORE when new files are created using these commands.

File ACDs become a part of the permanent file objects and therefore will survive a re-boot or optionally can be STORE'd or FCOPY'ed to a tape. Device ACDs, however, are not part of permanent objects and must be re-applied each time the system is re-booted. It is suggested that this be done by adding the appropriate :ALTSEC commands to create the device ACDs in the SYSSTART file or to a command file which is included in the SYSSTART file.

Manipulating ACDs using Commands

ACDs can be created, modified, and deleted using the :ALTSEC command. This command can also be used to copy an ACD from one object to another. The standard access specifications are used (e.g., **R, W, A, L, X**). Two new specifications have been added: **NONE** and **RACD** to allow no access whatsoever to the file and to allow read and copy access to the file's ACD, respectively.

To see the contents of an ACD, the :LISTF and :LISTFILE commands are used for files and the :SHOWDEV command is used for devices.

```
:ALTSEC
                                    [;NEWACD =  {(pair_spec)}                    ]
                                                {^acdfilename}                   ]
                                                                  [,FILENAME*]   ]
                                    [;COPYACD = {sourceobjectname}  [,LDEV     ]]
                                                                    [,DEVNAME  ]]
                        [,FILENAME*]            {(pair_spec)}                    ]
ALTSEC  objectname      [,LDEV     ] [;ADDPAIR = {^acdfilename}                 ]
                        [,DEVCLASS]
                        [,DEVNAME ] [;REPPAIR = {(pair_spec)}                    ]
                                                {^acdfilename}                   ]
                                    [;DELPAIR = {(userspecification)}            ]
                                                {^acdfilename}                   ]
                                    [;DELACD                                     ]
```

* - Default entry

```
Examples:       ALTSEC AFILE;NEWACD=(R,W:OPERATOR.SYS;X:@.@)
                ALTSEC 22,LDEV;COPYACD=21,LDEV
```

```
:LISTF XYZZY, 4
```

The output of this command will show if the file has an associated ACD, and what access the user has according to that ACD.

```
    ********************
    FILE: XYZZY.PUB.CSSO

    SYSTEM     READ:     ANY
    SECURITY--WRITE:     AC
     (ACCT)    APPEND:   AC
               LOCK:     AC
               EXECUTE:  ANY

    SYSTEM     READ:     GU
    SECURITY--WRITE:     GU
     (GROUP)   APPEND:   GU
               LOCK:     GU
               EXECUTE:  GU
               SAVE:     GU

    SECURITY--READ:      ANY                  FCODE: 0
     (FILE)    WRITE:    ANY                **SECURITY IS ON
               APPEND:   ANY                  ACD EXISTS
               LOCK:     ANY
               EXECUTE:  ANY

    FOR JOE.CSSO: READ,WRITE,APPEND,LOCK,EXECUTE
```

The "ACD EXISTS" display is new. Other possible displays in this place are: NO ACD and ACD CORRUPTED.

```
:LISTF XYZZY,-2
```

This new "-2" option of the :LISTF command will display the content of the ACD associated with the file. If there is no ACD applicable to the file, the output will show "NO ACD". Output for this command will contain all entries of the ACD as follows.

```
ACCOUNT=  CSSO        GROUP=  PUB

FILENAME        ------------ACD ENTRIES--------------

XYZZY               JOE.DOE             : R
                    @.OSE               : R,W,A,L,X
                    @.@                 : X
```

The new :LISTFILE command contains an option to list the ACD associated with the file. The output for this option is the same as shown above.

:SHOWDEV has been modified to display ACDs associated with devices.

```
          [devname     ]
:SHOWDEV  [ldev        ] [;ACD]
          [devclassname]
```

An example output from the :SHOWDEV command with the ACD option included is as follows:

```
:SHOWDEV 14;ACD

LDEV  AVAIL          OWNERSHIP          VOLID          DEN   ASSOCIATION

 14   SPOOLED        SPOOLER OUT
      ACD ENTRIES:   @ @ : R,W,X
```

Manipulating ACDs using Intrinsics

Programmatic query and manipulation of ACDs is accomplished through the use of the HPACDINFO, HPACDPUT, and HPFOPEN intrinsics.

HPACDINFO allows a program to obtain information regarding an ACD's number of entries, version identification, and identity of the first user in the access list. For each additional entry, information that can be obtained includes the specified user's access modes and the next user identity in the list.

HPACDPUT allows a program to create, delete and copy entire ACDs for specified objects. Additionally, user specification/pairs can be added, replaced or deleted.

Additionally, a new option, HOP_OPTION_ACD (=64), has been added to the list of options for the HPFOPEN intrinsic. This option allows the caller to specify an ACD specification to be applied to the new file which is created by the HPFOPEN call. This option is only legal for new files.

How ACDs are Used

ACDs are used to determine if a user trying to access a file/device is authorized to do so. When there is an ACD associated with a file/device, the old access matrix (including lockwords) is not used to determine who can access a file/device. Instead, the ACD is the only mechanism used to determine who can access the file/device.

Thus, ACDs are checked in two different ways: when accessing a file, and when acquiring a device.

Accessing a File

In order for a user to access a file, the system will execute the following checks at HPFOPEN/FOPEN time:

1. The system will check whether the user is any of the following:

   - System Manager (has SM capability),
   - Account Manager (has AM capability), or
   - CREATOR of the file.

If the user is any of these three, then access to the file is granted. Otherwise, the following check is executed by the system.

2. The system checks if there is an ACD associated with the file. If there is one, the system evaluates the ACD to determine whether the user is granted access to the file. This evaluation is done by matching the user to any of the user specifications given in the ACD.

Matching the user to the ACD is done as follows:

- the user name is compared to all the specific names (username.accountname) in the ACD. If the name does not match, then
- the user name is compared to all the account groupings (@.accountname) in the ACD. If no matching account entry is found, then
- the user name is compared to any system grouping present (@.@). If no system entry (@.@) is found in the ACD, then
- the user is not granted access.

If a match is found, the user is given the access and permissions specified in the match entry.

3. If no applicable ACD is found, then the old access matrix and lockword are used to determine if the user is granted access to the file. When a user is denied access to a file, whether because of ACD verification or according to the old access matrix, the same file system error, security violation, is returned to the user.

Special Case:

PRIVILEGED FILES:

1. The system checks whether the process has PM capability, whether the file code matches, and the privilege level is the correct one. If none of these conditions hold, then the user is denied access. If all of these conditions hold, then

2. The system checks if there is an ACD associated with the file. If there is one, the system evaluates the ACD to determine whether the user has access to the file.

3. If there is no ACD, then the old access matrix and lockwords are used to determine if the user is granted access to the file.

Acquiring a Device

For a user to acquire a device, the system will execute the following checks at HPFOPEN/FOPEN time:

1. The system will check whether the user is System Manager or the process has PM capability, in which case the user is allowed to acquire the device.

2. Otherwise, the system checks if the device has an ACD associated with it. If there is an ACD, then the system evaluates the ACD the same way that is described above. That is, the system tries to match the user's name with any of the user specifications in the ACD. If there is a match, then the user is allowed to acquire the device as specified in the ACD. Otherwise, the user is disallowed acquisition of the device.

3. If there is no ACD, then acquiring the device is done as is done today.

A user FOPENs a device by giving either a device class name, a device name, or a device number.

If the user FOPENs a DEVICE CLASS then, the system will take the first available device which belongs to that device class and perform the checks described above to determine if the user is allowed to acquire the device. If the user is disallowed, then the user will have to try again.

If the user FOPENs a DEVICE NUMBER then, the system will only try to match the user with a user specification in the ACD for the specific device. If there is a match, then the user is granted access. If there is no match then the user is denied access.

When a user is denied access to a device, whether because of ACD verification or according to the old access checking method, the same file system error, security violation, is returned to the user.

Commands to COPY an ACD

There are several ways ACDs can be copied. Use the **:ALTSEC** command to copy an ACD from one file/device to another. The **:COPY** command, the **FCOPY** utility, and **STORE/RESTORE** operate on files only. Detailed descriptions follow.

:ALTSEC

```
                    [,FILENAME*]                                        [,FILENAME*]]
ALTSEC objectname   [,LDEV     ]  [;COPYACD = (sourceobjectname)  [,LDEV      ]]
                    [,DEVCLASS]                                        [,DEVNAME  ]]
                    [,DEVNAME ]
```

* - Default entry

Wild cards will be allowed for the target filename so a user can copy an ACD to more than one file at a time. Wild card characters allowed and their meanings are the same as in the :LISTF command.

Similarly, an ACD can be copied to all devices on the system by using the "@" wild card instead of targetdevicenumber/targetdevicename.

:COPY

The COPY command has been modified to always copy the ACD associated with the source file to the target file, if one is present. No syntax change is needed for the :COPY command.

:COPY [from=] sourcefile ; [to=] targetfile

FCOPY

The FCOPY subsystem has also been modified to copy a file's ACD by default. Use the ;NOACD option to create the new file without an ACD:

:FCOPY from=sourcefile;to=targetfile [;NOACD]

Note that copying the file will fail if the user copying the ACD is not authorized to do so. That is, neither the file nor the ACD will be copied even if the user is authorized to copy the file.

STORE

STORE has been modified to store ACDs associated with files by default. If it is desired to have files stored without their ACDs, use the ;NOACD option on the STORE command line.

:STORE filename ... [;NOACD]

When storing a file with an ACD associated with it, the ACD will be evaluated to check if the user is granted access to store the file and the ACD (RACD permission is required). If the user is not granted access to copy the ACD, then storing the file will fail.

If wild cards are used to define file names, then the system will try to store each of the files with their associated ACD. If in the process there are files for which storing the ACD fails, an appropriate message will be displayed.

RESTORE

As with STORE, the default for RESTORE is to copy a file's associated ACD when restoring the file. Use the ;NOACD option on the RESTORE command line to override the default and not copy the ACD.

```
:RESTORE filename ... [;NOACD]
```

When restoring a file which has an ACD associated with it, the ACD will be evaluated to check if the user is granted access to the file and the ACD (RACD permission is required). If the user is not granted access to copy the ACD, then restoring the file will fail. Otherwise, the ACD will be copied from tape and attached to the restored file on disk.

If wild cards are used to define file names, then the system will try to restore each of the files with its associated ACD. If in the process there are files for which restoring the ACD fails, an appropriate message will be displayed.

Migration of ACDs

Migration of file ACDs is accomplished via the STORE/RESTORE process. Transporting files with ACDs is possible both ways, from an MPE V/E system to MPE XL, and vice versa.

Since device ACDs are tied to the system configuration of a specific system, device ACD migration from one system to the next is not available.

ACDs on MPE XL vs. MPE V/E: Key Differences

Although ACDs are the same on MPE XL as those on MPE V/E, there are some differences between the two versions as noted below:

- The maximum number of ACD entries (pairs) has been increased from 20 to 40 on MPE XL.
- Unlike on MPE V/E where device ACDs are permanent until changed, device ACDs on MPE XL do not survive across system startups. This is because configuration may change at any startup and all device and class files in the "3000devs" account (where ACDs are kept for devices) are rebuilt at every start. To keep the same device ACDs, the :ALTSEC command can be used in the SYSSTART file to create the ACDs at every start.
- Internally, MPE XL ACDs are kept in the File Label Table, a disk data structure of MPE XL. However, on MPE V/E an ACD is physically part of the file (kept in the file's pseudo extent).
- File wildcards can be used with the :ALTSEC command on MPE XL to manipulate the ACDs for multiple files. This feature is not implemented on MPE V/E (can only do one file at a time).
- The HPACDPUT and HPACDINFO intrinsics are fully compatible with those on MPE V/E. However, additional options have been added to allow users to specify a file by its Unique File Descriptor (UFID), and to specify a device by its device name.
- When spoolfiles are transferred from MPE V/E to MPE XL using the SPFXFER program, if the file creator does not have access to the printer the spoolfile is put in DEFER state.

LOGON ACCESS SECURITY

This section describes the logon access security features implemented for FOS Security:

- A new CI command for manipulating user passwords
- Enhancements to password prompting
- Job submitter banner

:Password Command

The :PASSWORD command is a new FOS command which allows all users to change the password associated with their user name. Prior to the introduction of this command only account or system managers could manipulate user passwords.

When a user name is associated with a non-blank password, users are re-authenticated before they are allowed to change the password. This re-authentication protects against a person other than a user ID's owner from walking up to an unattended terminal and using the :PASSWORD command within a logged-on session to change a user password. Re-authentication is performed by prompting users for their user passwords and verifying their responses. If an incorrect re-authentication password is entered, the :PASSWORD command terminates after displaying an error message on the user's terminal and a message on the system console.

New passwords are prompted for twice in order to catch nonrecurring typographic errors. All password responses are converted to upper case upon input. If both new password responses are not the same (ignoring case), the password will not be changed and the command will terminate with an error message. New passwords must satisfy password syntax rules and be different than the old user password. These password syntax rules are the same as the rules enforced for the :NEWUSER and :ALTUSER commands: the password must be no longer than eight characters long, begin with a character, and contain only alphanumeric characters. A message indicating whether a user password has been changed is displayed on the user's terminal before the command terminates.

The CI's HPTIMEOUT variable value is enforced while waiting for user responses to password prompts. If this time interval expires while the :PASSWORD command is waiting for a user to enter a password, the session is terminated just as if the time interval expired while waiting at the CI prompt for input.

Because passwords should not be stored in files where they can be discovered by other users, and non-interactive input must be stored in some type of file, the :PASSWORD command has been restricted to be executable only in interactive sessions whose $STDIN and $STDOUT have not been redirected.

Password privacy would be seriously undermined if the :PASSWORD command could be invoked from within a job file containing the new password. Programs may execute the :PASSWORD command programmatically as long as the program is executing within a session environment which satisfies the previously stated restrictions. The :PASSWORD command avoids displaying passwords on full duplex terminals by disabling character echo while prompting for passwords and by not allowing passwords to be specified as command line parameters.

:PASSWORD Dialogue Examples

Successful Password Replacement

```
:PASSWORD
ENTER OLD USER PASSWORD:          (old password entered)
ENTER NEW USER PASSWORD:          (new password entered)
ENTER NEW USER PASSWORD AGAIN:          (new password entered again)
PASSWORD WAS CHANGED SUCCESSFULLY.
:
```

Unsuccessful Re-authentication

```
:PASSWORD
ENTER OLD USER PASSWORD:          (incorrect password entered)
INCORRECT PASSWORD.  (CIERR 2502)
PASSWORD WAS NOT CHANGED.
:
```

Unsuccessful New Password Verification

```
:PASSWORD
ENTER OLD USER PASSWORD:          (old password entered)
ENTER NEW USER PASSWORD:          (new password entered)
ENTER NEW USER PASSWORD AGAIN:          (typographic error)
NEW PASSWORD IS NOT CONSISTENT. (CIERR 2503)
PASSWORD WAS NOT CHANGED.
:
```

Interactions with Other Features

Successful invocations of the :PASSWORD command will generate password change log records if system logging of password changes has been enabled.

Compatibility

The MPE XL :PASSWORD command is upwards compatible with the MPE V/E :PASSWORD command. The MPE XL :PASSWORD command differs from the V/E command in three ways. User passwords are changed only when the new password is different than the old password. Secondly, the MPE XL :PASSWORD command is programmatically executable. Thirdly, the CI's HPTIMEOUT value is enforced while waiting for user input.

Enhanced Logon Password Prompting

Logon password prompting is being enhanced in two ways. The first enhancement is that all password prompts will contain the account, user, or group name in addition to the type of password (account, user, or group) which is currently displayed. For example, the logon prompt requesting the user password associated with the MANAGER.SYS user ID will be **ENTER USER (MANAGER) PASSWORD:**. The second enhancement is that the password prompting which currently occurs for the :HELLO and :CHGROUP commands will be extended to the following areas:

- :STARTSESS commands within sessions
- STARTSESS intrinsic calls in programs executing within sessions
- :STREAM commands within sessions, prompting will occur for each first level :JOB and :DATA command

Password prompting and verification will continue to follow the existing rules for the :HELLO command. Users will be prompted a maximum of three times for each password. Each failure to supply a correct password will cause a message containing the user's identity and the logical device number of the logon device to be displayed on the system console. If the logical device belongs to a device class which has been associated with a user, the error message will be displayed on the $STDLIST of the associated user rather than on the system console. Failure to supply a correct response after being prompted for the same password three times will cause the error message **INCORRECT PASSWORD. (CIERR 1441)** to be displayed on the user's terminal. :STREAM commands will ignore input after this error occurs until either another :JOB or :DATA command is read or an end of file occurs; other commands will terminate after displaying this error message.

Because the :DATA command does not include a group specification, group password prompting does not apply to :DATA commands. Password prompting does not occur when $STDIN or $STDLIST have been redirected, because reading responses from a redirected $STDIN would be just another form of embedding passwords in command lines. Allowing passwords to be ommitted from job files will help reduce the risk of password exposure due to passwords embedded in job files.

ENHANCED PASSWORD PROMPTING EXAMPLES. The following logon password prompting examples assume the MANAGER.SYS user ID has both user and account passwords and that the SECURITY group has a group password. The following examples are assumed to have run under session number 72 logged on as the OPERATOR.SYS user ID using a terminal with the device name OPERTERM.

Job File :STREAM Example

```
:PRINT JOBFILE
!JOB JOBFILE,MANAGER.SYS,SECURITY
!SHOWME
!EOJ
:STREAM JOBFILE
ENTER ACCOUNT (SYS) PASSWORD:          (correct password supplied)
ENTER USER (MANAGER) PASSWORD:         (correct password supplied)
ENTER GROUP (SECURITY) PASSWORD:       (correct password supplied)
#J420
:
```

Interactive :STREAM Example

```
:STREAM
>!JOB INTERACT,MANAGER.SYS,SECURITY
ENTER ACCOUNT (SYS) PASSWORD:          (correct password supplied)
ENTER USER (MANAGER) PASSWORD:          (correct password supplied)
ENTER GROUP (SECURITY) PASSWORD:          (correct password supplied)
>!SHOWME
>!EOJ
#J421
>:
:
```

Job Authentication Failure Example

```
:STREAM JOBFILE
ENTER ACCOUNT (SYS) PASSWORD:          (incorrect password supplied)
ENTER ACCOUNT (SYS) PASSWORD:          (incorrect password supplied)
ENTER ACCOUNT (SYS) PASSWORD:          (incorrect password supplied)
INCORRECT PASSWORD.  (CIERR 1441)
:
```

On the system console the message **INVALID PASSWORD FOR "!" DURING LOGON ON LDEV #\. (js 65)** is displayed three times, once for each incorrect password. The user's logon information [jsname],user.account,group and the ldev of the user's logon device are substituted into the console message.

Interactions with Other Features

Existing password prompting will use the new password prompts which include the user's user, account, or group name. For example for the command :HELLO MANAGER.SYS the new prompt will be **ENTER USER (MANAGER) PASSWORD** rather than the old **ENTER USER PASSWORD** prompt.

The password prompting for the :CHGROUP command will be made consistent with other password prompting. An **INCORRECT PASSWORD (CIERR 1441)** error message will no longer be displayed on the user's terminal for the first two incorrect attempts for each password. This error message was not displayed during other password prompting dialogues. The present inconsistent behavior was the result of a programming defect in the STARTLOGON module.

If the Stream Initiation log type is enabled, a log record will be logged for all successful batch submissions.

Job Password Prompting Limitations

Although job password prompting can reduce the need to embed passwords in job files it does not eliminate this need. Jobs which are streamed by jobs must still contain embedded passwords. Job password prompting also does not address the desire to limit knowledge of authentication passwords to the user responsible for a user ID. Users must reveal their authentication passwords to other users if they wish to allow other users to submit jobs on their behalf.

Job Submitter Banner

The job submitter banner information displayed to $STDLIST is the date and time a job was submitted, the job submitter's user identity (user.account), job/session number, and logon device number. This banner is displayed between the logon banner and the welcome message in the format:

```
STREAMED BY jsname,user.account (#jsnum) on LDEV# ...
   STREAM DATE:  day, mmm dd, yyyy,  hh:mm AM/PM
```

An example of the banner format is:

```
STREAMED BY BACKUP,OPERATOR.SYS (#S73) ON LDEV# 20
   STREAM DATE:  MON, APR 15, 1991, 10:50 AM
```

Some jobs are initiated by a system process. An example might be a job streamed as part of the system startup activities contained in the SYSSTART file. The job submitter information for these jobs will be as follows:

```
STREAMED BY  (SYSTEM PROCESS) ON LDEV# 20
   STREAM DATE:  MON, APR 15, 1991, 10:50 AM
```

The job submitter information is kept in a new data segment (DST #61). The file that preserves this DST on disk is called DSTJSEC.PUB.SYS.

SECURITY AUDITING

System logging has been enhanced with eight new logtypes which provide additional auditing information. The new logging events are:

- Logging of password changes is added as log type 134. It can be activated to record USER, GROUP, and ACCOUNT password change events.
- Logging of system logging configuration is added as log type 135. It records the current system logging configuration.
- Logging of RESTORE is added as log type 136. It records restoration of files onto the system.
- Logging of printer access failure is added as log type 137. It allows the system manager to audit failures in attaching spoolfiles to printers.
- Logging of ACD changes is added as log type 138. It records all ACD creations, alterations, and deletions.
- Logging of stream initiation is added as log type 139. It allows the system manager to know who streams or tries to stream a job.
- Logging of user logging is added as log type 140. It records all OPENLOG and CLOSELOG intrinsic calls.
- Logging of process initiation is added as log type 141. It provides information for tracing the creation of processes on the system.
- Auditability by named user provides the ability to selectively audit the actions of one or more users based on individual identity. LOGTOOL will allow users of the utility to format log records selected by user identifications.

NOTE

With these enhancements to auditing, system log files will be filling up faster. This will require more attention to the archiving of old log files.

SYSGEN will show system logging configuration as:

configurable item	max	min	current
# of user logging processes	64	2	64
# users per logging process	256	1	128

system log events	event #	status
System logging enabled	100	OFF
System up record	101	ON
Job initiation record	102	OFF
Job termination record	103	OFF
Process termination record	104	OFF
File close record	105	OFF
System shutdown record	106	ON
Power failure record	107	ON
Spooling log record	108	OFF
I/O error record	111	ON
Physical mount/dismount	112	OFF
Logical/mount/dismount	113	OFF
Tape labels record	114	OFF
Console log record	115	ON
program file event	116	ON
New commercial spooling	120	ON
Architected interface	130	ON
Password changes	134	ON
System logging configuration	135	ON
Restore logging	136	ON
Printer access failure	137	ON
ACD changes	138	ON
Stream initiation	139	ON
User logging	140	ON
Process creation	141	ON
Chgroup record	143	ON
File open record	144	ON
Maintenance request log	146	OFF
diagnostic information record	150	ON
high priority machine check	151	OFF
low priority machine check	152	OFF
CM file close record	160	OFF

Logging of Password changes

This feature allows the system manager to audit the changing of passwords via the MPE XL commands (:ALTACCT, :ALTGROUP, :ALTUSER, and :PASSWORD) and the ;DIRECTORY option of the RESTORE utility.

A new log type 134 will be added to the system logging facility. The MPE operating system and LOGTOOL will recognize this new log event. It can be enabled by SYSGEN followed by a START NORECOVERY. This event is initially disabled.

System logging will record when a user, group or account password is changed via MPE commands or other HP utility programs. The log record will contain the following information:

Header:

- record type
- record length
- timestamp
- job/session number
- PIN

Log information:

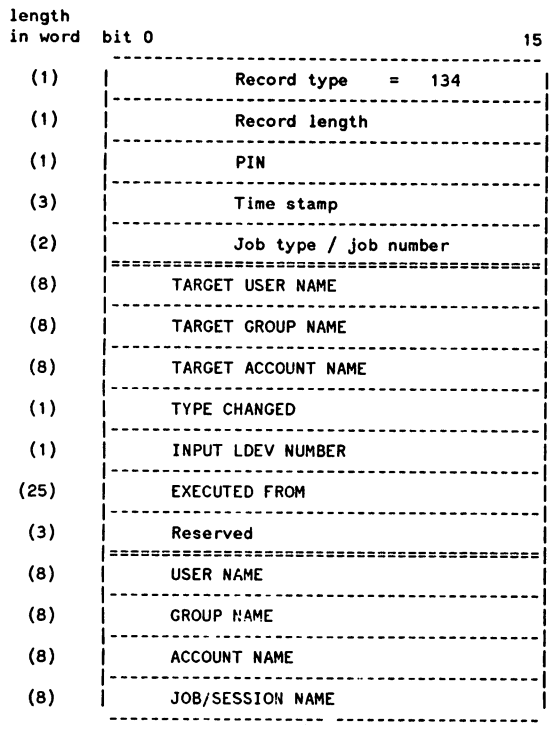
- identification of user who changed password (includes job/session name, user name, group name, account name)
- identification of user whose password was changed (includes account name if account password changed; group name if group password changed; user and account names if user password changed)
- input logical device number from which the password was changed
- program file name from which password change was executed
- type changed (1 User; 2 Group; 4 Account)

For example, JOHN.PAYROLL,DOE with job/session name JREPORT, successfully changed account password for PAYROLL account via the command executor. The change was made from ldev 21.

LOGTOOL will format the following layout after the standard header:

```
TARGET USER:                    TARGET GROUP:
TARGET ACCOUNT:    PAYROLL      TYPE CHANGED:    ACCOUNT
LDEV:              21
EXECUTED FROM:     CI.PUB.SYS
USER:              JOHN         GROUP:           DOE
ACCOUNT:           PAYROLL      JSNAME:          JREPORT
```

The following is the password changes logging record format:



Logging of System Logging Configuration

The run time system logging configuration can be altered at boot time using a configuration file created by SYSGEN. This feature will provide the audit trial of the logging configuration changes.

A new log type 135 will be added to the system logging facility. It can be enabled by SYSGEN followed by a START NORECOVERY. This event is initially enabled.

The MPE operating system and LOGTOOL will recognize this new log event. When this feature is enabled, system logging configuration will be recorded when the system is started.

The log record will contain the following information:

Header:

- record type
- record length
- timestamp
- job/session number
- PIN

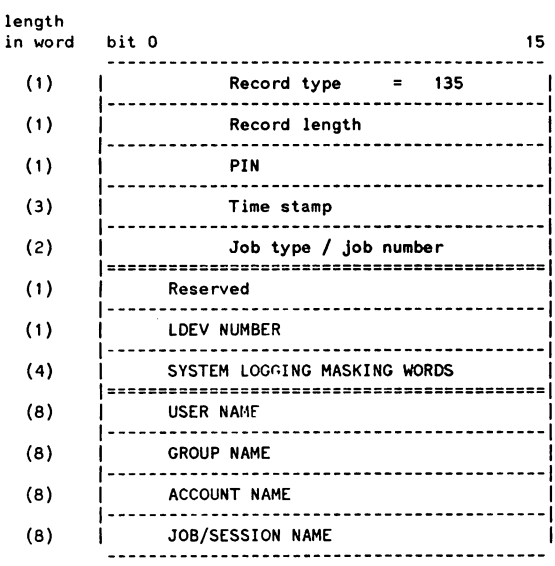
Log information:

- identification of the user who makes changes to the configuration (includes job/session name, user name, group name, account name)
- logical device number
- new system logging configuration

For example, user CONFIG,MANAGER.SYS,PUB changed the system logging configuration from ldev 20. The system log types 100, 101, 102, 111, 151, and 152 are enabled. LOGTOOL will format it as:

```
LOGICAL DEVICE:      20
SYSTEM LOG CONFIG:
LOG FAILURE:         ON              SYSTEM UP:           ON
JOB INITIATION:      ON              JOB TERMINATION:     OFF
PROCESS TERMINATION:OFF              NM FILE CLOSE:       OFF
SHUTDOWN:            OFF             POWER FAILURE:       OFF
SPOOLING:            OFF             I/O ERROR:           ON
PHY. MOUNT/DISMOUNT:OFF              LOG. MOUNT/DISMOUNT:OFF
TAPE LABELS:         OFF             CONSOLE LOG:         OFF
PROGRAM FILE EVENT:  OFF             NCS LOGGING:         OFF
AIF LOGGING:         OFF             PASSWORD CHANGE:     OFF
SYS LOG CONFIG:      OFF             RESTORE:             OFF
PRINTER ACCESS:      OFF             ACD CHANGE:          OFF
STREAM INITIATION:   OFF             USER LOGGING:        OFF
PROCESS CREATION:    OFF             SECURITY CONFIG:     OFF
CHGROUP:             OFF             FILE OPEN:           OFF
COMMAND LOGGING:     OFF             AUTO-DIAG/SUM:       OFF
HPMC:                ON              LPMC:                ON
CM FILE CLOSE:       OFF
USER:                MANAGER         GROUP:               PUB
ACCOUNT:             SYS             JSNAME:              CONFIG
```

The following is the system up log record format:



Logging of Restore

The creation of files via RESTORE needs to be auditable.

A new log type 136 will be added to the system logging facility. It can be enabled by SYSGEN followed by a START NORECOVERY. This logging type is initially disabled.

The MPE operating system and LOGTOOL will recognize this new log event. If this feature is enabled, the system logging facility will log restore of files.

The log record will contain the following information:

Header:

- record type
- record length
- timestamp
- job/session number
- PIN

Log information:

- identification of the user who restored a file (includes job/session name, user name, group name, account name)
- name of the file which was restored (includes file name, group name, account name)
- name of the creator of the file
- disk volume identification where the file will reside. It consists of volume type and volume type name. (volume type: 0 = volume name, 1 = volume class name, 2 = volume set name)
- access type: 1 = Restores a new file; 2 = Replaces an existing disk file

For example, a user, JOHN.PAYROLL,DOE with job/session name as JREPORT, restored a new file FTEST.TESTGP.PAYROLL on a volume set called MY_TEST_VOL_SET. The file creator was DOLE.

LOGTOOL will format the following layout after the standard header:

```
FILE NAME:          FTEST          FILE GROUP:          TESTGP
FILE ACCOUNT:       PAYROLL        CREATOR:             DOLE
ACCESS TYPE:        1
VOLUME ID:          MY_TEST_VOL_SET
USER NAME:          JOHN           USER GROUP:          DOE
USER ACCOUNT:       PAYROLL        JOB/SESSION NAME:    JREPORT
```

The following is the Restore log record format:

length in word	bit 0 15
(1)	Record type = 136
(1)	Record length
(1)	PIN
(3)	Time stamp
(2)	Job type / job number
(8)	FILE NAME
(8)	FILE GROUP
(8)	FILE ACCOUNT
(8)	CREATOR
(17)	VOLUME IDENTIFICATION
(1)	ACCESS TYPE
(8)	USER NAME
(8)	GROUP NAME
(8)	ACCOUNT NAME
(8)	JOB/SESSION NAME

Logging of Printer Access Failure

This feature allows the system manager to audit failures in attaching spoolfiles to printers. This does not include creating new spoolfiles which are logged by FOPEN.

A new log type 137 will be added to the system logging facility. It can be enabled by SYSGEN followed by a START NORECOVERY. This event is initially disabled.

The MPE operating system and LOGTOOL will recognize this new log event. When this feature is enabled, failures to attach spoolfiles to printers will be logged.

The log record will contain the following information:

Header:

- record type
- record length
- timestamp
- job/session number
- PIN

Log information:

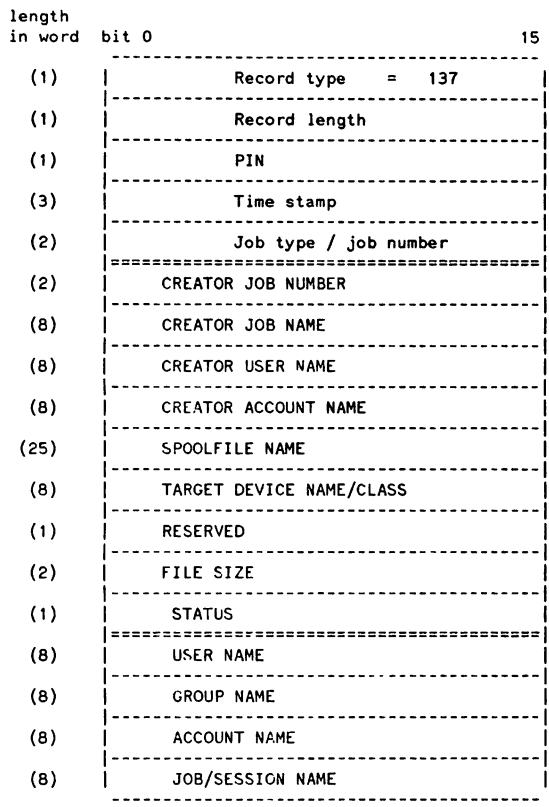
- identification of the user who owns the output spoolfile (includes job number, job name, user name, account name)
- spoolfile name
- target device name/class
- number of records in the output spoolfile
- failure flag: 1 = internal failure of access check; 2 = security violation

For example, a user, JOHN.PAYROLL,DOE with job number #J12 and job name as JREPORT, spooled an output file FNAME.TEST.PAYROLL with file size of 200 records to a printer whose target device class name was LP. It failed in device ACD security check.

LOGTOOL will format the following layout after the standard header:

```
USER:              JOHN                ACCOUNT:           PAYROLL
J/S TYPE  :        JOB                 JOB NUMBER:        12
JOB NAME:          JREPORT
SPOOLFILE NAME:    FNAME.TEST.PAYROLL
FILE SIZE:         200                 TARGET DEVICE:     LP
FAILURE FLAG:      Security violation
USER NAME:         JOHN                USER GROUP:        DOE
USER ACCOUNT:      PAYROLL             JOB/SESSION NAME:  JREPORT
```

The following is the printer access failure logging record format:



Logging of ACD changes

ACDs can be changed by MPE commands and intrinsic calls. This feature allows the system manager to audit both types of these events.

A new log type 138 will be added to the system logging facility. The MPE operating system and LOGTOOL will recognize this new log event. It can be enabled by SYSGEN followed by a START NORECOVERY. This event is initially disabled.

System logging will record when ACDs are changed (created, deleted, copied, or modified) via MPE commands and intrinsic calls. The log record will contain the following information:

Header:

- record type
- record length
- timestamp
- job/session number
- PIN

Log information:

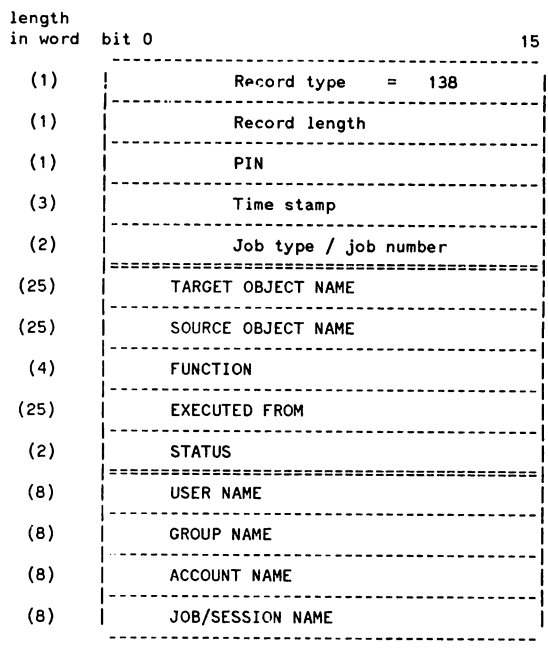
- identification of the user who changed ACD (includes job/session name, user name, group name, account name)
- object type and object name whose ACD was changed
- object type and object name from where ACD was copied
- type of change to the ACD: create, add pair, replace pair, copy, delete pair, or delete
- program file name from which ACD change was executed
- status returned (HPE status)

For example, a user JOHN.PAYROLL,DOE with job/session name as JREPORT, successfully created an ACD for a file FTEST.TESTGP.PAYROLL from the command executor.

LOGTOOL will format the following layout after the standard header:

```
TARGET OBJECT:    FTEST.TESTGP.PAYROLL
SOURCE OBJECT:
FUNCTION:         CREATE
EXECUTED FROM:    CI.PUB.SYS
STATUS:           Successful
USER:             JOHN            GROUP:          DOE
ACCOUNT:          PAYROLL         JSNAME:         JREPORT
```

The following is the ACD changes logging record format:



Logging of Stream Initiation

This feature enables the system manager to know who streams a job and when and where it occurs.

A new log type 139 will be added to the system logging facility. The MPE operating system and log file report generator, LOGTOOL will recognize this new log event. It can be enabled by SYSGEN followed by a START NORECOVERY. This event is initially disabled.

When this feature is enabled, any user streaming a job will have the event logged. The new log record will contain the following information:

Header:

- record type
- record length
- timestamp
- job/session number
- PIN

Log information:

- identification of the user who streamed the job (includes job/session name, user name, group name, account name)
- name of the job file that was streamed (includes file name, group name, account name)
- logical device number where the job was streamed
- identification assumed by the job (includes j/s number, user name, group name, account name)
- time that the job was scheduled to be launched
- name of the job streamed

For example, a user, with user name JOHN, account name PAYROLL, group name DOE, and job/session name JREPORT, streamed a job file JTEST.TESTGP.TESTACCT which logged on as #J12 QAMGR.QAACCT,QAGP with a job name as JOBTEST from ldev 21 and an input spool id #15. The job is scheduled to be launched at 3:00 pm, Monday, August 5, 1991.

LOGTOOL will format the following layout after the standard header:

```
INPUT LDEV:          21
JOB FILE NAME:       JTEST.TESTGP.TESTACCT
LOGON J/S TYPE:      JOB                  LOGON J/S NUMBER:   12
LOGON USER:          QAMGR                LOGON GROUP:        QAGP
LOGON ACCOUNT:       QAACCT               LOGON JOB NAME:     JOBTEST
INPUT SPOOLFILE ID:  15
SCHEDULE DATE:       MON, AUG  5, 1991    SCHEDULE TIME:      3:00 PM
USER:                JOHN                 GROUP:              DOE
ACCOUNT:             PAYROLL              JSNAME:             JREPORT
```

The following is the stream initiation logging record format:

length in word	bit 0 15
(1)	Record type = 139
(1)	Record length
(1)	PIN
(3)	Time stamp
(2)	Job type / job number
(1)	INPUT LDEV
(25)	JOB FILE NAME
(2)	JOB LOGON J/S NUMBER
(8)	JOB LOGON USER
(8)	JOB LOGON GROUP
(8)	JOB LOGON ACCOUNT
(8)	JOB NAME
(2)	INPUT SPOOLFILE ID
(1)	SCHEDULED DATE
(2)	SCHEDULED TIME
(8)	USER NAME
(8)	GROUP NAME
(8)	ACCOUNT NAME
(8)	JOB/SESSION NAME

Logging of User Logging

This feature enables the system manager to determine who accesses or tries to access the user logging facility by logging all OPENLOG and CLOSELOG intrinsic calls.

A new log type 140 will be added to the system logging facility. This new log record will be recognized by the MPE operating system and log file report generator LOGTOOL. It can be enabled by SYSGEN followed by a START NORECOVERY. This event is initially disabled.

When it is enabled, all OPENLOG and CLOSELOG intrinsic calls will be logged. The new log record will have the following information:

Header:

- record type
- record length
- timestamp
- job/session number
- PIN

Log information:

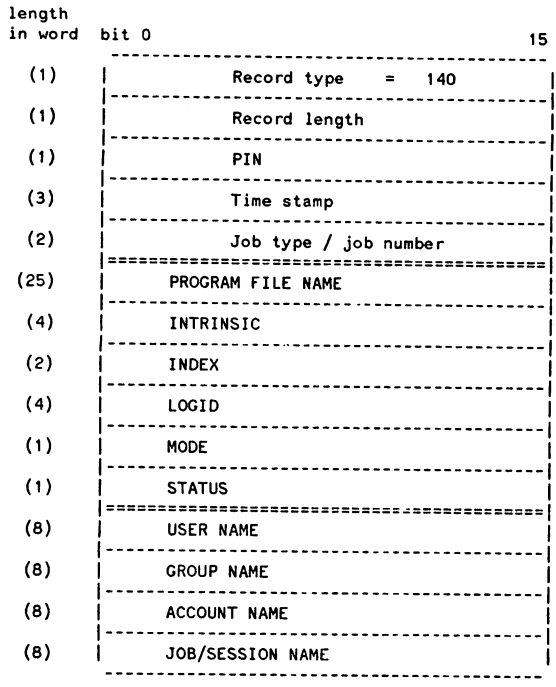
- identification of the user who called OPENLOG or CLOSELOG (includes job/session name, user name, group name, account name)
- name of the program file from where OPENLOG or CLOSELOG was called
- intrinsic that the user called, either OPENLOG or CLOSELOG
- logging index (its value is returned from OPENLOG. It identifies the access to the user logging facility)
- logging identification
- mode of the intrinsic call, either 0 for WAIT or 1 for NOWAIT
- status of the intrinsic call

For example, a user JOHN.PAYROLL,DOE with job/session name as JREPORT, ran a program FTEST.TESTGP.TESTACCT which called the OPENLOG intrinsic. The intrinsic call had index 1000, logid JOHNID, mode 0 and there was no error returned from the call.

LOGTOOL will format the following layout after the standard header:

PROGRAM FILE NAME:	*FTEST.TESTGP.TESTACCT*		
INTRINSIC:	*OPENLOG*	INDEX:	*1000*
LOG ID:	*JOHNID*	MODE:	*WAIT*
STATUS:	*0*		
USER:	*JOHN*	GROUP:	*DOE*
ACCOUNT:	*PAYROLL*	JSNAME:	*JREPORT*

The following is the user logging log record format:



NOTE

The LOG ID field in the log record will be "XXXXXX" for CLOSELOG intrinsic when the index is bad.

Logging of Process Creation

This feature provides information for tracing the creation of a process on the system.

A new log type 141 will be added to the system logging facility. It can be enabled by SYSGEN followed by a START NORECOVERY. This event is initially disabled.

When it is enabled, the system logging facility will log all process creations. The new log record will have the following information:

Header:

- record type
- record length
- timestamp
- job/session number
- PIN

Log information:

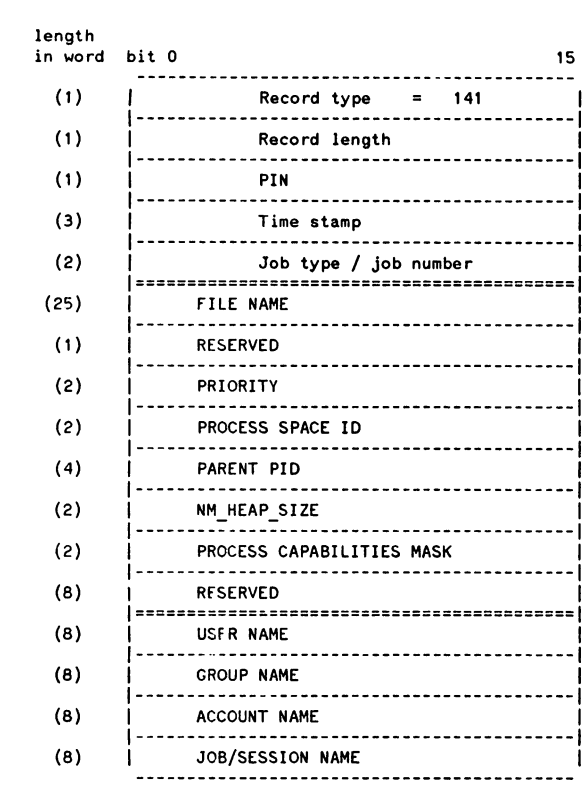
- identification of the user who initiated the process (includes job/session name, user name, group name, account name)
- process identification (PID) initiated
- parent PID
- priority of the process
- process space ID
- program name that the process was initiated from
- capabilities of the created process
- native mode heap size in bytes

For example, a user JOHN.PAYROLL,DOE with job/session name as JREPORT, ran a program FTEST.TESTGP.TESTACCT with PID 2300000002. The program initiated a process with PID 3300000001, priority 130 and space ID 556. The process created had capabilities of IA,BA,PH,PM and had NM heap size 200000 bytes.

LOGTOOL will format the following layout after the standard header:

```
PROGRAM FILE NAME:  FTEST.TESTGP.TESTACCT
PID INITIATED:      3300000001      PRIORITY:          130
SPACE ID:           556             PARENT PID:        2300000002
NM_HEAP_SIZE:       200000
PROCESS CAP BA:     YES             PROCESS CAP IA:    YES
PROCESS CAP PM:     YES             PROCESS CAP MR:    NO
PROCESS CAP DS:     NO              PROCESS CAP PH:    YES
USER:               JOHN            GROUP:             DOE
ACCOUNT:            PAYROLL         JSNAME:            JREPORT
```

The following is the process initiation log record format:



```
System logging mask:

User Attributes                    Program/Group Attributes
bit  capability                     bit   capability
 0  - SM                            23  -  BA
 1  - AM                            24  -  IA
 2  - AL                            25  -  PM
 3  - GL                            28  -  MR
 4  - DI                            30  -  DS
 5  - OP                            31  -  PH

File access attributes
 6  - CV
 7  - UV
 8  - LG
 9  - SP
10  - PS
11  - NA
12  - NM
13  - CS
14  - ND
15  - SF
```

Auditability by Named User

The "auditability by named user" feature will enable a system manager to selectively audit the actions of one or more users based on individual identity. To do this the LOGTOOL utility has been modified to select security relevant log records from the system log files. System managers define a named user by specifying (via the LIST command) a job/session name, user name, and/or account name. Security relevant events are then selected.

Three optional parameters, JSNAME, USER and ACCOUNT, have been added to the LIST command in LOGTOOL. The syntax will be:

```
LIST LOG=<log list>
     [;JSNAME=<job/session name>]
     [;USER=<user name>]
     [;ACCOUNT=<account name>]
```

The input should not be longer than 80 characters. The default value of these new parameters is @.

For example, to LIST log records from log files numbered 1 to 5, with log type 142 and user identification JTEST,JOHN.PAYROLL, the LIST command will be:

```
>LIST LOG=1/5;TYPE=142;JSNAME=JTEST;USER=JOHN;ACCOUNT=PAYROLL
```

To select log records from log files numbered 1 to 5, with log type 142 and user identification @,@.PAYROLL, the LIST command will be:

```
>LIST LOG=1/5;TYPE=142;ACCOUNT=PAYROLL
```

To allow LOGTOOL to select security relevant log records based on the user identification, all security relevant log records will contain user identification entries (JSNAME, USER, and ACCOUNT). Those security relevant log events are:

- job initiation (type 102)
- job termination (type 103)
- process termination (type 104)
- file close (type 105, 160)
- physical mount/dismount (type 112)
- logical mount/dismount (type 113)
- tape labels (type 114)
- console log (type 115)
- program file event (type 116)
- Native Mode Spooling (type 120)
- Architecture Interface (type 130)
- password change (type 134)
- system logging configuration (type 135)
- restore (type 136)
- printer access failure (type 137)
- ACD change (type 138)
- stream initiation (type 139)
- user logging access (type 140)
- process initiation (type 141)
- security monitor configuration change (type 142)
- change group (type 143)
- file open (type 144)
- command logging (type 145)

The following existing log records will be appended with user identification entries:

- job termination (type 103)
- process termination (type 104)
- physical mount/dismount (type 112)
- tape labels (type 114)
- console log (type 115)
- program file event (type 116)
- Native Mode Spooling (type 120)
- change group (type 143)

Paper # 3122

Serial Message Routing and Electronic Authorization

The Key to Work Flow Automation

Martin Hurren

Hewlett-Packard Company
Pinewood Info. Sys. Division
Nine Mile Ride
Wokingham
Berkshire
UK

Tel. 44-344-763526

There are three objectives for this presentation. First, to define and segment work flow automation highlighting the central role of serial message routing and electronic authorization; second, to present specific examples of work flow automation and the measurable benefits it can provide; and, third, to discuss Hewlett-Packard's implementation of work flow automation.

Office automation is often thought of as automating the secretarial task of creating documents and the accounting task of spreadsheet analysis. Measurable benefits, such as those provided by computer integrated manufacturing in decreasing the time to turn raw material into finished goods and lower the cost of manufacturing, have not been easy to establish in the office environment.

Work flow automation can be viewed as computer integrated manufacturing for the office. Most office tasks, such as getting a purchase requisition approved and into the hands of a purchasing agent or processing an insurance claim, can be thought of as repeatable processes with raw material, work in progress, production schedules and points of specific action.

Changing the focus of office automation from automating discrete tasks to automating processes can bring measurable increases in the benefits provided. In addition, the support for multiple media types (text, data, graphics and scanned images) greatly expands the number of processes which can be automated.

Work flow can be segmented along the dimensions of the complexity of the process and the frequency with which the process is repeated, or, more simply, the transaction volume.

The four major segments of work flow are the following:

- Routine work flow automation: This is the serial routing of work from one step to the next with authorization and/or acknowledgment required along the way. These are relatively low complexity tasks and the frequency can vary from low to high.

- Dynamic work flow automation: This is similar to routine work flow, but the processes being automated are more complex, typically change more frequently, and actions are more content and context dependent.

- Document creation: This covers relatively complex process which are not often repeated.

- High speed continuous: These are high volume process, such as check processing.

There are a number of major components that appear in work flow applications, but the amount and complexity of the procedures and routes are what distinguish routine from dynamic work flow applications.

The major components of work flow automation are:

- Work Units - processible "chunks" of works
- User Mailboxes - individual mail drops and rules for processing
- Routes - paths for work units and procedures to be invoked along the way
- Procedures - stored programs called and executed when work units enter mailboxes
- Queues - accumulation of processed work units in mailboxes at a point in time

Source: M. Howard, Gartner Group, Technology, T-260-730 May 29, 1990

Electronic mail systems today provide mail boxes, routing algorithms and user directories. These electronic mail components become the basic underpinning for work flow when they are extended to allow serial distribution and procedures, such as electronic authorization.

A fundamental decision when implementing work flow is whether to leverage an electronic mail system with corporate directories, world wide networking, and sophisticated administration or to implement a separate work flow system which must ultimately duplicate most of the features of electronic mail.

Nine different applications of work flow automation are presented below:

Routine Work Flow Automation

- Purchase requisitions
- Engineering change orders to manufacturing and service/support
- "Exception" reporting

Dynamic Work Flow Automation

- Insurance claims processing
- Medical records management

Document Creation

- Technical documentation
- Training guides

High Speed Continuous

- Check processing
- Credit card processing

Some examples, such as insurance claims processing, are industry specific, but most of the applications are relevant to business, government and education.

The benefits listed above from work flow automation are well defined and measurable. Lowered costs result from both simple changes, such as electronic forms allowing data to be entered only once whereas data entered onto paper forms must later be keyed into a system or transferred from one form to another, and more complex changes, such as being able to monitor the flow of work through a process, analyze bottlenecks and modifying the process to remove those bottlenecks.

The fact that units of work can be made "read only" and authorization is provided by secure passwords improves security. In addition, each step of a process can be logged by the system and reviewed, so no work can be "lost" or "buried in an in tray". These transaction logs can be retained for later auditing, if necessary.

The speed with which work, such as customer service requests, is completed can be the source of increased customer satisfaction and, ultimately, revenue. In addition, the use of electronic forms and preconfigured routing distribution lists allows much more rapid change to work processes. With paper based systems, the pace of change can be dictated by the speed with which new forms can be manufactured and distributed. This can effectively put the stationary department on the critical path when implementing change.

The value of any communications system is largely a function of the number of people it links together. Hewlett-Packard's approach, therefore, is to build an electronic messaging infrastructure which supports users across a wide variety of workstation types and both HP and non-HP systems.

To achieve this multi-vendor infrastructure, HP provides its electronic messaging components on non-HP hardware. This infrastructure is built upon a client/server architecture and industry standards, such as X.400 and MicroSoft Windows, to provide the broadest possible reach and the tightest possible integration with existing applications.

Routine work flow automation is achieved by extending this electronic messaging infrastructure through the addition of serial message routing and electronic authorization. This approach has been called "surfing on standards", and it is one of the ways in which HP is adding value beyond the standards bodies' definitions.

HP's electronic messaging infrastructure is an integral component of our office systems product line, NewWave Office. NewWave Office covers both routine work flow as described above and also dynamic work flow with its document image management component. All NewWave Office components conform to the same client/server and standards based architecture, so a single, consistent approach to mailboxes, directory management and routing mechanisms can ultimately be shared between NewWave Office solutions for routine and dynamic work flow automation.

The serial message routing and electronic authorization extensions to HP's NewWave Office will begin shipping at the beginning of '92.

As the computer integrated manufacturing system for the office, work flow automation provides clear and measurable benefits. These include:

Lower Costs

- Eliminate printing/stocking/scrapping paper based forms
- Eliminate paper mail, paper storage & retrieval
- Leverage e-mail directory management & administration

Increase Security and Reliability

- Eliminate tampering
- Increase confidentiality
- Auditable transaction logging

Increase Customer Satisfaction and Revenue

- Reduce time to action
- Flexibility for rapid change
- Closed reliability

There is a broad range of potential applications of work flow automation in the office, but underpinning all of these applications are basic electronic messaging components, such as mail boxes, directories, and message routing.

Routine work flow automation is created by extending standard electronic mail applications with serial message routing and procedures, such as electronic authorization. Dynamic work flow automation builds upon this infrastructure and allows more complex, changeable and user driven processes to be automated.

Hewlett-Packard is at work extending its NewWave Office multi-platform, multi-vendor electronic messaging products to provide routine work flow automation. This development is coordinated with the NewWave Office document image management system to enable dynamic workflow automation. HP's approach of "surfing on standards" and support for both HP and non-HP hardware allows it to provide robust messaging and workflow solutions in customers' often complex multi-vendor environments.

Client/Server Application Development Tools

John Yu/Henry Lieu
Hewlett-Packard Company
Information Networks Division
MS 43LS
19420 Homestead Road
Cupertino, CA 95014

Paper Number: 3123

1. Introduction

As the trend in business moves towards decentralized operations to increase its competitiveness and to better serve its customers' needs, the computing resources have to be distributed to allow for increased communication in this diverse environment. There are two emerging trends for distributed computing. The first one is **Enterprisewide** computing which implies that all the computers in a company are connected into network, with overall management accomplished by a mainframe. The second one is **Client/Server** computing which divides tasks between clients (usually personal computers and workstations) and servers (often mini-computers and mainframes). The client/server computing, in particular, has been gaining momentum to become THE architecture to provide resource (data, peripheral devices, etc.) sharing in a distributed computing environment. The following statements are quoted from market surveys published by many reputable industry groups:

- **Sierra Group April 1990:** Client/server computing is a fixture in many large corporations. The study found that 64 percent of Fortune 500 companies that responded are implementing client/server solutions; the remaining companies expect to do so within one year.
- **Business Research Group October 1990:** Of the 750 Fortune-1000 firms surveyed, two-thrids are currently using a client/server architecture, or will do so within next 18 months.
- **Yankee Group Janurary 1991:** A survey of Fortune 1000 information systems departments reveals that 69 percent of the respondents either have client/server systems or plan to acquire them in the near future. Some 85 percent of those firms use or plan to use client/server systems for mission-critical applications.

Although more and more companies are adopting the client/server architecture in their distributed computing environment, this architecture is currently being implemented mainly to provide peripheral sharing (such as printer sharing and file sharing) and limited database sharing only. The complexity and lack of development tools has hindered the widespread development of client/server applications. The lack of a set of easy-to-use tools which enables the users to develop their distributed applications in a productive way for information sharing through transparent data access across a multi-application, multi-network, multi-platform, and multi-vendor environment, has prevented users from getting the full benefit of resource sharing in a distributed computing environment.

Today, software developers would face the following problems while developing a distributed application:

- User's programs are not network-transparent. Developers need to re-write their programs whenever a new network transport protocol support is required.
- Developers need to choose the right network Application Programmatic Interfaces (APIs) that meet their needs. Understanding the complexity of network APIs and choosing the right ones are technical challenge to the application developers.

- Developers need to learn a new network transport interface whenever the underlying transport is changed. Developers also need to learn new programming languages and development tools in order to write applications on some of the new network transports.
- Developers need to be familiar with the storage allocation scheme associated with a structured data to be exchanged between the client and the server programs developed in different languages or different language implementations, and executed on different platforms.

The upper part of Figure 1 illustrates the complexity of developing and maintaining a client/server application which supports multiple network transports. The lower part of Figure 1 shows the diversity of a client/server application which uses different network transport Application Programmatic Interfaces (APIs) through different calling conventions and different calling sequences.

Figure 2 examplifies a case where the memory allocation schemes and data presentation methods vary considerably even when programs are written in the same programming language and executed in a single-vendor environment.

1.1 What is NHLL

NHLL (Network High Level Language) is a programming toolkit containing facilities required to aid programmers in developing client/server-based distributed applications. It aims at providing network transparency to 3GLs (3rd Generation programming Languages such as Cobol, C, Fortran, and Pascal) users. Using NHLL, the 3GL users can concentrate on their application program development and let NHLL handle the complexity of different types of underlying networking transport protocol. Programs using NHLL can be moved from one networking protocols to another without re-compiling through NHLL's capability of linking the correct transport protocol at run-time by NHLL run-time network configuration.

The key components of NHLL are summarized below:

- **NHLL Embedded Statements** consist of six data transfer statements and three data conversion statements. They are simple to understand and easy to use. The syntax of these statements is language independent, and networking transport independent as well.
- **NHLL Data Conversion Services** are built into NHLL to help developers resolve data representation differences between client and server running on different platforms or using different languages. This is a very critical feature required whenever data is exchanged in form of structure (not simple byte stream) between client and server.
- **NHLL Run-Time Library** maps NHLL statements to network transports and resolves "service" to "network address" translation at run time. It also provides six exported C subroutines for developers to write programs directly accessing to NHLL services without using language pre-processor.
- **NHLL Network Configuration** provides "service-to-network address" translation.
- **NHLL Supportability Tools/Functions** are additional features provided by NHLL in areas of network tracing and logging, and server management among others.

The upper part of Figure 3 illustrates how NHLL can be used to develop complex client/server applications in an easy and structured way. It demonstrates that multiple NHLL server programs can support multiple NHLL clients over different transports concurrently.

2. NHLL Embedded Statement

NHLL Embedded Statements are language-independent verbs which can be included in programs written in conventional languages. NHLL verb definitions are simple to understand and easy to use. Users don't have to re-learn NHLL when they switch from one programming language to another. NHLL statements are also network independent and they run on the top of various industry standard transport APIs (such as NamedPipe, Novell SPX, BSD Socket, and LU 6.2). This allows user to switch from one network transport to another without changing the application and to support additional new transports without modifying source code. NHLL also provides optional data conversion services so that applications written in different languages on different machines or same language but different implementations (e.g. MicroFocus Cobol on PC and Hewlett-Packard Cobol on HP 3000) can send and receive structured data in addition to simple byte streams. It makes distributed applications developed in NHLL highly portable to different vendors' platforms.

All NHLL Embedded Statements start with "EXEC NHLL" and end with "END-EXEC". The following are syntax rules for the NHLL embedded statements:

EXEC NHLL LISTEN ON *:service_name* RETURN (*:status*) END-EXEC.

EXEC NHLL TALK TO *:service_name* RETURN (*:status*) END-EXEC.

EXEC NHLL RECEIVE VALUES (*:msg* [, *:len*]) [NOWAIT] RETURN (*:status*[, *:msgid*]) END-EXEC.

EXEC NHLL SEND VALUES (*:msg* [, [*:len*][, *:msgid*]]) RETURN (*:status*) END-EXEC.

EXEC NHLL CLOSE [ALL] RETURN (*:status*) END-EXEC.

EXEC NHLL CONTEXT TO *:service_name* RETURN (*:status*) END-EXEC.

EXEC NHLL COMPILER IS *:language_vendor__name* END-EXEC.

EXEC NHLL CONVERSION {NONE | SAME LANGUAGE | DIFFERENT LANGUAGE} END-EXEC.

EXEC NHLL {BEGIN | END} DECLARE SECTION END-EXEC.

The lower part of Figure 3 depicts the client/server communication model using NHLL embedded statements (data transfer verbs).

2.1 LISTEN ON Statement

SYNTAX

EXEC NHLL LISTEN ON *:service_name* RETURN (*:status*) END-EXEC.

DESCRIPTION

LISTEN ON is callable by the server only. The call is to associate the server application with an indicated **service_name** and thus makes itself available to serve the requests from clients. No activity can occur between the server and clients unless LISTEN ON has been run successfully at least once.

Even though the server can run on the top of multiple transports simultaneously, only one LISTEN ON call is needed. LISTEN ON will translate an indicated 'service_name' into one

or multiple network/transport addresses according to NHLL configuration set up by the NHLL network administrator. Once the call returns successfully, the server should be able to receive requests from clients over any of supported transport types. In case of multiple transport supports, LISTEN ON is considered a failure only when the server fails to listen over ALL of the transport supports as specified in the configuration.

2.2 TALK TO Statement

SYNTAX

EXEC NHLL TALK TO *:service_name* RETURN (*:status*) END-EXEC.

DESCRIPTION

TALK TO is used by the client application only to establish its connection to a server that provides the services as indicated by the *service_name*. There may be more than one transport address associated with an indicated service_name. TALK TO may use one as a primary route and the rest as alternative routes. No request/reply message a client can send/receive to/from a server until it establishes its connection to a server by calling TALK TO successfully. A client may connect itself to multiple servers concurrently by issuing a series of TALK TO calls with different service names.

2.3 RECEIVE Statement

SYNTAX

EXEC NHLL RECEIVE VALUES (*:msg* [, *:len*]) [NOWAIT] RETURN(*:status*[, *:msgid*]) END-EXEC.

DESCRIPTION

RECEIVE can be called by both client and server applications. Server uses this call to receive any request message from any connected client. Client uses this call to receive a reply message from a server which is currently communicating with.

RECEIVE can run in either **WAIT** or **NOWAIT** mode. In a WAIT mode, RECEIVE won't return until a message is available to be received. However, in a NOWAIT mode, RECEIVE will return immediately with NO_MESAGE status when there is no message available for the caller to receive. If NOWAIT is omitted in the NHLL embedded statement, it indicates that RECEIVE is in a WAIT mode. Otherwise, RECEIVE operates in a NOWAIT mode.

If the optional parameter *len* is present, then the data being received is treated as a byte stream and the user controls the amount of data received and its interpretation. If the optional parameter *len* is missing, then as far as the user is concerned, a data structure is being received. The data specification verbs, together with the declaration of the parameter *msg* in the NHLL declaration section, determine conversions performed.

msgid is used by a client to correlate a reply message to an outstanding request message. It is used by a server in replying to a received request message. However, *Msgid* is optional. In the absence of *Msgid*, a request and a reply message are correlated by subsequent SEND and RECEIVE calls.

If the caller-provided buffer is smaller than the whole message's length, RECEIVE will fill up the receiving buffer and return **MORE_DATA** as a status code. And subsequent RECEIVE

call(s) will receive the remaining portion of message.

For a server, RECEIVE can receive a message from any connected client over any transport type. For a client who maintains connections with multiple servers, it should be aware of which server it is currently having the CONTEXT with. If the client chooses to receive message from another server, it has to call CONTEXT TO before RECEIVE.

2.4 SEND Statement

SYNTAX

EXEC NHLL SEND VALUES (*:msg* [, [*:len*] [,*:msgid*]]) RETURN (:status) END-EXEC.

DESCRIPTION

SEND can be called by both client and server applications. Server uses this call to send a reply message to a client. Client uses this call to send a request message to a server.

SEND is in **WAIT** and **message** mode. It won't return the control to the caller until a whole message has been sent out throught the network transport successfully or an error has been encountered.

If the optional parameter *len* is present, then the data being sent is treated as a byte stream and the user controls the amount of data received and its interpretation. If the optional parameter *len* is missing, then as far as the user is concerned, a data structure is being sent. The data specification verbs, together with the declaration of the parameter *msg* in the NHLL declaration section, determine the conversions performed.

The client can choose an integral number ranging from 1 to (2**16-1) as a message identifier. Once the client received a reply message from a server, it can then use the message identifier obtained in RECEIVE to corelate it to an outstanding request. The usage of *msgid* is OPTIONAL. Its usage is very useful especially when a client has multiple outstanding requests at a time.

If a client who maintains connections with multiple servers, it should be aware of which server it is currently having the CONTEXT with. If the client chooses to send a message to another server, it has to call CONTEXT TO before SEND.

2.5 CLOSE Statement

SYNTAX

EXEC NHLL CLOSE [ALL] RETURN (*:status*) END-EXEC.

DESCRIPTION

CLOSE can be called by both client and server. If the server calls CLOSE with ALL option, all the connections to clients over all supported transport types will be closed down. Afterward, a server can no longer serve clients until LISTEN ON is called successfully again. If the server calls CLOSE without the ALL option, then only the connection to a client the server just finishes communicating with will be disconnected.

If the client calls CLOSE with ALL option, all the connections established by the client will

be disconnected. If the client calls CLOSE without the ALL option, it will only disconnect the connection with a server which it is currently having a context with.

2.6 CONTEXT TO Statement

SYNTAX

EXEC NHLL CONTEXT TO *:service_name* RETURN (*:status*) END-EXEC.

DESCRIPTION

CONTEXT TO is only callable by the client while it is maintaining multiple connections. The client should invoke CONTEXT TO to switch to a desired server that it would like to communicate with next. If CONTEXT fails, it won't destroy the current context.

2.7 COMPILER IS Statement

SYNTAX

EXEC NHLL COMPILER IS *language_vendor_name* END-EXEC.

DESCRIPTION

COMPILER is one of the commands used for data conversion support. The memory allocation on a given machine for a particular language depends on the compiler. For instance, on an IBM-PC or its compatible machine, a MicroFocus Cobol compiler may have different memory allocation for a structure than a Ryan MacFarland Cobol compiler.

The *Language_vendor_name* must be chosen from a list of specified names, which are platform dependent. If this command is not specified, then a default *language_vendor_name* is selected. The default supplier_names are dependent on the platforms and languages supported by NHLL.

2.8 CONVERSION Statement

SYNTAX

EXEC NHLL CONVERSION {NONE | SAME LANGUAGE | DIFFERENT LANGUAGE} END-EXEC.

DESCRIPTION

CONVERSION is one of the commands used for data conversion support. Three options are provided to make an optimal compromise between data transfer needs and the overhead imposed as a result. This command must occur before the BEGIN DECLARE SECTION command. If not specified, the option **NONE** is assumed.

The representation of a data structure varies with the type of system (hardware and OS), the language in which the data structure is defined, the compiler being used, and the compiler options being used. Figure 2 illustrates the differences in data alignment, byte-ordering, and storage allocation of the same data structure on three different platforms.

- Some examples of system related differences are - An IBM machine will uses EBCDIC code for character representation, whereas a HP machine uses ASCII code for character representation. A PC has a different byte ordering scheme for an integer than a HP or SUN machine. The sizes of integers typically vary by machine type. A DEC machine based on MIPs architecture will have a different byte alignment than a SGI machine based on the same architecture.
- Some examples of language related differences are - In C language, a character string is null terminated whereas in Cobol, a string is blank filled. MicroSoft C on PC has different byte ordering than MicroFocus Cobol on the PC. Cobol can represent an integer in many different ways depending on the USAGE clause. Of course, data structure layout is completely different.
- Some examples of compiler and compiler related differences are - Using MicroSoft C on the PC, compiler options can be specified (large, small, medium model etc.), that will impact the layout of a data structure. Two different compilers from different vendors on the PC (e.g. MicroSoft C vs. Borland Turbo C), or any other hardware are quite likely to generate different data alignment. MicroFocus Cobol compiler on PC or any other machine will do data alignment, and even byte ordering, differently depending on Compiler options chosen.

In a client/server environment, it is highly likely that the pieces of the product will be running on different systems. It is also likely, that the language being used will be different, e.g., writing the client program in C and writing server program in Cobol. In this environment, the developer faces the task of first figuring out what the differences are, and then, developing codes to eliminate the differences. NHLL gives the developer the ability to eliminate this problem without any intervention on behalf of the developer.

Data Conversion impacts only those structures being sent or received for which the optional parameter *len* is not specified in the SEND and RECEIVE commands. Also all such data structures must be declared in the NHLL declaration section.

The option **NONE** is typically used in cases where client and the server applications have been developed in the same language using the same compiler, and are running on the same type of systems. In this case, NHLL performs no data conversion.

The option **SAME LANGUAGE** is typically used in cases where the client and the server application have been developed in the same language, but using different compilers from different vendors, or running on different types of machines or both. In this case, the declaration of the data structures should be the same on both the client and the server. NHLL provides data conversion services required to resolve the differences of storage allocation for structured messages and data representation of the messages to be exchanged between the client system and the server system.

The option **DIFFERENT LANGUAGE** is typically used in cases where the client and the server application have been developed in different languages and may be running on different types of machines. In this case, the declaration of the data structures must be semantically the same on both the client and the server. NHLL provides data conversion services required to resolve the differences of memory allocation for structured messages and data representaton of the messages to be exchanged between the client program and the server program written in different languages and executed on different platforms.

2.9 DECLARE Statement

SYNTAX

EXEC NHLL {BEGIN | END} DECLARE SECTION END-EXEC.

DESCRIPTION

BEGIN DECLARE SECTION starts the NHLL declaration section and END DECLARE SECTION ends the NHLL declaration section. All *msg* variables used for data transfer in NHLL SEND and RECEIVE statements must be declared in the NHLL declaration section unless the optional parameter *len* is used for simple byte stream transfer.

The purpose of declaring the *msg* variables for the SEND and RECEIVE commands in the declaration section is to allow the generation of data definition structures and inline conversion codes for those structures for which data conversion is desired. NHLL will add data declarations in the NHLL declaration section. It will also generate code in either procedures or paragraphs to support the data conversion for the structure declared and used later. Also inline code generated will be dependent on the data structure and the type of data conversion selected.

3. NHLL Client/Server Sample Programs

There are two NHLL sample programs included here to help developers realize how easy it is to use NHLL to develop client/server application programs. The first one is a NHLL client program written in C and second one is a NHLL server program written in Cobol. The client program accepts a database update request (add, modify, or delete) from the user and sends it to the server for processing. The server receives the request, calls a database update routine, and returns the request status back to the requesting client.

Sample NHLL Client Program Written in C

```
/* This is a NHLL/C source program which adds/deletes/modifies an employee
   database record based on the user's request.                          */

EXEC NHLL CONVERSION DIFFERENT  LANGUAGE END-EXEC.

long nhll_status;  /* variable for nhll status return */
char service_name[] = "employee"; /* string for service name */

EXEC NHLL BEGIN DECLARE SECTION END-EXEC.
struct {
   char request_type;
   int employee_number;
   char employee_name[20];
   int request_status;
} emp_record;
EXEC NHLL END DECLARE SECTION END-EXEC.

main()
{
/* talk to the server by specifying the service name.   */

   EXEC NHLL TALK TO :service_name RETURN (:nhll_status) END-EXEC.
   if (nhll_status != 0) exit(1);

/* Wait for user's request Add/Modify/Delete employee database record     */

   while (1) {
      get_database_request(&emp_record);
      if (emp_record.request_type == 'E') close_conn(0);

/* Send request to the server for database update processing             */

      EXEC NHLL SEND VALUES (:emp_record) RETURN (:nhll_status) END-EXEC.
      if (nhll_status != 0) close_conn(1);

/* Wait for response from the server                                     */

      EXEC NHLL RECEIVE VALUES (:emp_record) RETURN (:nhll_status) END-EXEC.
      if (nhll_status != 0) close_conn(1);
      display_database_response(&emp_record);
   }
}
close_conn(ret_code)
int ret_code;
{
/* close the connection */

   EXEC NHLL CLOSE RETURN (:nhll_status) END-EXEC.
   exit(ret_code);
}
```

```
001800 IDENTIFICATION DIVISION.
002000 PROGRAM-ID. SERVER.
002200
002300 AUTHOR.  John Yu.
002500 DATE-COMPILED. APRIL 30, 1991.
003000
003100 REMARKS.
003102         This program is one of the 2 programs which demonstrates
003104         the functionality and easy of use of the NHLL (Network
003105         High-Level Language).  This program accepts request from
003105         the client and processes the employee database update
003108         request. Once done, it then sends request status back to
003109         the requesting client.
004300
004400 ENVIRONMENT DIVISION.
004800
006500 DATA DIVISION.
006600 FILE SECTION.
006900
011000 WORKING-STORAGE SECTION.
012110
012111 01 EMP-RECORD.
012112    02  REQUEST-TYPE      PIC  X(01)    VALUE SPACES.
012112    02  EMPLOYEE-NUMBER   PIC  S9(9)    SYNC.
012171    02  EMPLOYEE-NAME     PIC  X(20)    VALUE SPACES.
012171    02  REQUEST-STATUS    PIC  S9(9)    SYNC.
012171 77 NCode                 PIC  S9(9)    SYNC  VALUE 0.
012800
012900 PROCEDURE DIVISION.
013550 BEGIN SECTION.
013600 0000-MAIN-CONTROL.
013700
014810 EXEC NHLL LISTEN ON 'employee' RETURN (:NCode) END-EXEC.
014840     IF NCode NOT = 0 THEN  GO TO CLOSE-CONN.
014842
015101 0000-MAIN-LOOP.
015107
015108 EXEC NHLL RECEIVE VALUES (:EMP-RECORD) RETURN (:NCode) END-EXEC.
015109     IF NCode NOT = 0 THEN GO TO CLOSE-CONN.
015114     CALL "database-update" using EMP-RECORD.
015115 EXEC NHLL SEND VALUES (:EMP-RECORD) RETURN (:NCode) END-EXEC.
015116     IF NCode NOT = 0 THEN GO TO CLOSE-CONN.
015117     GO TO 0000-MAIN-LOOP.
014880
014881 CLOSE-CONN.
014882 EXEC NHLL CLOSE END-EXEC.
014883     STOP RUN.
```

4. NHLL Run-Time Library

NHLL Run-Time Library provides a set of C routines which are directly callable from programs written in C language or any other languages supporting C subroutine calling mechanism. It consists of six exported C library routines as following:

Run-Time Library Routine	Corresponding Embedded Statement
nhll_listen	LISTEN ON
nhll_talk	TALK TO
nhll_receive	RECEIVE
nhll_send	SEND
nhll_close	CLOSE
nhll_context	CONTEXT TO

The functional description for these six routines are almost identical to their counterparts of NHLL Embedded Statements. The following syntax rules specify the parametric interface when calling NHLL run-time library routines:

SYNTAX

```
void  nhll_listen(service_name, status)
  char *service_name;
  long *status;

void nhll_talk(service_name, status)
  char *service_name;
  long *status;

void nhll_receive(msg, len, msgid, time_out, status)
  char  *msg;
  long  *len;
  long  *msgid;
  long  *time_out;
  long  *status;

void nhll_send(msg, len, msgid, flag, status)
  char   *msg;
  long   *len;
  long   *msgid;
  long   *flag;
  long   *status;

void nhll_close(status)
  long  *status;

void nhll_context(service_name, status)
  char  *service_name;
  long  *status;
```

IMPORTANT NOTE: Software developers who decide to use this run-time library should be aware that it only provides NHLL data transfer services and it is only callable from programs written in C or any other languages supporting C subroutine calling mechanism. Applications which either are written in languages without calling mechanism to access C subroutines, or require NHLL data conversion services should use NHLL embedded statements to obtain the

complete NHLL services. There is another NHLL run-time library which contains all the routines required for data conversion services. These routines, however, are only callable by programs which have been pre-processed by NHLL. Since application programs are shielded from the complexity of the NHLL data conversion routines, this particular library will not be discussed in this paper.

5. NHLL Network Configuration

NHLL itself doesn't provide network transport mechanism. Instead, it uses industry standard APIs (Application Programmatic Interfaces) to access to various commonly used types of transport. To provide network transparency to NHLL application developers and their end-users, a configuration file is required to allow NHLL to establish a connection between the client and the server for subsequent data exchange over the configured transports. NHLL will search for a file named **nhllcnf** first in the current working file directory and then in the NHLL default file directory. The first file matches will be used by NHLL to map the specified *service_name* to its associated *network addresses*.

NHLL configuration information should be stored in **nhllcnf** in the following format:

**service_name_1*
Transport_type/Transport_attribute_1/.../Transport_attribute_N
Transport_type/Transport_attribute_1/.../Transport_attribute_M
.
Transport_type/Transport_attribute_1/.../Transport_attribute_P
**service_name_N*
Transport_type/Transport_attribute_1/.../Transport_attribute_Q
.
Transport_type/Transport_attribute_1/.../Transport_attribute_Z

Thers is one ***service_name** entry occupying one line for each service to LISTEN ON or TALK TO. There should be one or more transport configuration items directly under the associated service name entry. Each transport configuration item occupies one line and consists of multiple attributes separated by the character '/'. The number of attrubites varies depending on the types of transport specified. The basic format of transport configuration item for TCP/BSD, HP NS, and LM is described as below:

TRANSPORT_TYPE/HOST_NAME/TRANSPORT_ADDRESS

There is a list of NHLL-supported transport types. The TRANSPORT_TYPE attribute must be spelled exactly as NHLL expects:

TCP ..... for TCP/BSD Sockets
LM ..... for LanManager/NAMED-PIPES
NS ..... for HP NS/3000 and NS/9000 NetIPC
SPX ..... for Netware or Portable Netware
etc.

TRANSPORT_ADDRESS means PORT_ADDRESS in the case of TCP/BSD; means NetIPC socket name in the case of HP NS; and means PIPE name in the case of LanManager.

For example: supposedly there are two server applications. One handles service named "sales" which resides on system "nodeA" and can serve its clients via both TCP/BSD with PORT_ADDRESS = "X90FA" in hex format and LM with PIPE name = "pipea". Another handles services named "inventory" which resides on system "nodeB" and can serve its clients only via

TCP/BSD with PORT_ADDRESS = "12345" in decimal format. Then you need to prepare the **nhllcnf** file which looks like:

```
*SALES
TCP/nodeA/X90FA
LM/nodeA/pipea
*Inventory
TCP/nodeB/12345
```

NHLL accesses to **nhllcnf** only when LISTEN ON or TALK TO is invoked. Any changes to **nhllcnf** will not take effect until the next LISTEN ON or TALK TO is called. LISTEN ON will try to establish ALL the connections associated with desired transports as configured in **nhllcnf** under the specified service name. TALK TO, however, will only establish one connection to the server and it is the first transport which establishes a successful connection will be used by TALK TO.

6. NHLL Supportability Tools/Functions

NHLL provides various tools and built-in functions to facilitate users in developing and managing client/server-based distributed applications. These tools and functions can be divided into two groups, namely, application development support and application management support.

6.1 NHLL Application Development Support Tools/Functions

The following NHLL application development support tools/functions can be used to facilitate users in debugging and testing their programs during the application development stage.

NHLL Procedure Tracing: NHLL provides a procedure tracing mechanism which allows software developers to capture critical NHLL-specific information in a trace file for debugging purpose.

NHLL Loopback Testing: Many types of network transport NHLL supports have the ability to allow the communication between a client and a server which are running on the same machine. This feature gives the application developers a chance to test the complete application on a single development machine before deploying the client program out onto the remote machine. (Note: The development and testing machine must have multi-processing capability in order to execute both client and server programs together on the same machine.)

6.2 NHLL Application Management Support Toos/Functions

The following NHLL application management support tools/functions can be used to facilitate users in troubleshooting and server monitoring when distributed application gets deployed in a production mode.

NHLL Event/Error Logging: NHLL provides an event/error logging mechanism which allows NHLL network administrators and users to capture critical NHLL-specific events and errors in a log file for troubleshooting purpose.

NHLL Server Monitor: NHLL provides a utility which allows NHLL network administrator to monitor the status of NHLL servers and associated network transports running on the local system.

7. Peer-to-Peer Communication

Since NHLL architecture is based on the client/server model with a hierarchical nature, it doesn't provide peer-to-peer communication (such as client-to-client and server-to-server) support

directly. However, by adding a thin layer of application-specific handshaking protocols, developers would be able to build the peer-to-peer communication capability on top of NHLL. The key to implementing peer-to-peer communication over NHLL is that the server needs to have a table to keep track of which client has signed on and what its NHLL network identifier is. (Note: The NHLL network identifier is stored as part of the *msgid* which is only known to the server.) Developers can design the client/server programs in such a way that the first message a client sends to the server is always its signon identifier, e.g. user name. The server can record the client's signon identifier along with its associated NHLL network identifier which can be extracted from the received *msgid*.

Client-to-Client Communication: The upper part of Figure 4 illustrates how client-to-client communication can be implemented using a pass-through NHLL server.

Server-to-server Communication: The lower part of Figure 4 depicts how server-to-server communication can be implemented using a pass-through NHLL client. The example used in the case is a typical service distribution architecture. The front-end server to which all regular clients connect acts as a service dispatcher. It determines which back-end server has the ability to handle the client's request and sends/receives requests/responses to/from a pass-through client associated with the requested back-end server. The following logic flow describes how this implementation works:

- All the back-end servers are up and running. All their associated pass-through clients are up and running too. Each pass-through client TALK TO its associated server then TALK TO the front-end server. Each pass-through client sends a "signon" message to the front-end server identifying itself as a pass-through client along with its associated service name.
- The front-end server receives signon messages from its clients. If a signon message is from a pass-through client (not a regualr client), then it understands that a particular back-end server is ready to serve.
- When the front-end server receives a service request from a regular client, it looks up the signon table to see if the requested server is ready. It then sends the request to the pass-through client associated with the desired back-end server by filling this client's NHLL network identifier in the previously received *msgid*.
- When the pass-through client receives a request, it calls CONTEXT TO verb to switch its connection to the associated back-end server and relays the request to the back-end server.
- When the pass-through client receives response from the back-end server, it calls CONTEXT TO verb to switch its connection back to the front-end server and relays the response to the front-end server. The front-end server can then send the response to the requesting (regular) client.

8. Summary

Computing in the 1990s will require distributed applications which will allow for increased communication in a diverse environment. Choosing the right tools to develop and integrate distributed applications is pivotal to the success of users' business. NHLL provides data transfer services and data conversion services and productivity improvement tools so that application developers can focus on how to deal with the complexity of the business applications rather than worrying about how to solve the networking and data conversion problems. Using NHLL, the distributed application developers can obtain benefits in the following areas:

- Major reduction in network-based software development time and cost.
    - o Hardware and language differences between client and server are done by NHLL
    - o Lower maintenance cost with one version of software

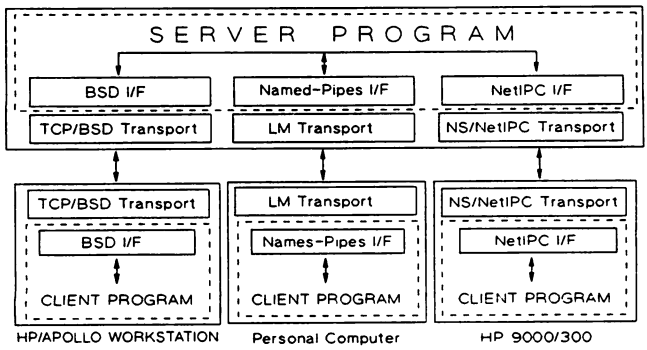
- Frees up networking developers to work on other product development
- Supports developers in the programming languages and tools they are familiar with

- Less programming training required for client/server implementations
  - No training on networking technologies
  - Minimal training for use of NHLL statements
  - Same set of verbs for all programming languages
  - Network protocol transparency supported by a single interface

- Leveraged investment of software development
  - Multiple-vendor platform portability
  - Multiple network transport support with one version of software
  - Faster time to market with new products and additonal transports

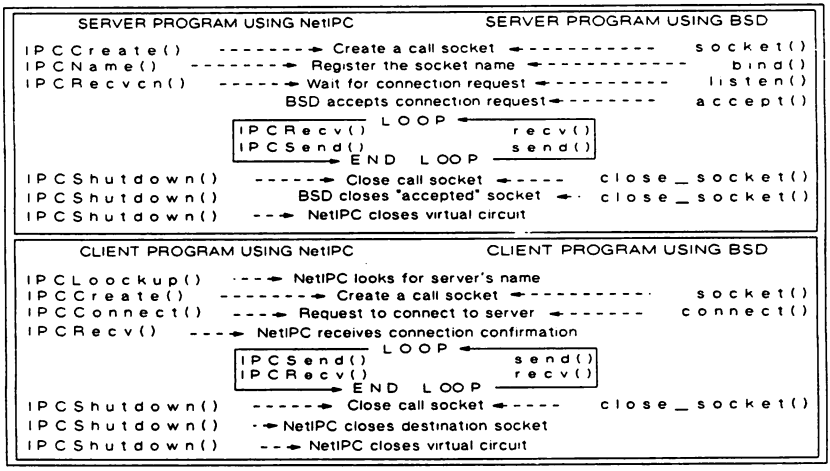
The solution providers, application developers, and system integrators can use NHLL to gain the benefits with today's technologies and will also grow into solutions for tomorrow.

Dependency On Network Transport Type



X number of transport interfaces to develop/maintain on the server side
X versions of client program to develop/maintain on the client side

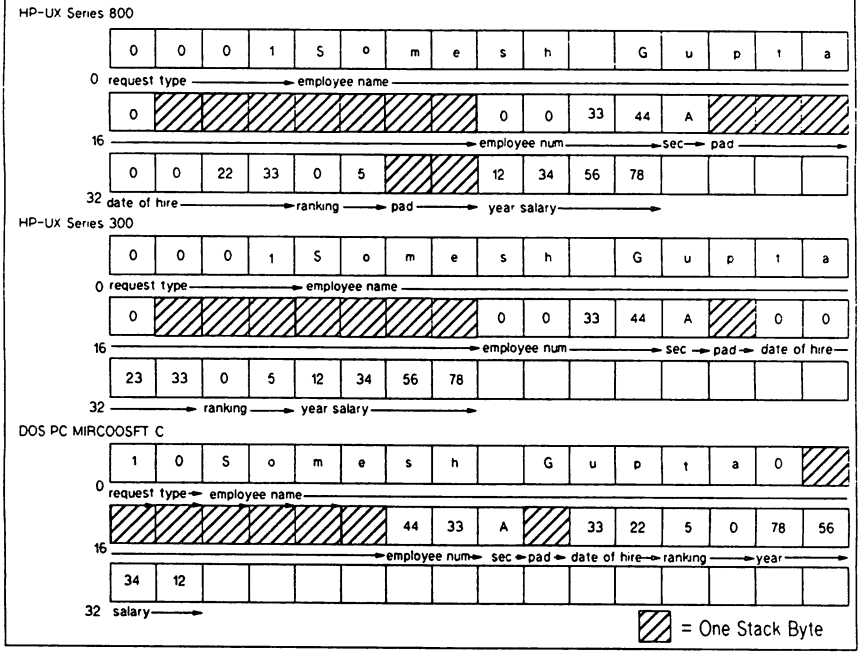
Dependency On Network Transport Interfaces



Network High-Level Language FIGURE - 1

MEMORY ALLOCATION AND DATA PRESENTATION ON THREE DIFFERENT PLATFORMS

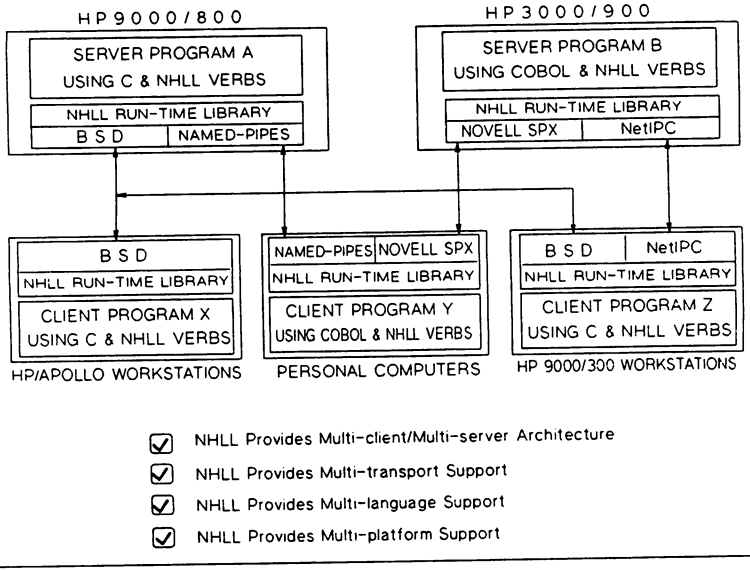
```
struct {
    unsigned int request_type;          Value = 0x1;
    char employee_name[20];             Value = "Somesh Gupta";
    unsigned int employee_num;          Value = 0x3344;
    char sec;                           Value = 'A';
    unsigned int date_of_hire;          Value = 0x2233;
    short ranking;                      Value = 5;
    unsigned long year_salary;          Value = 0x12345678;
} emp_record;
```



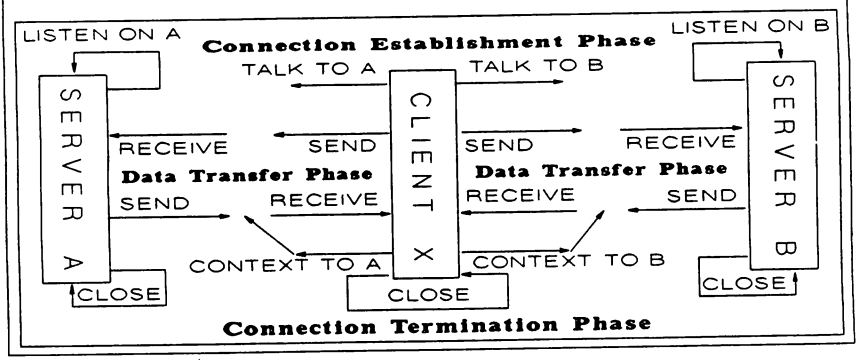
Network High-Level Language

FIGURE - 2

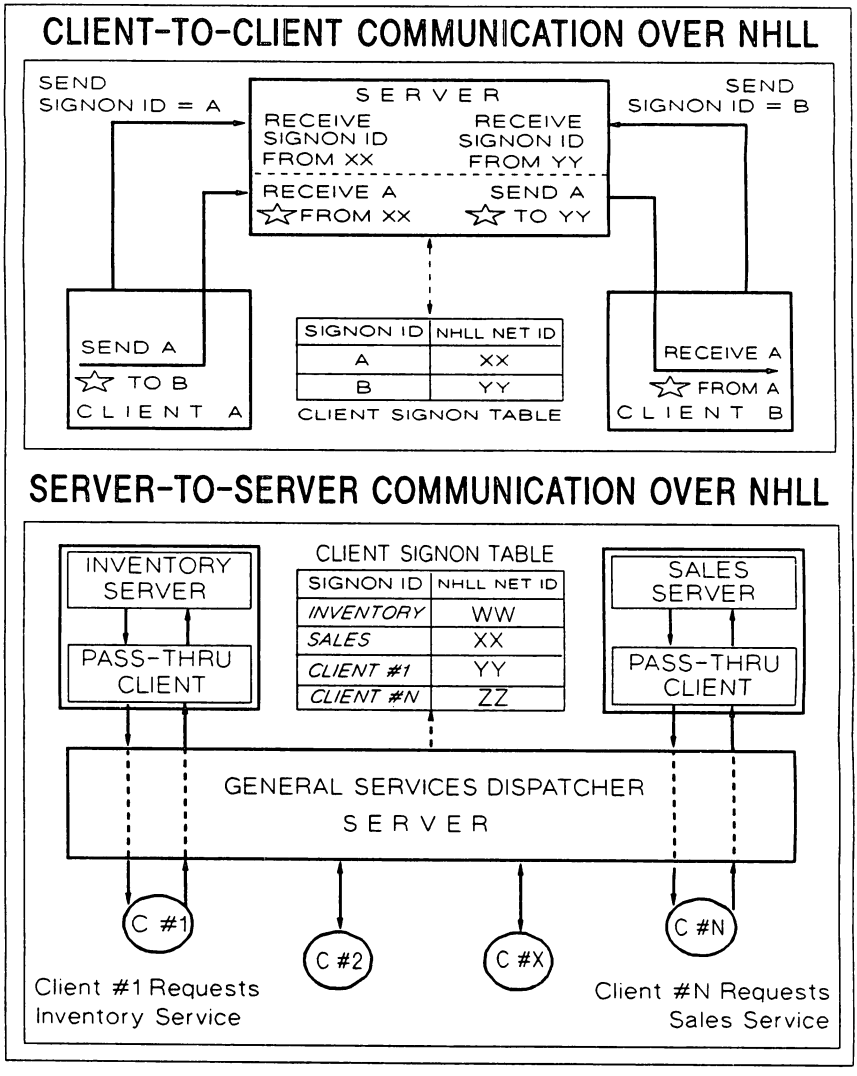
NHLL Data Transfer & Data Conversion Services



NHLL Client/Server Communication Model



Network High-Level Language FIGURE - 3



Network High-Level Language FIGURE - 4

DISK RECORDING TECHNOLOGY-FROM DC TO LIGHT

Michael Rusnack
Hewlett-Packard Company
Disk Storage Systems Division
11413 Chinden Blvd.
Boise, ID 83704

INTRODUCTION

This presentation discusses the evolution of disk drive recording technology. Examples used in this paper mainly refer to Hewlett-Packard products. Even though HP products are highlighted, the same examples can be applied to most disk drive manufacturers.

The goal of this tutorial is to help you to understand the vocabulary associated with the specification of disk drives. From there, using the terms and definitions learned, we will walk through the 20 years of disk drive history-from magnetic recording technology through optical data storage. The goal of this presentation is to bring to you awareness of the trends in disk recording technology over the years. Second, understanding of the offering of each disk drive model. And finally, to determine what are your specific recording needs and how to select a product to meet those needs, be it magnetic, writable optical or read-only optical.

When reviewing a data sheet, often, you will find the phrase "state of the art" or similar claim. What is state of the art? Is it the highest capacity? Fastest transfer? Smallest size? What other specifications should be considered? To begin this 20-year tour through disk recording technology, the vocabulary of the technology must be reviewed. Next, the evolution of the technology will be discussed. This will include the growth of magnetic recording in capabilities, and the simultaneous reduction in size of the devices. Finally, an introduction to optical recording technology will be presented.

TERMS AND DEFINITIONS

Disk drive technology has its own vocabulary, unique meanings of words and acronyms like most technologies. As preparation for the more in-depth technical portions of this tutorial, I would like to review some of these terms:

HEAD - The device used to sense (read) or alter (write) the magnetic signals on the media. Typically there is one head per surface.

DISK - It is important to understand the proper spelling. Until audio compact discs, the spelling was with a *c*. In order to differentiate computer mass storage device from audio devices...

AUDIO/VIDEO - disC

COMPUTER - disK

Certainly it is up to the discretion of the manufacturer how they spell the word, however those in the know spell it with a *k*.

MEDIA - The material used to record the data. Magnetic disk drives use an aluminum platter coated with magnetic particles. Until recently, the disks were coated with a thin layer of iron oxide. The iron oxide is sprayed on to the disks as it is spun at a high rate of speed. The oxide spreads over the surface, collecting at the outside diameter.

Higher recording densities have been achieved as the result in breakthroughs in *thin film* media development. The same aluminum disk is used; however, very thin layers of metal are sputtered onto the surfaces. The result is a very shiny, mirror-like surface. Typically, both sides of the recording surface are used to store data.

Recording media can range in diameters from 14" to 2.5". Industry standard sizes include 14, 8, 5.25, 3.5 and 2.5" diameters. Figure 1 below shows a size comparison of these disks.

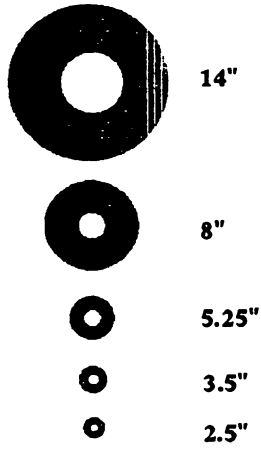


Figure 1- Comparison of Media Size

BIT - The term is derived from binary logic, a bit is either *on* or *off*, one or zero.

BYTE - Eight bits make up one *byte*. The memory capacity of a device can be specified in either *bits* or *bytes*. You can be easily confused if the *bits* and *bytes* were confused.

A convention often used when describing the capacity of a device is to abbreviate; two million bytes is 2 MB. Five million bits is expressed as 5 Mb. Note the large B (bytes) and small b (bits).

BPI - Bits per inch

FRIPI - (pronounced fripee) Flux reversals per inch. The last three are measures of data density. Basically, this is a description of how tightly packed is the data on the media. This corresponds to the capacity of the drive.

BPI and FRIPI are measures of linear density. The data on the disk is written in concentric circles or tracks. If you were to take the track (circumference) and draw a straight line, this would be the linear density.

TRACK - A concentric circle of data, typically numbered 0 to N, from the outside (OD) to the inside diameter (ID).

TPI - Tracks per inch

Sector - The track is made up of sectors. Each sector contains information necessary for position verification, data and error correction. Figure 2 below shows the typical format of one sector.

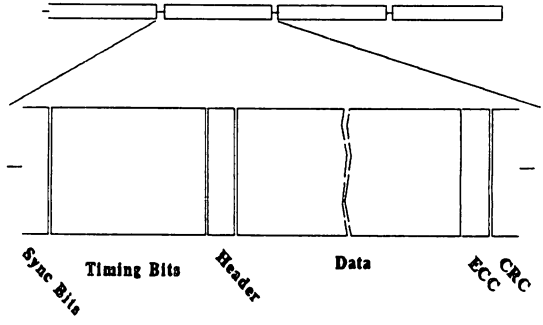


Figure 2 - Sector Make Up

The sector has several components. The first bits encountered (after the sector gap) are the *sync bits*. These awaken timing circuits that use the *timing bits* for clock synchronization. Next, the *header* contains address location, including head (or surface), cylinder (or track) and sector number. The *data* field contains the user data. The ECC (Error Correction Code) and CRC (Cyclic Redundancy Code) are used to ensure the accuracy of the data.

ACTUATOR - The actuator is the device that positions the heads on to the media for the purpose of reading and writing the data. There are several types of actuators. Low cost drives, floppies and hard drives alike, may use a stepper-motor. This actuator is slow, and its steps are larger compared to linear or rotary actuators. The result is the need for tracks to be wider and spaced further apart.

Extremely high performance actuators use a linear voice coil. The actuator moves in a straight line, front to back. Although high performance movement is achieved, the actuator assembly is large and bulky.

A compromise is the rotary actuator. Like the tone-arm of a record player, the arm rotates at an arc across the surface. Speed and accuracy are present in a much smaller and more compact package.

SERVO - To understand the differences in disk drives, and their evolution, some mention must be made about the servo positioning system. This refers to the system that locates the read/write head over the proper track. The type of positioning system affects three parameters:

Accuracy
Time to move from track to track
Track density

There are several types of servo systems, each have benefits and detriments. Listed below are some types and examples:

Electro-Mechanical	Stepper motor
Optical	Source/sensor (graticle)
Embedded	
Dedicated	Written on *servo* surface
Sampled	Written with *data*

The stepper motor is used most typically in floppy drives, and the embedded used in hard disk drives.

The *dedicated servo system* uses a dedicated surface to contain the position information. The *sampled servo system* writes the position information in areas that no data is written, the gap between sectors.

THE MAGNETIC DOMAIN

Early disk drives used removable disk cartridges or packs. These packs allowed users to interchange data much like a floppy system today. Magnetic recording on hard disks is much like tape recording in that a magnetic material is magnetized in specific patterns that represent encoded data. In the case of tape technology, the tape (magnetic media) is drug over the read/write head. The speed of the tape is optimized to minimize wear to either the tape or head. The data on a disk passes by the head at over 1600 inches per second, compared to 30 inches per second for a tape drive. To read and write data at these speeds, the head must not actually contact the disk, however it must be very close. The data head had to "fly" over the data; typical flying heights range from 15 to 25 micro-inches. It is difficult to relate to just how much 20 millionths of an inch is. In the figure below, several comparisons are made to everyday items.

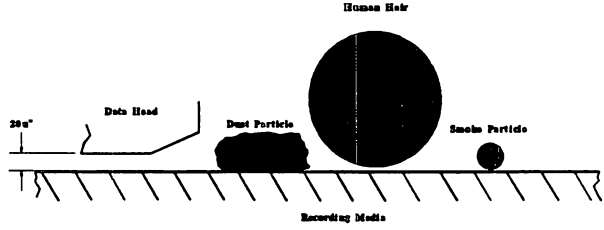


Figure 3 - Flying Height Comparison

The stable flying of the head is critical. At the equivalent of 90 miles per hour, the head must be fly stable in a clean chamber. To ensure this, early manufacturers required that the media chamber be sealed, purged and filled with nitrogen or even helium. As the technology developed, the sealed/pressurized chamber was replaced with an air filtration and circulation system. A blower and an absolute filter were used to circulate *clean* air through the disk chamber. Often, the disks will spin to purge the chamber of any contaminates prior to the loading of the heads.

If the head and media were to come in contact, no matter the reason, irreparable damage will occur. This event is called a head crash. Often, the slightest contact will result in the damage of the media surface. The oxide surface is soft and is easily removed by the hard-glass head. This results in the generation of contaminates. The contaminants are generated faster than the filtration system

can remove them, thus the crash propagates. The final result is total loss of data; the recording media is literally scraped away.

The early disk recording technology was lead by IBM (International Business Machine). Whenever a "futures" slide was presented, everyone was one generation behind IBM. Often, the disk technology was referred to by the heads/media interface. Flying head drives like HP's MAC family, HP 7906/20/25 drives used the IBM 3330 technology head design. These heads were relatively large and cumbersome. They literally flew over the disks. Pictured below is a head assembly. At rest, the assembly is bent as seen below. To bend the assembly straight takes 150 grams (or 5.25 ounces). The upward force that keeps the head flying above the media is due to the aerodynamic design of the head assembly.

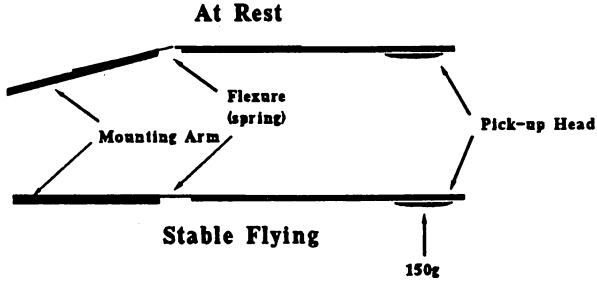


Figure 4 - Head Loading

This technology used 14 inch (or larger) platters. With bit and track densities relatively low, the only way to increase storage capacity per spindle was to add additional platters. A 14 inch disk has about two inches of usable space for data storage. Hewlett-Packard's first offering in this area was the HP 7900; a 14-inch drive using two platters. This yielded 5 MB total storage. This first offering had 100 TPI with 2200 BPI. How does this compare with future offerings? Illustrated in the figure below is the increase of tracks per inch and the increase of data storage per 14 inch surface.

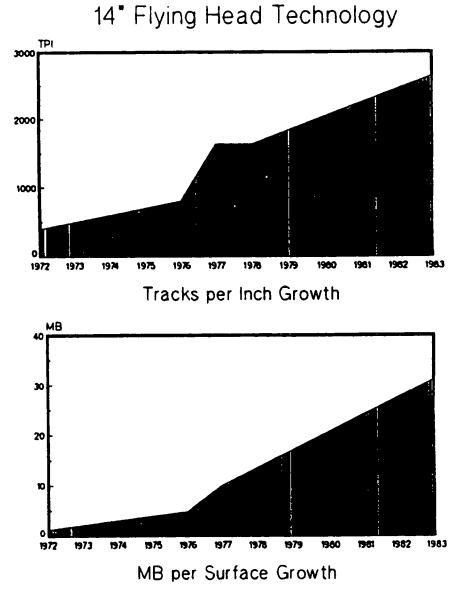


Figure 5 - Technology Growth

The capacity needs of computer users were being increased over the years with new products every couple of years. Each product increased the bit and/or track density with each generation. To further increase capacity, more platters were added to the drives.

The demand from the disk drive users was not only to increase storage, but to decrease the size of the package. The disk drive was not only examined for it capacity, but its size and power draw were studied.

Again, as with previous entries into the disk drive market, the technology was driven by IBM. This class of drives was referred to as *Winchester Technology*. The differentiating factor of this technology was that the heads did not retract from the disk's surface at power down. An area where the heads were allowed to reside when the disks were not spinning was designated as the landing zone. This area was typically located at the ID, well away from any customer data. Before the development of the harder, thin-film media, the oxide media was coated with a thin layer of oil that reduced the chance of damage when the heads came to rest on the disk's surface.

The second distinguishing feature was the heads and media were now sealed in a nearly air tight chamber. With flying heights being reduced and tighter tolerances being maintained, the sealed HDA (Head Disk Assembly) was born.

With the exception of a small vent hole, the heads and media were sealed from outside contamination.

New oxide formulations and smaller head sizes allowed the increase in tracks per inch, thus resulting in even greater storage capacity per data surface. The demand for greater storage per spindle grew, as did the requirements for smaller sized disk drives. To meet these needs, the next step was to thin-film media. These disks start with the same aluminum disk substrate as the oxide, however the magnetic coating consists of several layers of metal deposited on to the surface by sputtering. The first HP product to use this new technology media was the HP 7936/37 family of disk drives. The cross section of a thin film disk is depicted in the graphic below. This is substantially more complex than the oxide media.

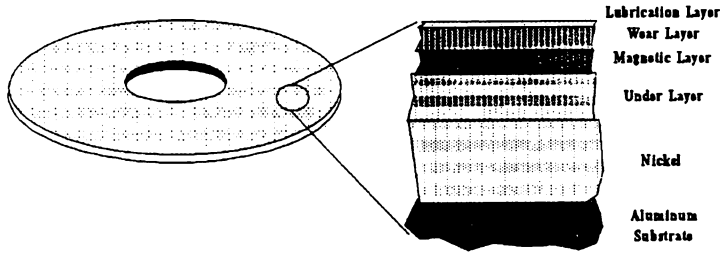


Figure 6 - Thin Film Layers

The thin-film media used was 8 inch rather than the previous 14 inch used in earlier products. Though the 8 inch disk had only 1.25 inches of usable area, the capacity per surface was 37.8 MB: a gain of twenty two percent, more storage capacity on a disk that is nearly half the diameter and one third the footprint of the previous disk products.

Like its *Winchester* predecessor, the thin-film disk drive's HDA is sealed. The new technology disks present a new challenge- corrosion. Special filtration systems must be applied to not only remove particulates, but to remove moisture and corrosive chemicals. In parallel with the reduction in the size of the HDA, a corresponding reduction in the overall number of electronic components occurred. Many of the functions performed by hundred of discrete components are now integrated a few chips. Through hole electronic components are replaced by much smaller surface mount components to save space and reduce the size of the printed circuit assemblies. Electronics that once were contained on three PC boards are reduced to one board, and one half the size of earlier boards. With the smaller HDAs and highly integrated PCAs, the reliability of these disk drives exceed their predecessors by a factor of ten.

The demand for more storage capacity grows with each and every computer system. With the increase in competition within the disk drive industry, the focus is on cost reduction. The new technology media is the secret to the increase in MB per drive, however the cost of the new disks are several times that of the oxide media. The focus is not on capacity per spindle but the cost of that unit. There were two primary methods to reduce cost - use 5.25 inch disks and manufacture in volume. Using smaller disks, more can be produced per deposition machine. Secondly, at higher volumes the cost per disk drops significantly.

Where is the limit? Where will it all end? We have experienced growth in capacity, offering hundreds of MB on a single spindle. Conversely, we have watched the disk size shrink from 14 inch to 8 inch all the way to 2.5 inches. From the table below, the answer may be apparent.

Technology Trend for Magnetic Media

	TPI	BPI	MB/ Surface
8"	1121	18.8K	22
5.25" - I	1950	20.5K	27
5.25" - II	1667	30.6K	48
3.5"	1850	42K	47
2.5"	1456	39.7K	11

Figure 7 - Technology Trends

From the second generation 5.25 inch disk on, the densities appear to have peaked. Indeed, there will be a breakthrough in the head/media technology someday. Vertical recording technology is in the laboratory today, and is said to be the next step in magnetic recording. Until then, or another technology breakthrough, these physical limitations will remain the barrier.

...and then there was light

With the advent of *Winchester* technology and the sealed HDA, the computer users are in need of a removable media drive. Applications such as data interchange, archive and security, optical data storage have come to pass. In addition to having a removable storage medium, the optical drives are lower in

cost than their magnetic counterparts. Magnetic storage will not be displaced by optical technology today, however, as it has much slower performance.

Data is written (and read) using a LASER as opposed to a magnetic head. Due to the ability of the LASER to be focused to a very small area, the data densities on optical disks are 10 times or more than the equivalent size magnetic disk. Hewlett-Packard's Magneto-Optical disk drive Model 650A, boasts of densities of 15,875 TPI on a 5.25 inch disk. This translates to 325 MB of storage per surface. That is a 10 fold increase in track density and over 6 times increase in raw data storage when compared to a magnetic disk.

Features of optical recording technology include:

-Low cost per MB
-High data densities
-Removability of media
-Longevity of written data
-Not susceptible to magnetic fields

There are three types of optical storage devices:

o Read/Write
o Write Once/Read Many (WORM)
o CD-ROM

Read/Write optical devices come in three flavors:

o Magneto-Optical (MO)
o Dye Polymer
o Phase Change

MO media can be erased and rewritten repeatedly, like hard disks. The MO drive uses a high power LASER to assist in the write process, and a low power LASER to read the written data. There is a layer in the MO media that has a very high magnetic resistance. The magnetic field required to alter the bit direction varies greatly with temperature. The LASER is focused to a small point on the media. This action heats the magnetic layer to its *Curie Point*. At this high temperature, the properties of the magnetic layer change, thus allowing the drive's magnetic field to alter the magnetic polarity of the bit. The direction of the magnet's polarity determines whether the bit is a 0 or a 1. This write process is shown in the figure below.

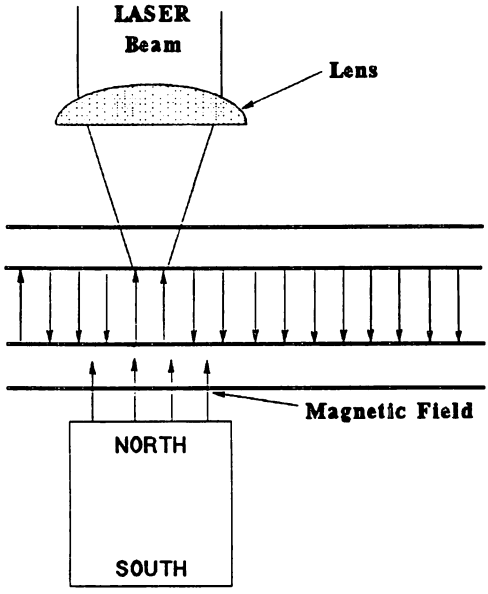


Figure 8 - MO Write Process

To read data written on an MO cartridge, the technology takes advantage of a physical law known as the *Kerr Effect*. This law states that a magnetic field affects the polarity of reflected light. By detecting the light's polarity, the direction of the magnetic field can be determined.

Dye Polymer technology uses a translucent plastic disk with a colored layer which absorbs heat from the drive's LASER. A blister is formed on the area heated by the LASER. Reading the Dye Polymer disk is similar to reading a CD-ROM. The blisters reflect light differently than the flat areas. One drawback of this technology is the media life is limited to less than 10,000 write cycles.

The Phase Change method uses a plastic disk with a special metal layer. Heat generated by the drive's LASER changes the molecular structure of spots on the metal layer from an amorphous state to a crystalline state. To read, differences in the reflected brightness in the crystalline spots are detected. As in the Dye Polymer method, the Phase Change disk has a finite limit to the number of write cycles.

WORM or Write Once Read Many is another type of optical disk drive. WORM uses a disk with a special metal layer. Heat generated by the drive's LASER alters the media surface. Like the process noted above, the WORM media can be written only once. WORM is used for archive for material that will not change, such as insurance data, title history or printed documents. Since the WORM data can not be altered, it is secure and considered a permanent record.

Computer based CD-ROM technology is built on the home stereo mechanism. This read only device can store up to 550 MB of computer data. That is the equivalent of 200,000 printed pages of text, 5000 high quality color images or one hour of CD quality audio.

Data on the CD-ROM is organized in equal-length sectors that spiral from the inside to the outside diameter of the disk. This differs from the magnetic disk, as well as its optical counterparts, in that they are written in concentric circles. The format of the 'digital' CD is the same as the conventional CD that stores music. The result is that integrated with the digital data is audio.

Shown below is how information is recorded on a compact disk.

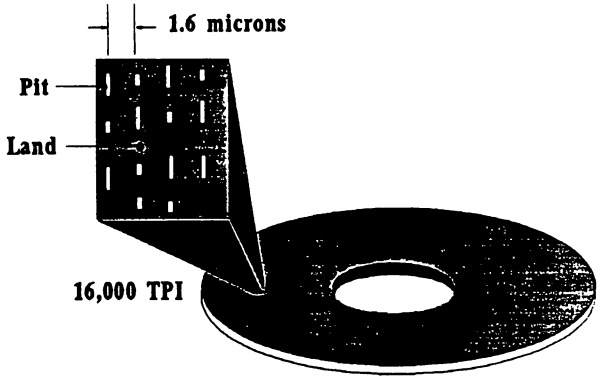


Figure 9 - CD-ROM Data Pattern

The ones and zeros are recorded as *pits*. The flat areas between the pits are called *lands*. The continuous spiral spaces the tracks 1.6 microns apart. This results in a track density of 16,000 TPI, 10 times more dense than magnetic recording.

The process to produce CD-ROM media is relatively low cost and simple. The data is mastered from any of a number of sources, tape, floppy, even magnetic disk. When the preparation of the data is complete, the formatted CD-ROM

image is cut into a glass disk using a high power LASER. The glass master is then taken to an electroplating process where a metal stamper is made. Using the metal stamper, an injection molded disk can be produced one every few seconds. The base material is then placed in a metalization process where it is coated with a reflective aluminum layer. The aluminum layer is covered with a protective layer of clear polycarbonate. Pictured in the graphic below is a cross-section of a compact disk.

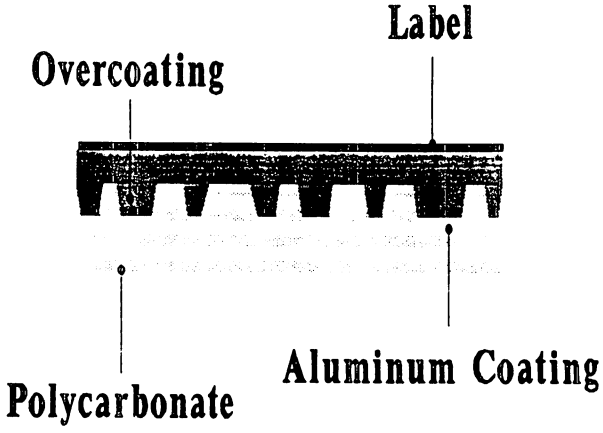


Figure 10 - CD-ROM Cross Section

Encyclopedia Britanica recently introduced "Compton's Multi Media Encyclopedia" that includes spoken segments of famous speeches, classical music and a 20 minute glossary of terms. The encyclopedia, with 31,000 articles, 15,000 photographs, charts and diagrams is only one of the many applications of this young technology.

For many years, IBM has led the charge in computer technology. Today, there are many companies with R&D efforts in this field. Although "Big Blue" manages to be on the forefront of technology most of the time, the rest of the world is right there too.

This presentation has provided you with a brief overview of the evolution disk recording technology. To cover every aspect of recording technology would take hours. It is my intention that presentation has provided you with new knowledge and understanding of this technology. With this new awareness, I hope that you will see your disk drives as more than just data storage.

DISK ARRAYS - MASS STORAGE OF THE FUTURE?
Paper 3125

Ed Pavlinik
Hewlett-Packard
Disk Storage Systems Division
11413 Chinden Boulevard
Boise, Idaho 83707
(208) 323-2060

INTRODUCTION

Disk storage arrays have recently become a leading topic of discussion in the computer systems business, not just among industry gurus, but also at trade shows and user group meetings worldwide. A variety of papers have been published and numerous articles have appeared in industry publications. As a result, innovative users are more than mildly curious about the ramifications of using these devices on their computer systems. In fact, some users may already be using disk arrays on PC based systems.

What makes the disk array different from traditional on-line disk storage devices and how can the typical data processing manager take advantage of this new technology? The goal of this paper is to provide an understanding of disk arrays as a potential solution to changing mass storage requirements.

THE DISK ARRAY CONCEPT

Large Capacity Storage from Small Disks

The typical disk storage array is basically a mass storage system utilizing a large number of small form factor disk drives, such as 5.25" or 3.5", which are linked together with an intelligent controller to provide a large amount of disk storage. Capacities of the 5.25" diameter disks have been steadily increasing as disk manufacturers push the limits of magnetic recording technology. Similarly, 3.5" diameter disk drives have also been increasing in capacity due to improvements in track and bit densities.

Since the smaller diameter disks are also being mass produced in large volumes, production economies of scale will result in lower manufacturing overhead and cost per megabyte when compared to eight-inch or fourteen-inch disk mechanisms. Obviously, the same storage capacity can be designed into a large single spindle high capacity disk drive. The drawbacks of the single large expensive disk are higher manufacturing costs due to lower production volumes and potentially lower performance, since so much data is stored on a single spindle under one actuator.

The disk storage array allows larger computer systems to take advantage of many of the benefits associated with smaller form factor disk drives and overcomes some of their limitations through the intelligence built into the disk array controller. The disk array controller can be designed to offer a high degree of flexibility in meeting

diverse user requirements. Optimization of the various tradeoffs can also be achieved in a general purpose array or if appropriate, other special arrays can be designed to solve specific user needs.

Array Terminology

The terminology used to describe disk arrays can lead to some confusion, since several vendors use the term "disk array" to describe different products. For example, some data sheets refer to products as disk arrays, even though technically they might more properly be described as conventional disk storage systems or disk cluster controllers. By another definition, the term disk array consists of multiple disk drives under the overall command of a single controller. Despite possible confusion in terminology, the primary function of the array is to increase total storage capacity as well as to improve performance by selectively spreading files over multiple disks. The file spreading technique utilized by the intelligent controller is called disk striping. Striping is a technique which writes a single byte of information to each disk drive in the array in a parallel fashion. All the striped disks work in unison on a single I/O transfer.

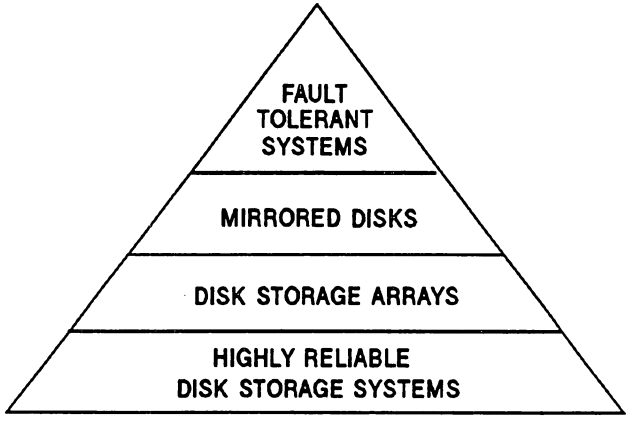
Disk striping can also be accomplished in software by designing a special disk driver to spread files across several individual disk volumes. In this case the computer performs all of the data manipulation as well as the disk management and as a result, the overhead may be quite high. In a disk array the array controller does the striping outboard of the CPU, thus saving CPU cycles for other tasks. Products are available from a number of vendors which illustrate both methods of disk drive data striping.

Data Availability Hierarchy

A good way to visualize the concept of data availability is to consider a complete hierarchy of data protection. In this model, higher data availability is achievable via various techniques for a corresponding increase in cost. Depending on the level of availability desired, one can pick a point on the cost curve for the specific solution desired. For example, if a medium level of availability is perfectly adequate for some applications, there are solutions available to meet those needs for a lower cost than the completely redundant fault tolerant solution.

Here's an example showing these various levels of availability starting from highly reliable disk storage systems and then working upward to progressively greater data availability through the use of such innovations like disk arrays, disk mirroring, and fault tolerant systems. As you might expect, costs will increase as one moves toward the top of the pyramid. Many applications, however, demand these high availability systems despite the higher costs, since the benefits outweigh them. It all depends on the cost of system downtime unique to every business.

DATA AVAILABILITY HIERARCHY



The highest level in this hierarchy represents fault tolerant systems which incorporate full redundancy and include disk storage systems with no single point of failure. As a result, a system is created that features almost no downtime. These systems represent perhaps the ultimate in high data availability for use in mission critical applications and offer continuous data processing with such features as multiple processors, redundant I/O, and software checkpoints.

Disk drive reliability may be considered to be a cornerstone or foundation in this hierarchy upon which additional layers of data protection may be constructed. Continual improvements in disk drive reliability have made possible the design of disk mechanisms with a Mean Time Between Failure Rate (MTBF) in excess of 150,000 hours. However, this large MTBF does not guarantee against failures, but only implies that failures will occur less frequently on average.

Moving up to successively higher layers in the data availability hierarchy becomes increasingly costly, since redundant duplication of hardware is a requirement for fault tolerant systems. However, the middle sections of the hierarchy may offer acceptable levels of data availability for far less cost than the fully "bulletproof" solutions. Many applications can take advantage of this medium level of availability and the lower cost is an attractive feature. These intermediate levels of the availability hierarchy consist of disk storage arrays and disk mirroring products which will be examined in more detail in later sections of this paper.

Increasing disk hardware reliability will have an enormous effect on data availability. However, a high Mean Time Between Failure (MTBF) for a disk drive does not guarantee against disk failure. It represents an average number based on field experience or a theoretical calculation based on component failure rates. High

MTBF disk drives will still fail, but not with the same frequency as they have in the past. When a disk failure occurs, it may bring down the computer system for a long period of time, particularly if the data has been corrupted and a system reload is needed. Ten hours or more of downtime is not an uncommon result when a disk failure occurs. To many customers this amount of system downtime is intolerable.

As the number of individual disk mechanisms in an array increases, the Mean Time Between Failure (MTBF) for the entire disk storage system decreases proportionately. This makes intuitive sense, since the more components that exist in a system, the less reliable the total system becomes. For example, the approximate number for the average MTBF of a disk storage configuration will be the average MTBF for a single disk divided by the number of mechanisms attached to the system. Given these facts, an array consisting of multiple disk mechanisms required a new method to increase data availability as a means to avoid more frequent disk failures and resultant system crashes. This led to the search for a controller design which maintains system and data availability even in the event of a disk failure.

As a result, the parity disk concept was developed, providing the array controller with the intelligence to reconstruct data from a failed disk drive on demand. This results in a data protection shield which renders a disk failure completely transparent to the user. Disk failures in a parity disk array now have no effect on the end user! The array will now provide high data availability, use lower cost disk mechanisms, provide large quantities of storage, and provide performance improvements, all at the same time. Consequently, the disk array is a highly flexible mass storage device by virtue of meeting all these diverse user requirements.

Disk Array Performance

Although the primary function of the array is to provide a large amount of storage at a high level of data availability, in certain modes and applications some performance gains may be realized. Through the use of an intelligent controller managing the operation of several disk drives, it is possible to achieve performance gains from a number of different perspectives. If all spindles are synchronized and the data is striped over all disks, data transfers take place in parallel. Therefore, the array has a potential speed advantage for large data transfers. Instead of one disk transferring data, the array has multiple disks transferring simultaneously. In this mode of operation the disk array appears to the host system as one large disk drive.

In another controller design, the array may appear to the computer system as several unique disk drives all operating independently of each other. This allows for concurrent disk mechanism operation without data striping or parity, useful for very I/O intensive applications, since multiple I/O's can be executed at the same time.

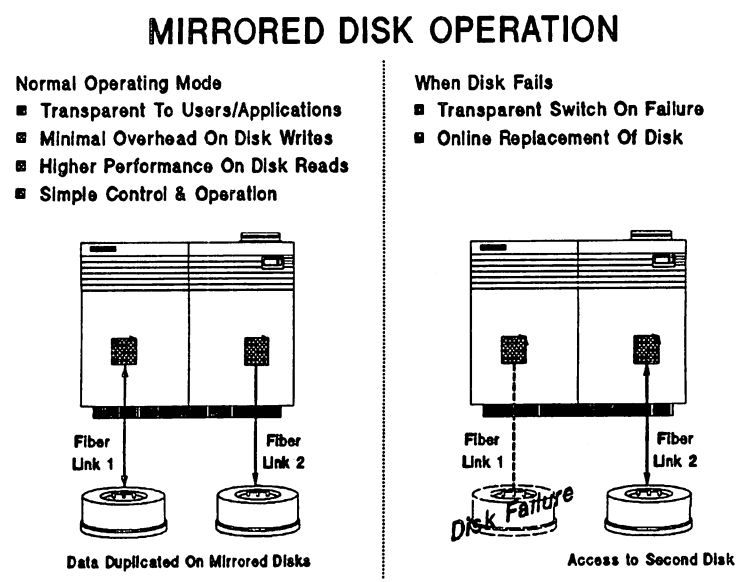
Different types of arrays can also achieve concurrent operation for certain operations by working together in pairs of a much larger group. Arrays of this type can process multiple small transfers simultaneously to speed up performance. In this example, the striping is done on a smaller number of disks in the group for small transfers and large transfers will keep all the drives busy in parallel. This will be explained in more detail in subsequent sections of this paper.

REDUNDANT ARRAYS OF INEXPENSIVE DISKS (RAID)

A common industry buzzword associated with the subject of disk arrays is called **RAID**, an acronym for Redundant Arrays of Inexpensive Disks. Many different configurations of disk arrays are possible, depending on the requirements of the end user and the goals of the manufacturer. Each design has a different functionality built into the controller to accomplish specific goals related to disk performance and data availability. A University of California at Berkely paper entitled "A Case for Redundant Arrays of Inexpensive Disks (RAID)" by Patterson, Gibson, and Katz, summarized five categories of disk arrays. Since only three of these configurations are practical for most on-line transaction processing systems, we will examine them in more detail.

RAID LEVEL ONE - DISK MIRRORING

This classification describes the concept of disk mirroring, where fully redundant disk drives are used to store data on a computer system. Here's a representative sample of a typical disk mirroring storage system:



In the event of a disk failure, the special disk mirroring software will automatically switch all I/O activity for the mirrored pair of disks to the surviving disk in the pair. Repair and resynchronization of the failed disk drive can be done transparently to users and applications. The system knows that a failure has occurred via a console message, but the users and their applications will have the same access to the data as

if nothing at all had happened. High data availability is a key user benefit from disk mirroring, since it extends system uptime by saving the system in the event of a disk failure. Mirroring has minimal overhead on disk writes, since two copies of the data and any changes have to be made. Higher performance can be achieved on disk reads however, since I/O can be processed by two disk spindles concurrently. This provides a performance benefit in addition to high data availability.

The disk mirroring solution duplicates the entire disk system, thus protecting the data against power supply, controller, fan, and cabling failures, in addition to a failure in the disk mechanism. The disadvantage of disk mirroring is the cost of duplicating the disk drives, making the effective cost per megabyte twice that of an unmirrored system. Actually, the true cost may actually be more than twice as much due to the cost of the disk mirroring software which manages the entire operation. For data which must be protected at all costs and remain on-line in a high availability system, disk mirroring advantages will far outweigh the higher costs.

Disk mirroring also allows a system to be designed to achieve on-line backup. Since there are two disks storing every bit of data on the system, one disk in the mirrored pair can service on-line transaction processing, while the other can be dedicated for backing up to a secondary storage device. The only delay in the system is approximately five minutes of quiescent time at the beginning of the process to maintain data integrity. After the backup has been completed, the software schedules an on-line resynchronization of the mirrored disks.

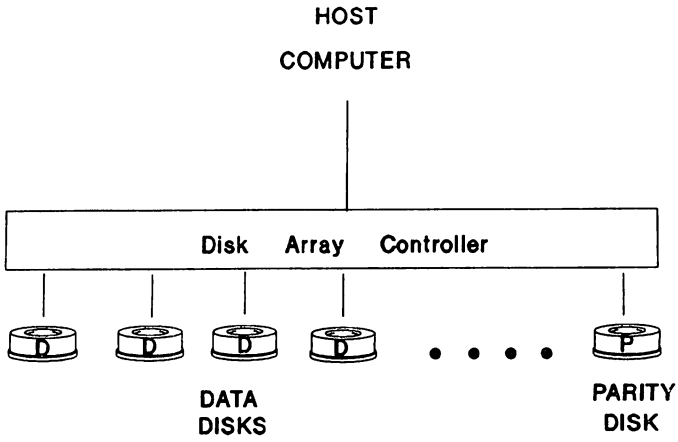
RAID LEVEL THREE - PARITY DISK

This type of disk array uses a separate parity disk to store checksum data. The function of the parity disk is to store the EXCLUSIVE OR of the data kept on the data disks. This allows for a bit-by-bit comparison and subsequent reconstruction of the data in the event of a failure of one of the data disks. The number of data disks in the group is usually chosen to be an even number and will depend on total capacity desired as well as packaging considerations. The disk spindles are synchronized so that at a given point in time all the heads in the group of disks are reading or writing on the same sector location in parallel.

Data is spread or striped across all the data disks in the array on a byte-by-byte basis, and the array appears to the system as a single large disk drive. During normal operation the array transfers data in a parallel fashion at the theoretical rate of a single mechanism multiplied by the number of data drives. Actual transfer rate depends on the host bus adapter bandwidth and system data patterns. For some applications, a performance improvement will be achieved in the resultant higher transfer rate. Every write operation will involve all disks in the array, since new parity needs to be written to the parity disk. Reads involve all the data disks in the array.

Here's an example of a typical dedicated parity disk array configuration. Notice how the parity disk can protect even multiples of data disk drives, even though only four disks are shown.

DISK STORAGE ARRAY
PARITY CONFIGURATION



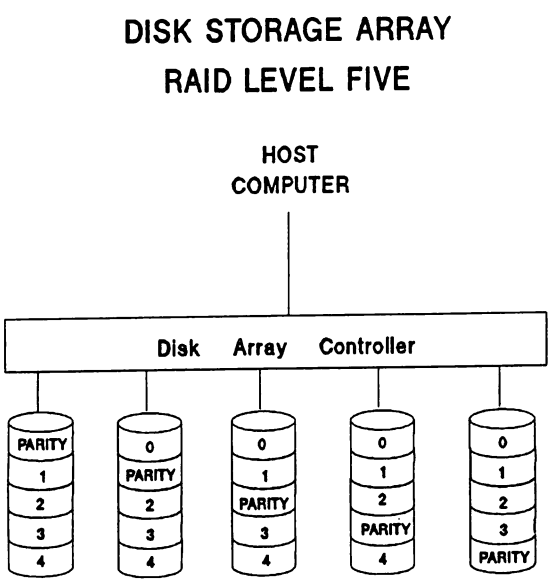
In the event of a disk failure, the array controller reconstructs data that is missing from the failed disk through the use of the parity information. Since all the bytes are buffered, there should be no loss of performance as the controller is specially designed for speed to accomplish the extra tasks involved. The array can operate in this mode with no loss of efficiency until the chance occurrence of another disk failure, but statistically the probability of that happening is very small. The system knows that a disk failure has occurred, and the failed mechanism can be replaced at the next service call or else during a slack period in the system operation. A new drive can be inserted to replace the faulty unit and the array controller will rebuild the data on the new disk. During all this activity, the system is still available for on-line transaction processing applications.

The cost per usable megabyte of storage in this type of disk array increases due to the dedicated parity disk which cannot be used for storage of user data. For example, in a four-way striped parity array, there is 25% overhead in storage costs. This is due to the fact that the product consists of five disk mechanisms, but user data can be stored on only four. The overhead associated with the parity is a small price to pay for the increased system uptime.

Other components of the array can be duplicated to guard against power supply and controller failures. A large amount of flexibility exists for variations in the design of this type of array, depending on user requirements.

RAID LEVEL FIVE - DATA EMBEDDED WITH PARITY ACROSS ALL DISKS

This design is more complex and therefore more costly than the level three disk array. Here's an example of a mass storage array incorporating a level five controller:



The data is still striped across all the disks in the group, but the parity information for each sector is not stored on the same disk. The array controller manages the generation and location of the parity information for each sector stored. For example, the controller parity and data storage sequence might be as follows: for sector zero the parity data is stored on disk one, sector one's parity is stored on disk two, sector two's parity is on disk three, sector three's on disk four, etc.

This level five array may in some applications have decreased write performance, depending on the system, since for a write involving just a few disks in the group for a small block of data, all the disks in the group need to be read, new parity calculated, and then new parity information rewritten. This "READ-MODIFY-WRITE" cycle represents extra overhead for small writes, when compared to a level three design. On small reads involving just a few disks, better read concurrence occurs, since the array may be processing multiple I/O's to different disks in the group.

In the event of a disk failure in a level five array, the missing data is calculated from the parity or checksum information in the same fashion as in the level three array. Data availability is correspondingly increased due to the extension of system uptime,

TITLE: HP 3000 Systems Management

AUTHOR: Robert Winter & David Strauss

Hewlett - Packard

c/o Ella Washington

19091 Pruneridge Avenue; M/S 46LK

Cupertino, CA 95014

FINAL PAPER NOT AVAILABLE, HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

PAPER NO. 3126

Paper #3127

Managing MPE/XL Configurations

Doug Claar and Fred Parkes
Hewlett-Packard
19447 Pruneridge Ave.
Cupertino, Ca. 95014
(408) 725-8111

MPE/XL offers great advantages over MPE V/E by providing increased flexibility and capabilities to manage system hardware configurations. System managers can use configuration files to customize their system configuration and manage multiple configurations.

This paper provides information that will enable MPE/XL system managers to effectively use and manage configuration files to meet their needs. The following questions are answered:

- How can I use SYSGEN to customize my system's configuration?
- How can I manage the configurations of multiple systems from a single system?
- Can my system run without a configuration?
- What precautions should I take?

With this information, system managers will be able to manage single and multiple systems more effectively and to avoid common configuration problems.

In order to describe the management of configuration files, we first define some terms and then set up a system. We will start with a factory system load tape (SLT) and install from it. Then we will do the first `START` after the `INSTALL`. Next we will describe the process of changing the configuration files for the system. This allows for as many iterations as required to perfect it. We then write a customer SLT for this system and update from it. Then we will describe further changes a system manager might need to make to the configuration files, including multiple configurations for the same system and multiple systems. Finally, we talk about common problems with configurations and how to recover from them.

We will limit our discussion to configuration files of the MPE/XL software platform. We do not discuss data communications and terminals, applications, or how to configure and run your business applications.

Definitions

When discussing system configuration, there are several terms and concepts that must be understood and agreed upon. Many of these are defined here.

Configuration Files. *Configuration files* are a set of eight files in the same group. These files are managed by SYSGEN and contain configuration information. These files come as a set, and some of them have pointers into other files in the set. The number of files in the set may change in the future. Configuration files must reside on LDEV 1, the main system volume.

Configuration Groups. A *configuration group* contains configuration files and is in the `SYS` account. The groups are created when the system is installed or updated or when `SYSGEN` needs a new group for keeping configuration files.

System Load Tape. A *system load tape* (SLT) is a tape written by SYSGEN. The SLT is in a format that can be booted by 900 Series HP 3000 hardware. It is *not* in STORE/RESTORE format. An SLT contains two sections, a tape boot section and a disk section. An SLT contains enough files and data to install an MPE/XL system on disk. After the installation, the system can be started and the console can be logged on as MANAGER.SYS. An SLT contains one set of configuration files in the group CONFIG.SYS. These are the files that are treated as configuration files by a system update. They are written to disk during an UPDATE CONFIG or any INSTALL. SLTs come in two types, factory and customer SLTs.

Factory System Load Tape. A *factory SLT* is created by Hewlett-Packard and contains a factory version of MPE/XL. This level of the operating system must be tailored by AUTOINST to provide business solutions. The tape also contains an "empty" configuration group, a copy of sample configuration files, and system logging numbers (used to determine the starting system log file numbers) set to zero.

Customer System Load Tape. A *customer SLT* is created by SYSGEN and contains a set of configuration files in the group CONFIG.SYS. By default, the system logging numbers will be non-zero, and the tape does not have any of the sample configuration files supplied by Hewlett-Packard. The operating system on the tape is a duplicate of the system it was written from. A customer SLT configured and written by SYSGEN while running AUTOINST will be ready for use as a software platform to install and run customer solution software.

When Is a Config File Not a Config File? On disk, configuration files are the files that contain configuration information. They are located in a group in the SYS account on LDEV 1. Configuration files are managed by SYSGEN.

On a system load tape, written by SYSGEN, there is only one set of files that are configuration files. These are the files in the group CONFIG.SYS. They come from the files in the base group that SYSGEN is pointing to when the TAPE command is invoked. Configuration files on an SLT are always written to disk by INSTALL. Configuration files are *not* replaced by an UPDATE NOCONFIG (the default parameter). Configuration files are replaced by an UPDATE CONFIG.

All other files on an SLT are either system files, IPL files, boot files, or autoboot files. The files on an SLT that look like configuration files (Hewlett-Packard-supplied sample configurations) are system files. System files are always written to disk by INSTALL and are always replaced by UPDATE.

The file IDP.CONFIG.SYS, on an SLT, is a system file—not a configuration file. It is always replaced by UPDATE and is always written to disk by INSTALL.

Empty Configuration Groups. An empty configuration group does not contain any configuration files except the file IDP. A factory SLT contains an empty configuration group CONFIG.SYS because the tape is intended for any system.

Default Options. The default options discussed here are START GROUP=CONFIG RECOVERY and UPDATE NOCONFIG.

Configuration Files in the BOOTUP Group. The configuration files in the group BOOTUP are saved there during a START NORECOVERY. They are used during a START RECOVERY. They are protected while the operating system is up and running.

Paths. The PA-RISC architecture defines where to store the information about the hardware paths to the devices that are needed to boot. The primary boot path tells where to find the device that the PA-RISC machine will usually boot from. This is almost always a disk. The alternate boot path tells where to find an alternate boot device, usually a tape drive. The console path tells where to send messages to and receive input from. This is usually the terminal used as the system console. Once the system is up, the primary boot path becomes LDEV 1, the console path becomes LDEV 20, and the alternate boot path usually becomes LDEV 7.

Setting Up a System for the First Time

A 900 Series HP 3000 hardware system begins life as hardware with no system software. The disk volumes are blank. The process of creating a running system starts with an INSTALL from an alternate device (in this case, a factory SLT on a tape drive). The INSTALL image is booted from the tape and initializes the system master disk, LDEV 1. Disk labels, system objects, and a file system with a minimum account structure are added. INSTALL then reads all the files from the SLT and writes all of them to LDEV 1 using the file system. When the installation is finished, the system then has enough data to boot the operating system. The configuration files on disk may not be what you expect. The group `CONFIG.SYS` is an empty configuration group. It contains a system identification file, `IDP.CONFIG.SYS`, but does not contain any configuration information. The configuration information is in the Hewlett-Packard-supplied sample files in groups such as `CONFG950` and `ALINK925`. The configuration in these groups is our best guess at what typical systems might look like. We intend to make your job easier by providing a configuration that requires minor changes to match your configuration. For more information, including which sample configuration group to use, please consult the installation manual for the factory SLT and the *System Startup, Configuration, and Shutdown Reference Manual* (32650-90042). The first startup after a system installation is different and requires some special attention.

The first startup after system installation from a factory SLT needs a `GROUP=` parameter with an appropriate configuration group. For example:

```
START GROUP=CONFG950 NORECOVERY
```

Using a group with configuration information in it gives the system a place to start. Starting with the default group, `CONFIG`, after an install from a factory SLT is not a supported way to start a system. The first startup after installation initializes additional members of the system volume set if they are configured, ready, and not a private volume. The first startup after system installation also initializes system values. Those are the values identified as changed in SYSGEN MISC configurator only when an UPDATE from tape is performed (for example, RINs and GRINs). The configuration group used by the `START` command is also the default group used by SYSGEN when it is run. If that group is empty, there is confusion when SYSGEN is run. It is best for all concerned that this first startup after installation be given a valid configuration group.

Once the system is running, it is time to change the software configuration information to match the real hardware configuration. This is one of the places where SYSGEN is used.

Using SYSGEN to Manage Configurations

Unlike many other systems, including MPE V/E, MPE/XL allows multiple configurations to be present on disk. As we saw earlier, the system can be started from any one of these configurations. There are some interesting possibilities here. We'll talk about using multiple configurations to

manage multiple systems later, but even with just one system, there are some uses. One use is to create a backup copy of the old configuration in case the new one doesn't work. It is much nicer to come up with what you had before than to come up with just your primary boot path, secondary boot path, and console path. Another possibility is to use multiple configurations when you're only going to need a particular configuration for a short while. For example, if a third party is going to use your machine for a few weeks and you need to hook up some equipment for them (or remove it), you can keep your regular configuration somewhere else and put it back when they leave.

The only "UL-approved" way to move configurations is with SYSGEN. Other methods run into pitfalls, such as moving the configuration files off the boot disk: You can't configure a disk into the system if the information to configure it is on that disk! Configuration files are an interlocking set: It is dangerous to mix and match individual configuration files. A system produced by mixing configuration files will usually not come up correctly. If you stick with using SYSGEN to work with configuration files, you'll be assured of success.

Using SYSGEN to Change Configuration Information

Some people are overwhelmed, at least initially, by the number of commands and options available in SYSGEN. It turns out that most things work quite nicely with the default values, so that, for example, even though there are more than 20 parameters available to the `ADEV` command, usually only the three required parameters are needed. Among the configurators, the IO configurator is probably the most frequently used; so we'll deal with that one in some detail, but first we'll take short trips to the other configurators.

The LOG Configurator. The LOG configurator is fairly straightforward. It is used to determine which system events will be logged, as well as to determine two characteristics of user logging. *System logging* is the process by which the system records information about the goings on in the system. Things like file opens, user logons, console messages, and many other events can all be recorded. The names of the system log events are self-explanatory—you can look at them and decide if you want to know about each one. I always tend to turn on the ones that tell me who's been on the system and what things they've touched, but it's up to you. MPE/XL records the events that it wants to know about whether you do anything or not, so there aren't any requirements from that end. Of course, the more events that you turn on, the faster you create log files, and the more disk space they will occupy—and there's never enough disk space. Log files are those files in `PUB.SYS` that look like `LOG####.PUB.SYS`, where `####` is some number. If you have lots and lots of them, either your system is going up and down frequently (you get a new log file each time you boot), or no one has been purging the old ones off. If it is the latter case, somebody out there in disk marketing is thanking you.

One common problem that you might encounter is when you turn on all sorts of logging events...and find that your log files don't have anything worth mentioning in them. You go back into SYSGEN, see that the events that you picked are all on, and start muttering about murder and mayhem. While in SYSGEN, you notice that event 100, 'System logging enabled,' is off. Turning on this master switch, keeping and rebooting, begins to fill up your log file with:

```
(OUTPUT)13:06/154/INVALID PASSWORD FOR "MANAGER.SYS," DURING LOGON ON LDEV #62.
```

and other useful messages. Event 100 is like the main circuit breaker: If it is off, only the emergency lighting (the log events that MPE/XL keeps on) works. If it is on, both the regular and the emergency lights are on.

User logging is mostly outside the control of SYSGEN. Only the number of user logging processes and the number of users per logging process can be controlled. I guess if you need user logging, you know what to do with these parameters.

The MISC Configurator. The MISC configurator, like its name suggests, contains a lot of miscellaneous information. Originally, this configurator was called LIMIT, because it was intended to have all of the operating limits, including system table sizes. You remember system tables, don't you? MPE V/E had lots of them, and SYSDUMP asked you the size of all of them:

```
:sysdump *t

ANY CHANGES? y
VERSION = G.03.05.?
MEMORY SIZE = 4096 (MIN=256, MAX=8192)?
I/O CONFIGURATION CHANGES?
SYSTEM TABLE CHANGES?
MESSAGE CATALOG CHANGES?
SOFTDUMP COMMAND CHANGES?
LOGGING CHANGES?
DISC ALLOCATION CHANGES?
SCHEDULING CHANGES?
SEGMENT LIMIT CHANGES?
SYSTEM PROGRAM CHANGES?
        oops, I wanted SEGMENT LIMIT CHANGES! Break, abort, try again...
:sysdump *t
```

With MPE/XL, a standard table interface was introduced that allowed the operating system more flexibility in determining sizes for itself, so there isn't as great a need for specifying limits. But nature abhors a vacuum, so other configurable items were invented. These things weren't really limits, but they certainly weren't IO, LOG, or SYSFILE either—and there were too many to put up there with the TAPE command, and, well, they ended up here. But since they weren't really limits, the configurator name was changed to miscellaneous, or MISC for short. Most of the parameters in MISC can be left alone, at least initially. In fact, every default configuration shipped, from Series 922 to Series 980, has the same MISC values in it.

The SYSFILE Configurator. The SYSFILE configurator contains information to manage the files that go onto a system load tape (SLT). In general, only those things necessary to boot and configure the system should be on the SLT. The rest of the files should be on a STORE tape. If you follow this line of thought, there won't be much reason for you to be making changes in this area. One thing that you should know about this configurator is that AUTOINST runs the SYSGEN SYSFILE configurator against every configuration that it can find in order to update the list of files to be dumped to tape so that it matches the current operating system. Otherwise, if you created an SLT from a configuration that had the release 2.0 list of files, but was a release 2.2 system, the system created from that SLT probably wouldn't even boot—and if it did, it wouldn't have all of the pieces necessary to correctly run the operating system. In this specific example, the programs needed for the new spooler wouldn't be present.

Running AUTOINST against all of the configuration files on the system can, however, have side effects. In the R&D labs, we created a test version of the operating system on a machine running a different version of the OS. When we updated one of these machines to release 2.2, our release 2.0 SYSFILE configurations were updated by AUTOINST to contain release 2.2 files. This left us temporarily unable to build release 2.0 systems. This won't generally be a problem outside of the

R&D lab, as it is primarily related to having configuration files for more than one operating system release on your system—but it is a side-effect worth noting.

The IO Configurator. The IO configurator is the most used configurator of the four. It's purpose is simple—to connect the system to the outside world. To do this, it defines paths, devices, and classes. It can also define members of the system volume set.

The path commands are perhaps misnamed, but "add a configuration for the cards that form a path to the device" was too long. The place where people often get confused is in knowing when to use the path commands and when to use the device commands. After all, the device is part of the path. The difference between the two is that, if you want to talk to something from MPE/XL, you need additional information, including a logical device number. One way to package this would have been to add the path for the device, and then have an additional command to say "for this path, here's some additional information"; but instead the decision was made to let the user do all the device related configuration in one command. So the rule is: "If it has a logical device number, use the device commands."

PA-RISC has a flexible, expandable, hierarchical bus structure that spans a wide range of systems. Depending on the performance of the system, the number of bus levels will vary. We see the results of this directly today in the Series 950 and above machines, which have a system memory bus, whereas machines like the Series 932 start with the midrange system midbus.

One problem with mapping SYSGEN IO components to the real world is that we don't configure the "pipes" (the busses); we configure the "fittings" (the adapters) and the "spigots" (the devices). So we talk about bus converters, channel adapters, and device adapters, which are the paths between the busses and not the busses themselves.

Fittings (adapters) are added from the source outwards. These components are added with the `APATH` command, generally specifying only the path and product number and defaulting the other parameters. The number of pieces that must be fitted depend on what the highest bus is and off of which bus the device hangs. For now, the path flows from (optionally) a bus converter, then to a channel adapter, a device adapter, and finally the device. But remember that the device must be added separately. So, depending on how many pieces exist when you start, you must do between zero and three `APATH` commands. One benefit of a hierarchical structure is that the earlier parts (like the bus converter) are probably already added for you, because someone else needed them, too.

Once you get to the device level, switch over to the device command, `ADEV`. Once again, only the required parameters—LDEV number, path and product ID—are usually needed. Classes can be added with the device commands, or they can be added with the class commands. The device must be added before the class, so do it all in the `ADEV` command, and save yourself an extra command.[1] So the path and device commands are the real workhorses of IO configuration.

There is one last group of commands in IO. The volume commands provide a subset of VOLUTIL's functionality for the system volume set. The primary use of this feature is to allow operations staff to reinstall the system without someone who can run VOLUTIL being around. This is especially nice if you are a system manager who works days plus emergencies and who has a 24-hour-a-day operations staff. Grave shift can reload the system without you having to pull an all-nighter.

Pulling It All Together. Once you have made the necessary changes, there is still the matter of getting them to take effect. After making changes in a configurator, you must use the `HOLD` command to hold onto those changes while you work in other configurators. SYSGEN will issue a warning if you try to leave a configurator without first holding. If you don't issue the `HOLD` command, your changes

[1] In hindsight, implementing classes as something added to devices was a poor design choice internally, leading to the currently troublesome limit of eight classes per LDEV.

will be lost. Once you have made all of your changes in all of the configurators, you must either make a tape, keep the configuration (in CONFIG.SYS), or both. MPE/XL documentation recommends always cutting a tape and doing an UPDATE CONFIG from it—and that's a good suggestion. But if you're in a hurry, you can just keep and reboot with START NORECOVERY, making the tape later. (Of course, if something happens and you have to update or install, you'll lose the changes that you made.) Your active configuration should always be in CONFIG.SYS, but it doesn't hurt to have a backup copy somewhere else. For example, before you start to make changes, you might issue a KEEP BACKUP command from within SYSGEN just in case. Don't keep your system configuration back in the sample group, because the next update from a factory SLT will write over it.

There are some things that will not change without an update. In particular, changes to the RINs and GRINs, to USERVERSION in the MISC configurator and to the number of user processes in the LOG configurator only take effect on an UPDATE CONFIG (or, of course, an INSTALL). By definition, the changes in the SYSFILE configurator require an update (either CONFIG or not) or an install to take effect, since that configurator deals only with the contents of the tape. Finally, changes made with the volume commands (AVOL, DVOL, MVOL) in the IO configurator will, by definition, only take effect on an install, since these commands indicate which system volumes should be initialized during the first start after install.

Activating Configuration Changes. Most configuration changes are made during system installation, but when any configuration changes have been made and kept in the group CONFIG.SYS with the SYSGEN KEEP command, it is time to use the information.

To do this, shut the system down with:

```
Control A SHUTDOWN
```

boot the system and issue

```
START NORECOVERY GROUP=CONFIG
```

While the system is coming up and after it is up, look for the correct configuration and anything that is not correct. If there are changes to be made, run SYSGEN and make any needed changes. Keep them in the CONFIG.SYS group. To test them, shut the system down, boot, and start it again, just as we did. When all configuration information is correct, it is time to continue with the next step. If this is part of an installation or update from a factory SLT, it is time to run AUTOINST. This installs the rest of the software and writes a customer SLT. Software installation is not complete until the system is updated with this customer SLT. Some of the subsystems added by AUTOINST require changes in the system NL.PUB.SYS. These changes are not complete until after an update from that customer SLT. Again shut the system down and update from the customer SLT.

Now start the system and make any other alterations needed, (SYSSTART file, welcome message, accounting information).

Release of a New Level of the System

The time will come when a system is up and running on one level of MPE/XL and a new level is released. The new release comes in the form of a factory SLT and some other tapes. Always consult the manuals that come with the new tapes. The process of updating to the new level is as follows:

1. Backup the system.

2. Make space available on LDEV 1.
3. Shut the system down.
4. Update from the new factory SLT.
5. Start the system.
6. Run AUTOINST.
7. Shut the system down.
8. Update from the new customer SLT.
9. Start the system again.

To prepare for an update from a factory tape, a full backup is good security. In case something does go wrong, a system can be recovered with a backup. Making space available on LDEV 1 is important. Without enough space, a problem or even a disaster is possible. The space on LDEV 1 is needed in case some of the files on the SLT have changed in size, or in case there are new files on the SLT. The ISL `UPDATE` command uses, by default, the `NOCONFIG` parameter. This means that the configuration files in `CONFIG.SYS` are not replaced. The configuration files in other Hewlett-Packard-supplied sample groups (for example, `CONFG950`) will be replaced.

The first start after an update is a `START NORECOVERY`. With some system releases, the format and sometimes the content of configuration files changes. When AUTOINST is run, it changes the format and content of all old configuration files on the system. It is important not to restore old configuration files after this step. AUTOINST writes a customer SLT with the changes needed for this system.

The update from the customer SLT puts the changes on LDEV 1 and makes them part of the system. Start the system again and make the system ready for use.

Managing Multiple Configurations

SYSGEN offers advanced users the ability to control multiple systems' configurations from one hub machine. Hewlett-Packard uses this capability to create and manage all of the various machine-specific configurations that are put on the factory SLT as system files. (for example `CONFG980`, `CONFG949`, and so on.) It's as simple as keeping the different configurations in different groups, but you'll generally want to use script files to keep track of things. Script files are discussed a little later.

Once you have created multiple configurations, you can propagate them by cutting a different tape for each configuration; or you can add the configuration files as system program files, as Hewlett-Packard has done. There are advantages to each setup, with the primary advantage of adding the files as system program files being that you can use the same tape for everyone. Thus, once you have set everything up just the way you want it, you can update from that one tape everywhere; however, you will then have to run SYSGEN on each system to move that system's configuration into `CONFIG.SYS`. With one tape per configuration, it's just an UPDATE, which can be done by operations.

Script Files

Script files are just ASCII files containing SYSGEN commands, with the recommended first command of `PERMYES`. (This prevents strange things from happening when SYSGEN decides to ask for

confirmation of something.) SYSGEN is safe to use in script mode: If any problems are encountered, SYSGEN does not make a tape or do a keep, and sets the JCW to a non-zero value. The scripts used in the lab to create the default configurations come in two parts: the IO part, which varies from platform to platform, and the everything-else part, since that part doesn't change. In general, this is probably a good separation. Unfortunately, SYSGEN doesn't currently support an `INCLUDE` command; so these files must either be combined outside of SYSGEN, or else SYSGEN must be run twice. We've chosen to combine the files outside of SYSGEN, which is easy given file redirection:

```
:FILE newpass=$NEWPASS;REC=-80,,F,ASCII
:PRINT firstpart >*newpass
:PRINT secondpart >>$OLDPASS
:SYSGEN config_group,,$OLDPASS
```

Doing Without

If you find a system with an incorrect set or with no configuration files at all and you want to bring the system up you can start it without configuration files. This method is usually used to recover from a disaster, but it can be used any time that it is needed. Issue the `START` command with the `GROUP=` parameter pointing to a nonexistent group. For example:

```
START GROUP=TRASH NORECOVERY
```

When the system is booting, it gets the path to LDEV 1 from the primary path stored in stable storage. It gets the path to LDEV 7 (the tape) from the alternate path and gets the path to LDEV 20 from the console path in stable storage. These are enough devices to run the system and change the configuration files or restore some from backup media. If the group in the `START GROUP=` exists, the system does not configure LDEV 7 and has only two devices configured.

Problems, Warnings, Do's and Don'ts

Always Use START NORECOVERY. To change configurations requires a `START NORECOVERY`. A start with recovery (the default) will use the configuration information from the last start with `NORECOVERY`.

Where Are My RINs and GRINs? If your system does not have the correct number of RINs or GRINs, this is what has happened: When a system is installed from a factory SLT, the group `CONFIG.SYS` does not contain any configuration information. A start with the default `GROUP=` parameter or `GROUP=CONFIG` is a start without configuration files. You will probably discover it quickly when there are only two LDEVs available. The system also has a default number of RINs and GRINs. The next start, even with the correct group and `NORECOVERY` does not change the number of RINs and GRINs. To correct this problem, either re-install and start with the correct group, or update with a customer SLT and use `UPDATE CONFIG`.

Moving Configuration Files with SYSGEN. Do use SYSGEN to manage configuration files. Do *not* use STORE and RESTORE. Do *not* move configuration files off of LDEV 1. If you have a system with configuration files on an LDEV other than LDEV 1, the system starts only with LDEV 1 configured. There are at least two ways to recover the system. One way is to issue

UPDATE CONFIG from your customer SLT. UPDATE purges the directory names of the configuration files in CONFIG.SYS and puts the files back on LDEV 1.

Another way is to start the system without configuration files. Then restore all of the configuration files (from @.CONFIG.SYS) from a backup STORE tape into another group. RESTORE will not be able to purge the files in CONFIG.SYS because they can not be opened. The RESTORE command can be used to restore them to another group. For example: create a new group TEMPCON.SYS and issue the command CHGROUP TEMPCON to change to the new group: then

```
RESTORE *T;@.CONFIG.SYS;SHOW;LOCAL
Control A SHUTDOWN
START GROUP=TEMPCON NORECOVERY
```

When the system is up and running with all system volumes online, use SYSGEN to keep the correct configuration files back in the CONFIG group on LDEV 1. At this point, your system has been recovered, and you might want to start again with START GROUP=CONFIG NORECOVERY.

Always Run SYSGEN after Datacomm Changes. Because of interactions between datacomm devices, local devices, and compatibility mode tables, SYSGEN must always be run, and the configuration kept, after making a change in the datacomm configuration. Just running SYSGEN is not enough, because SYSGEN stores information for datacomm in its configuration files, and a KEEP must be done to save that information. A START NORECOVERY must be done to make the datacomm changes take effect.

Conclusion

We have talked about what configurations are and where they come from. We have described how to change and manage configurations on an MPE/XL system. You have seen some of the more common problems, how to avoid them or at least how to recover from them. We believe that the information here will make you more effective as a system manager. You need no longer cower in fear when you have to add a device to your system. You can walk tall and proud. You are a system configuration super-star.

Bounds Analysis
or
The Poor Man's Capacity Plan!

Paper # 3128
Bryan Carroll
Hewlett Packard
19111 Pruneridge Ave. 44MV
Cupertino, Ca. 95014
(408) 447-5833

Performance of our computer systems is something we always have to have. We are always interested in ways to get more performance, but how can we tell when we are going to run out? This determination is critical to running a successful business, keeping computer users happy, and in some cases, keeping our jobs!

The traditional methods to address this problem are with an expensive sizing study or capacity plan involving lots of engineering time and money. Enter Bounds Analysis! Bounds Analysis is the simplest approach to solving the issue of computer system capacity using queueing network models. With very little work, it is possible to determine an upper and lower bound on system throughput and response times!

Bounds analysis can be computed very quickly by hand, yet provides us with valuable information about system bottlenecks, throughput, and response time. What would you give to know how much response time would change if you added 10 new users to your system? These questions and many more are answered with bounds analysis.

This paper will explore bounds analysis, the algorithms used, how to parameterize the algorithms, several uses and several examples.

Bounds analysis can provide us with a unique insight into the variables affecting performance. Have you ever been caught spending lots of time and perhaps money trying to enhance the performance of a system only to learn that performance was unchanged? This is often the result of trying to make changes to a part of the system which is not slowing the performance of the system. Bounds analysis will show us how much impact each resource is having on the performance of the system.

The algorithms used in Bounds Analysis are very easy to use and can be very simply calculated by hand. This is one of the major strengths of Bounds Analysis. Because it is simple, you can try different values which represent various alternatives you may be considering. In this way, Bounds

Analysis can help you determine the best alternative to pursue.

Bounds Analysis is most commonly applied in two situations. First, when evaluating the impact of changes to an existing system. The algorithms can be parameterized to reflect the performance of an existing system. Then various changes can be made to the parameters to reflect possible changes to the existing system.

The second of the two most common applications of Bounds Analysis is to predict the performance of a system that does not exist. When a new application is being developed or a new system is being acquired, estimates of resource usage can be made to parameterize the algorithms. The resulting graphs can be used to predict response times and throughput for systems that do not even exist!

The idea of addressing performance of software before it is actually written is an emerging field. It is referred to as Software Performance Engineering (SPE) and is the topic of an excellent book by Dr. Connie Smith. The title of the book is Performance Engineering of Software Systems published by Addison Wesley in 1990. The book contains several chapters which can help you arrive at estimates to parameterize the Bounds Analysis algorithms.

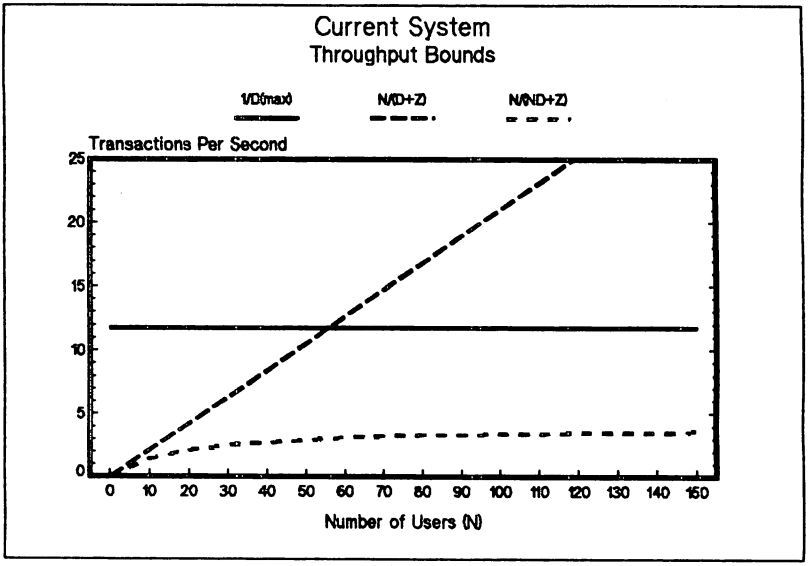
A quick note for those of you who may already be familiar with Bounds Analysis. There are two different types of bounds that can be computed, Asymptotic and Balanced System Bounds. Balanced System Bounds algorithms provide a slightly tighter bound on both response times and throughput. Asymptotic Bounds algorithms are easier to compute and they apply to a wider class of systems. The remainder of this paper will only deal with Asymptotic Bounds.

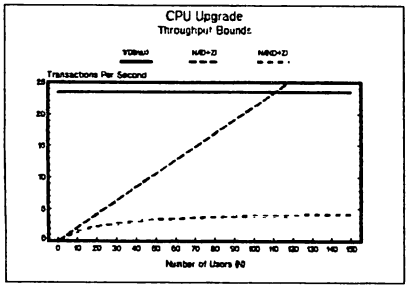
What does Bounds Analysis Look Like?

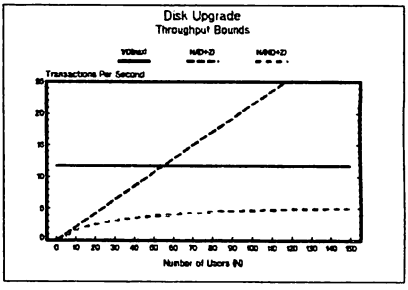
Let's take a look at what Bounds Analysis can do for us. A throughput curve is represented on a graph where the Y axis represents some relevant rate (usually transactions per second). The X axis varies the number of users in the system and is often called the population. The basic shape of a throughput curve starts at some low number and grows rapidly. When the critical resource becomes saturated, the throughput curve will flatten until there is no increase in throughput as users are added. Since we know this basic shape, all we need to know is how fast the curve rises at first and at what point it flattens out.

Assume you are considering the impact of two different changes to your system, upgrading the CPU or adding a faster disk. The following three throughput graphs show the

changes that can be expected. The lines are labeled with the algorithm used to generate the data, but do not get distracted with this notation now. We will discuss this later. The first graph shows the system as it exists today. The second graph shows the result of upgrading the CPU to a processor that is 50% faster. The third graph shows the result of adding a faster disk to the current system.







The actual throughput curves for the above systems will start off at zero and follow the N/(D+Z) line fairly closely. Throughput will continue to rise until it approaches the intersection with the 1/D(max) line where it

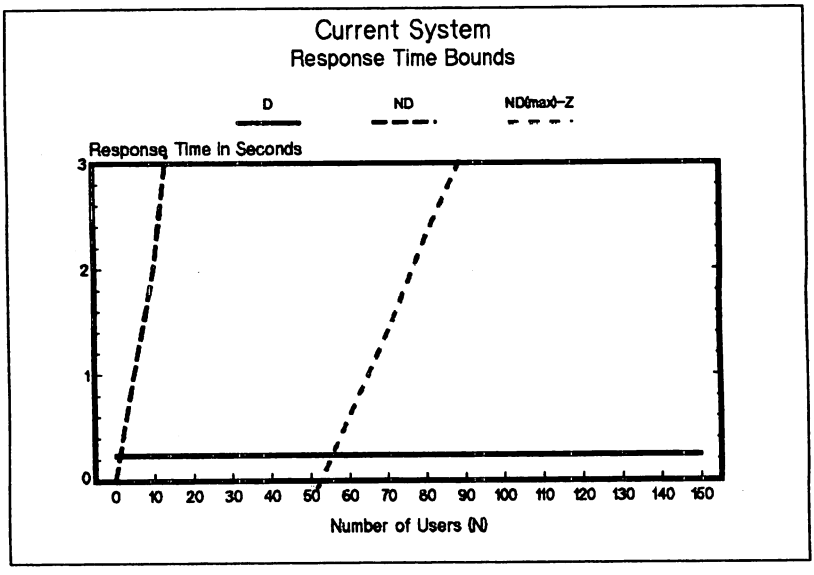
will begin to flatten. Therefore, the first two lines on the graph represent upper limits for throughput. This represents the best case and is known as the optimistic bound on throughput.

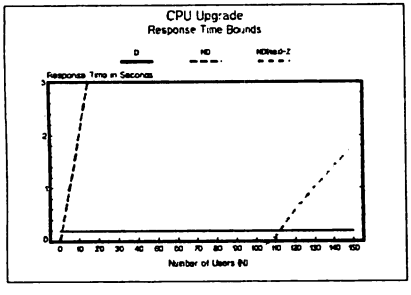
The lower line (N/(ND+Z)) represents the lower limit or the worst case for throughput. It is consequently called the pessimistic bound on throughput. This algorithm assumes that every transaction must wait at every resource behind every other transaction in the system. This is not very valuable and we will therefore not be concerned with this line through the remainder of the paper.

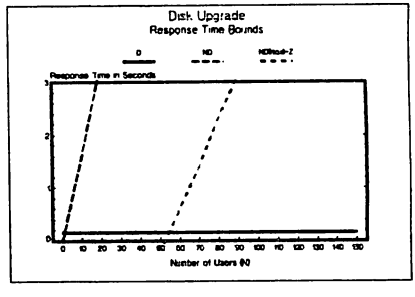
The point at which the critical resource becomes saturated is represented on the graph by the intersection of the N/(D+Z) line and the 1/D(max) line. In the current system example, the optimal throughput can be reached with about 50 or 55 users. Throughput will not increase much beyond this point since it is bounded by 1/D(max). The CPU upgrade alternative will move this optimal throughput value to about 110 users or a 100% improvement. The Disk upgrade alternative will actually cause the optimal throughput value to fall slightly BELOW the current system. We can conclude that the best throughput can be obtained by upgrading the CPU rather than the disks.

Response Time Example

The following three graphs show the same results for response times. Again the lines are labeled with the algorithm used to generate the line. Do not get side tracked on this notation. We will discuss it next. Response time curves are represented on a graph with the Y axis indicating average response time. The X axis is the same number of users as represented on the throughput graph. The basic shape of a response time curve, like the throughput curve, starts off at some low value. It then slowly rises until it reaches the same critical resource saturation point and then it goes up rapidly. Therefore, all we need to know to bound response times is the initial value and where the response times start getting high very fast.







The actual response time curves for the above systems will start off at some minimum value very near the line labeled 'D'. It will increase very slowly until it approaches the line labeled ND(max)-Z where it begins to rise very quickly. These two lines represent the best case for response time and are called the optimistic bound on response times.

The other line, ND, represents the worst case for response times and is therefore called the pessimistic bound on response time. Like the pessimistic bound on throughput, this algorithm assumes that every transaction must wait at every resource behind every other transaction in the system.

This is also not very valuable to us and we will not be using it throughout the remainder of the paper.

As in the throughput bounds, the critical resource saturation point is defined by the intersection of the two optimistic bound lines. This saturation point will be the same on the throughput and response time bounds since we are representing the same system.

As you can see, the changes to both throughput and response times are most dramatic when upgrading the CPU. The reason for this is that the CPU is the critical resource on this system.

Algorithms

We have ignored the actual algorithms up to this point to illustrate how valuable the bounds graphs can be. Now we will follow through with the promise of how simple the graphs are to generate. Notice that the optimistic bounds (the ones we are interested in) are all straight lines. The only math you must know is how to substitute values into a very simple equation and plot the line on the graph. We were conveniently able to ignore the curved pessimistic bound lines which are only slightly more difficult to plot.

Drawing these lines breaks down into remembering the basic equation for a line from algebra. The basic equation is:

$$Y = mX + b$$

where:

Y is the value to plot on the Y axis
m is the slope of the line. The higher the value, the steeper the line.
X is the value to plot on the X axis
b is the Y intercept. This is the value of Y when X is 0.

Enough of the algebra review. Just stay with me a little longer and you will see how simple it is.

Parameters

There is really just one small set of resource demands you must derive to develop the bounds graphs. These values are the service demands at the various devices. The other values needed are a total of the service demands and the maximum of all the service demands.

The notation used to represent these service demands is D(k) where k is a particular resource. For example, a typical system will have a D(cpu) to represent the service demand at

the CPU. D(disk1) would represent the service demand at device disk1. You will have as many D() values as you have resources that need to be represented in the bounds analysis. You may want to define a D(dc), for example, to represent the service time at a data communications device.

The service demand is the time the device is busy working on a transaction. A typical service demand might be D(cpu) = 40 ms. This indicates that the CPU was busy 40 milliseconds servicing a transaction.

One special case of a service demand is the think time. This is represented by the letter Z. This value will allow the algorithms to determine how many of the users are thinking and how many are requesting service from the system.

The service demands used for the current system in the example above are as follows:

```
D(cpu)   = 85 ms.
D(disk1) = 20.1 ms.
D(disk2) = 20.1 ms.
D(disk3) = 20.1 ms.
D(disk4) = 20.1 ms.
D(disk5) = 20.1 ms.
D(disk6) = 20.1 ms.
D(disk7) = 20.1 ms.
D(dc)    = 20.1 ms.
Z        = 4.5 secs.
```

Two special cases of service demands are significant and must be obtained from the individual service demands. They are the maximum single service demand (D(max)) and the total of all the service demands (D).

The maximum service demand is the highest value of all the D(k). In the example above, D(max) = D(cpu) = 85 ms is the maximum single service demand. This resource demand is very significant in that it is the service demand at the critical resource. This resource will be the limiting factor on both the throughput and response time bounds graphs.

The total of all the service demands is just a simple sum. D = D(cpu) + D(disk1) + ... D(dc). This value is very significant for the response time graph. In the best possible case, response time can only be as good as the sum of all the service times at all the devices used in the transaction, or D.

Other Notation

There are a few other notation definitions you will need to know to understand the algorithms. The X axis value for

both graphs is the number of users or the population. This value is represented by the letter 'N'.

The Y axis value for the throughput graph represents the rate at which work is being done. This is normally a transaction rate per second and is represented by the letter 'X'.

The Y axis value for the response time graph represents the time in seconds of the average response time. It is sometimes referred to as the residence time of the transaction in the system. The time a transaction spends requesting and receiving service from the various devices in the system is the time the transaction is resident in the system. This response time or residence time is represented by the letter 'R'.

We have mentioned already this notion of a critical resource reaching its saturation point. This is often referred to as the bottleneck resource. Since this point on the graph is very important, we use special notation to represent it. The symbol N* ('N' star) represents the value of N (the population) where the critical resource is saturated. N* is very important and tells us how many users a system can support before the system begins to suffer delays because a device is being saturated.

Throughput Bounds

The optimistic throughput bounds can be defined with two algorithms. The first algorithm will limit throughput until the critical resource becomes saturated. This algorithm is X = N/(D+Z) and is stated as the throughput (X) is equal to the population (N) divided by the sum of the total service demand (D) and the think time (Z). Plot this line by selecting values of N and plugging them into the algorithm to arrive at a corresponding value of throughput (X).

The second algorithm that limits throughput considers the critical resource saturation and therefore involves D(max). The algorithm is X = 1/D(max) and is stated as throughput (X) is the inverse of the highest service demand. Notice that this algorithm does not consider the population (N) and therefore will not change as the population changes. This will be a line parallel to the X axis that crosses the Y axis at a value 1/D(max).

Given these two algorithms, we can then compute the population where the critical resource saturates, N*. This value is where the two throughput (X) values are equal and can be represented as 1/D(max) = N/(D+Z).

Example

Let's look at another example of how to draw a throughput bound graph. Let's say that for a particular application we are writing, a typical transaction will do ten data base reads and two data base writes. Additionally, the transaction will use process handling to spawn a son process to do much of this work.

We must translate this transaction definition into service demands. We can consult our local data base expert or the Image Handbook for estimates or write a quick program to time these operations on a sample data base. We may be able to conclude from this that a data base read call required 5 milliseconds of CPU time and 15 milliseconds waiting for disk. The data base write averaged 15 milliseconds of CPU time and 45 milliseconds waiting for the disk.

The service demand represented by the process handling can be derived in the same way. Let's say that we wrote a quick program and measured 200 ms of CPU time and 650 ms of disk service time to do the process handling.

With the above environment, we can derive the service demands as follows. The demand at the CPU, D(cpu) would be ten times the CPU time per data base read, plus two times the CPU time per data base writes, plus the CPU time for the process handling. In other words:

D(cpu) = 10 * 5 + 2 * 15 + 200 = 280 ms.

The demand at the Disks, D(disk) would be ten times the disk wait time for data base reads, plus two times the disk wait time for data base writes, plus the wait time for the process handling or:

D(disk) = 10 * 15 + 2 * 45 + 650 = 890 ms.

This value represents the service time of all the disks combined. We need to break the disk service times down by the number of disks that will be used with this application. Let's say that we were using eight disk drives, so the service demand at each of the eight disks could be approximated by dividing the total by eight. In other words, D(disk1) = D(disk2) = D(disk3) = D(disk4) D(disk5) = D(disk6) = D(disk7) = D(disk8) = D(disk) / 8 = 890 / 8 = 111.25 ms.

> **Note:** MPE XL has been designed to scale with the speed of the processor. This means that the disk subsystem will not become the critical resource in most interactive workloads. Our bounds analysis calculation can take advantage of this by

selecting a number of disk drives that will make the service demand at each disk lower than the service demand at the CPU. This will accurately represent most MPE XL systems.

In addition to the above service demands, we may want to add 10% to the service demand at the CPU to take care of the local processing required to set up the process handling and the data base calls. Now let's add a think time of say 10 seconds. Therefore, the parameters we need to create the throughput bounds graphs is:

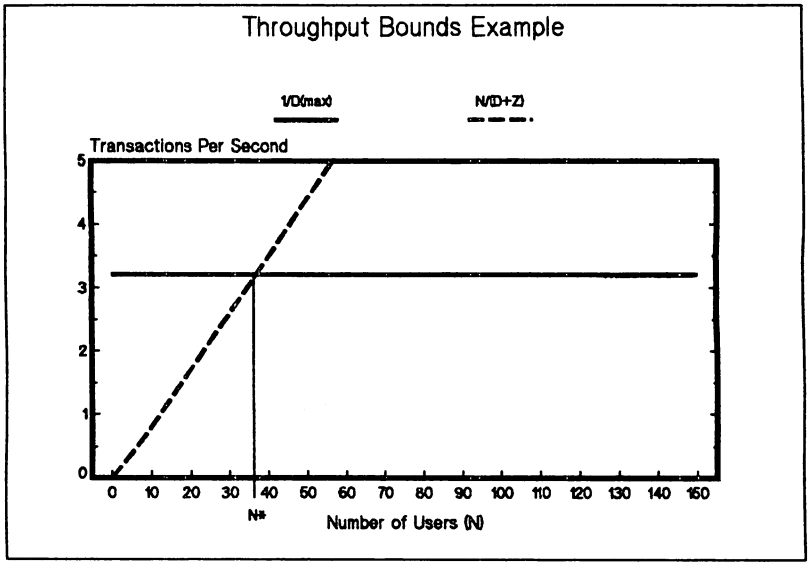
```
D(cpu)   = 280 ms + 10% = 308 ms
D(disk1) = 111.25 ms
D(disk2) = 111.25 ms
D(disk3) = 111.25 ms
D(disk4) = 111.25 ms
D(disk5) = 111.25 ms
D(disk6) = 111.25 ms
D(disk7) = 111.25 ms
D(disk8) = 111.25 ms
D(max)   = D(cpu) = 308 ms
D        = D(k) = 1198 ms
Z        = 10000 ms (10 seconds)
```

Now we have everything we need to draw the throughput bounds. Lets start with the optimistic bounds from 0 users through N*, the number of users at the saturation point of the critical resource. The critical resource will be the resource with the highest service demand which is the CPU in this case. This algorithm is throughput (X) equals the population (N) divided by the sum of the total service demand (D) and the think time (Z) or X = N/(D+Z). Compute a few points to put on the graph by substitution and convert from transactions per millisecond to transactions per second (multiply by 1000).

X = N/(D+Z)	X	N
	0	0
	.89	10
	2.7	30
	4.5	50

The optimistic bounds for users above the saturation point does not vary with the number of users (N). This bound will be a line that parallels the X axis with a throughput (X) value of 1/D(max). In our example this throughput value is 1/D(max) = 1/308 ms = 3.2 transactions per second.

Here is what our throughput bounds graph looks like.



Since we know the general shape of the throughput curve (starting low, rising quickly until approaching the 1/D(max) line, then flattening), we can approximate it given the above bounds. What we have learned is that the saturation point of the critical resource (CPU in this example) is reached with around 35 users. We will not be able to increase the throughput of the system be adding more than about 35 users.

Response Time Bounds

The optimistic response time bounds can be defined, as with the optimistic throughput bounds, with two algorithms. The first algorithm will limit response time from 0 users through the number of users needed to saturate the critical resource, N*. This algorithm is really just a single value which is the best case response time will every be and is represented by D. Recall that D is the sum of all the resource demands at all the devices. This line on the graph will parallel the X axis and cross the Y axis at a response time value of D.

The second algorithm will limit response time beginning around the saturation point of the critical resource. This algorithm is in the form of the equation of a straight line (y=mx+b). The algorithm returns the response time (R) which

is the result of multiplying the number of users (N) by the largest service demand (D(max)) and subtracting the think time (Z) or R = ND(max) - Z.

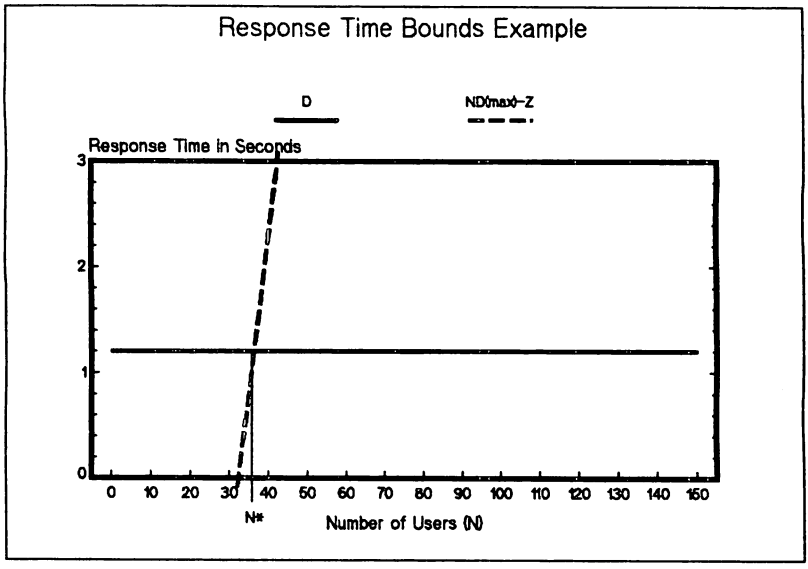
Now that we have these two algorithms, we can again compute the population where the critical resource saturates, N*. This value is where the two response values are equal and it will be the same as the N* computed for the throughput values. It can be computed with the response time algorithms by setting them equal to each other as in D = ND(max) - Z.

Let's use the same example we defined above to generate a response time bound graph. The bound for the number of users from 0 to N* is defined a D which from the table above is 1198 ms.

The optimistic bounds for users above the saturation point is defined by R = ND(max) - Z. Compute a few points to plot on the graph by substitution and convert response time from milliseconds to seconds.

R = ND(max) - Z	R	N
	-10	0
	-.7	30
	5.4	50
	8.4	60

Here is what our response time bounds graph looks like.



Again, we know the general shape of the response time curve (starting near the 'D' line, rising slowly until approaching the ND(max)-Z line, then rising quickly) and can approximate it given the above bounds. We can see that response time will get very bad very quickly beginning at around 30 users. Response time will be unreasonable if we need to support more than 30 users.

Saturation Point

We have already defined the saturation point of the critical resource and how it limits throughput and response times. We can go one step farther, however. Draw a smooth throughput or response time curve on the above graphs. The throughput and response time values that correspond to a population of N* is quite a bit less (less throughput, higher response time) than the value delivered by the algorithms.

This reflects the characteristic of queueing systems where significant queueing delay (waiting in line to use the resource like the CPU) will be experienced BEFORE the device is 100% utilized. The smooth line we drew should accurately reflect the actual throughput and response times we can observe on this system. Therefore, the number of users this system can support with good throughput and response time is

less than N*. N* represents the number of users it would take to consume 100% of the critical resource.

So how much below the N* value do we want to be? The answer to this, of course, is it depends. The optimal number of users will depend on what response time expectations your users have and how steep the vertical throughput (X=N/(D+Z)) and response time bound (R=ND(max)-Z) lines are. If the lines are very steep (close to paralleling the Y axis) a smooth curve will start rising long before reaching the N* value. A flatter curve (close to paralleling the X axis) will remain close to the optimistic bound near the N* value.

Analysis

We have learned that the system we defined will support something less than 35 users. The closer we get to 35 users, the faster throughput will flatten and response times will go up. What can we do if we need to support more than 35 users on this system? We might be able to purchase a different system or modify the application design.

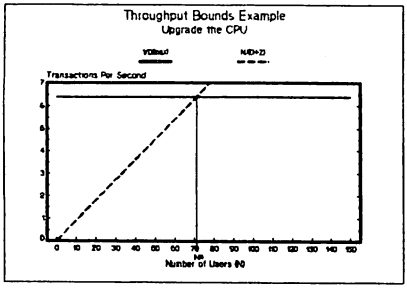
Re-Size the CPU

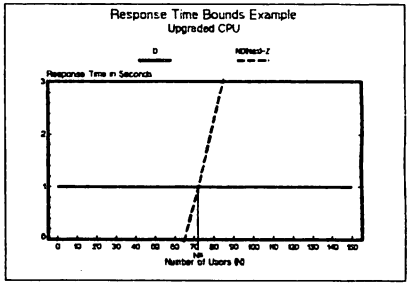
The critical resource in our example is the CPU, so if we can increase the speed of the CPU, we should be able to support more users. Let's go back and redefine the Bounds Analysis parameters to reflect a faster CPU.

Let's assume the next CPU we will consider is twice as fast as the first one. All we must do is reduce the demand at the CPU to reflect this change. Here is what our new parameters would look like.

```
D(cpu)   = 308 / 2 = 154 ms
D(disk1) = 111.25 ms
D(disk2) = 111.25 ms
D(disk3) = 111.25 ms
D(disk4) = 111.25 ms
D(disk5) = 111.25 ms
D(disk6) = 111.25 ms
D(disk7) = 111.25 ms
D(disk8) = 111.25 ms
D(max)   = D(cpu) = 154 ms
D        = D(k) = 1044 ms
Z        = 10000 ms (10 seconds)
```

We have cut the demand at the CPU in half. D(max) has also been reduced by the same amount and the total demand D, has also been reduced. The new throughput and response time bounds look like this.





We have been able to significantly increase N* which means we will be able to support more users. The slope of the vertical lines is also much lower meaning the throughput and response times will not degrade as fast when we approach N*.

Modify Application Design

Another alternative we may have is to modify the design of the application. If we examine our description of the application, we might be able to eliminate the process handling. The overhead of process creation can be eliminated if we can move all the code into one program. In this case, we can significantly reduce resource demands.

The new CPU demand (D(cpu)) will be ten times the CPU time per data base read plus two times the CPU time per data base write. We can eliminate the 200 milliseconds of overhead required by the process handling. Our new D(cpu) is:

$$D(cpu) = 10 * 5 + 2 * 15 = 80 \text{ ms.}$$

As before, we may want to increase D(cpu) by about 10% to take care of any local processing. This would increase D(cpu) by 8 ms to 88 ms.

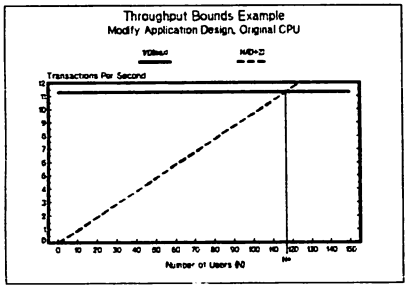
The demand at the disks (D(disk)) would also be significantly reduced. The new total would be ten times the disk wait time for the data base reads plus two times the disk wait time for the data base writes. The process handling overhead is again eliminated. Our new D(disk) is:

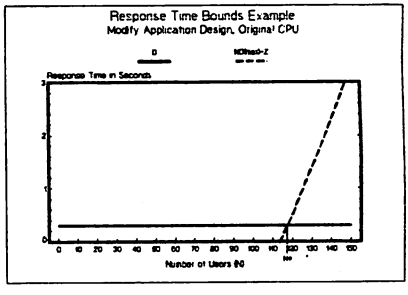
$$D(disk) = 10 * 15 + 2 * 45 = 240 \text{ ms.}$$

As before, this service demand would be spread among the available disk drives. In this example, we have eight disks, so an approximation of the service at each disk is D(disk) / 8 = 240 / 8 = 30 ms. Here is what our new service demands look like:

```
D(cpu)   = 88 ms
D(disk1) = 30 ms
D(disk2) = 30 ms
D(disk3) = 30 ms
D(disk4) = 30 ms
D(disk5) = 30 ms
D(disk6) = 30 ms
D(disk7) = 30 ms
D(disk8) = 30 ms
D(max)   = D(cpu) = 88 ms
D        = D(k) = 328 ms
Z        = 10000 ms (10 seconds)
```

We have now cut the service demands at the CPU and disk by more than 300%! Here is what our new throughput and response time bounds look like.





We have again been able to significantly increase N* which means we will be able to support even more users. In this case, the change in the application design will enable us to support more than 3 times more users! And we were able to make this determination before any code was written! The slope of the vertical lines is also much lower meaning the throughput and response times will not degrade as fast when we approach N*.

Bounds Analysis Limitations

Bounds Analysis is a very useful tool to have, but it does have two limitations worth mentioning. First, resource demands must be independent of the load on the system. In other words, the service demands must be the same with one or 100 users on the system. For most resources this is not a problem. Potential conflicts might arise when users are sharing a resource (say a data base structure) and must lock it before access can be granted. The time this lock is held may vary depending on how many users are on the system.

The other limitation is that all system activity must be generalized to one set of parameters. In others words, a single average D(cpu) must apply to the entire system. This is usually not a problem, especially in the two examples we discussed - sizing a system and predicting performance of a new application.

Note: There are Bounds Analysis Algorithms to address multiple workloads on a single system. These algorithms are more complex, however, which would wipe out one of the primary advantages of Bounds Analysis - simplicity.

Summary

Bounds Analysis is very powerful. Most of its power is in its simplicity and ease of calculations. A simple spread sheet can easily be created to generate the graphs.

The most difficult aspect of Bounds Analysis is obtaining the correct parameter values. These, however, are also not overwhelming. Since the calculations are so simple, initial guesses can be made at the parameters and refined as more information is acquired. Good resources are available to help estimate parameters in various technical papers and books. One excellent source is Dr. Connie Smith's book entitled Performance Engineering of Software Systems. Of course, the most accurate data can be obtained by writing simple programs to time various events.

References:

Quantitative System Performance, E. Lazowska Et.al., Prentice-Hall, Englewood/ Cliffs, New Jersey, 1984.

Performance Engineering of Software Systems, Connie Smith, Addison-Wesley, 1990.

Introduction

Over the past decade, the field of natural language processing has undergone a dramatic transformation driven by the availability of large-scale corpora and the development of neural network architectures capable of learning distributed representations of linguistic units. Where earlier approaches relied on hand-crafted rules and symbolic grammars, contemporary systems typically learn statistical regularities directly from data, with remarkable success on a wide range of benchmark tasks.

This shift has not been without controversy. Critics have argued that models trained on text alone can never acquire genuine understanding, since they lack any connection to the world that language describes. Proponents counter that sufficiently large models exhibit behaviors that are functionally indistinguishable from understanding in many practical settings, and that the philosophical question is less important than the engineering one.

Related work

Early work on distributional semantics established the principle that words appearing in similar contexts tend to have similar meanings. Subsequent refinements introduced dense vector embeddings, contextualized representations, and attention mechanisms that allow models to weigh the relevance of different parts of the input when producing each output.

Model	Parameters	Training tokens	Benchmark score
Baseline-S	125M	2.5B	61.4
Baseline-M	350M	7.0B	68.9
Baseline-L	1.3B	28.0B	74.2
Baseline-XL	6.7B	140.0B	79.8

Methodology

We trained all models on a filtered subset of a publicly available web crawl, removing duplicate documents and applying a language identification filter to retain only English text. Tokenization was performed using a byte-level subword vocabulary of 50,000 entries. Optimization used the AdamW algorithm with a cosine learning rate schedule and a linear warmup over the first 2,000 steps.

Results

Scaling the model from 125M to 6.7B parameters yielded consistent improvements across all evaluated tasks, with no evidence of saturation at the largest scale we tested. Performance on reasoning-heavy benchmarks improved more rapidly than on memorization-heavy ones, suggesting that additional capacity is being used productively rather than simply storing more facts.

Conclusion

These results support the view that scale remains a reliable lever for improving model quality, and that the gains from scaling are not limited to surface-level fluency but extend to more structured forms of reasoning. Future work will investigate whether the same trends hold for multilingual and multimodal settings.

Paper #3129

SUPPORTING AN NS/3000 NETWORK

Steve Beasley
Hewlett-Packard Company
Midwest Sales Region
2205 East Empire Street
Bloomington, Illinois 61701
309/662-9411

Introduction

For users of the HP3000, our definition of a network has changed drastically over the past seven or eight years. In 1984, many of us might have defined a local area network as two HP3000's in the same room with a cable connecting the INP's. Wide area networks meant having modems between two locations, and non-HP connectivity was primarily an RJE link to the corporate mainframe. Configuration was defined in the I/O section of Sysdump, and the DS/3000 software was fairly simple to operate and use.

The "network" was not that complicated during those days. When problems occurred, a quick check of the cables or the modems was always a good first step. If those were fine, the software configuration could be verified rather easily, and "tuning" the network was not really an option. Most importantly, the stable environment of our DS/3000 networks rarely created more than an occasional support headache.

As our networks grew to include real 802.3 LAN's, we got our first taste of NS/3000. The hardware was fast, the software was easy to implement, and the configuration was best left at the factory-defined defaults. Many sites began utilizing X.25 networks, but those networks were generally based upon the DS/X.25 software. Life in the network was still pretty simple and functionality was rather limited.

Networking seems to have exploded with functionality over the past few years, increasing our usage and dependence on our internal and external networks. Our LAN's and WAN's now include multiple topologies, with bridges, routers, various backbones, and various protocols. TCP/IP has become a requirement for most networks, and that means NS/3000 for the HP shop. Even for those environments that are limited to HP3000 communications, the migration to MPE XL has precipitated the move to NS/3000 and its TCP/IP protocols.

Along with this tremendous increase in functionality and connectivity comes an increase in the responsibilities necessary to support the network. From the MPE XL system's perspective, the responsibilities include network planning, network configuration, and network troubleshooting.

Network Planning

The most critical aspect of any project is the planning stage, which generally receives the least amount of time. This is particularly true with networks, since many networks begin simple and continuously expand to include new applications at various sites. However, when planning is given the appropriate emphasis, the following areas deserve attention:

TOPOLOGY

The network topology, or how the nodes will be connected, is critical to the success of the network. Topologies may include 802.3 LAN's, X.25 packet switches, extended LAN's with bridges, as well as Ethertwist networks and HP Routers. The topologies selected for the NS/3000 network are a key factor in determining the cost, performance, hardware configuration, and supportability of the network.

SUPPORTABILITY

Special attention must be given to the level of support required by the network topology. The ability to support the NS/3000 software will be somewhat dependent on the hardware components of the network. Network management for each of the components (switches, hubs, routers, etc.) should include configuration, diagnostics, and statistics reporting. If those network management features are not available remotely, then the network plan must address the need for some kind of on-site assistance when configuring or troubleshooting the remote locations.

DOCUMENT THE NETWORK

One of the most critical elements of supporting any network is the level of documentation available about the network. The documentation should specify all network-related addresses including X.25 addresses, IP addresses, and any hardware addresses being used. Additionally, all configuration parameters (line speeds, packet sizes, timer values, window sizes, etc.) should be appropriately documented. This documentation is valuable not only when implementing the network,

but also as a troubleshooting aid. For instance, no one wants to spend time trying to determine the particular address of a particular network component when fifty users are screaming that the network is down. The list of items to document may vary from one network to another, but the more complete the documentation, the easier it is to support the network.

The planning elements mentioned above will be crucial to one's ability to actually support any network. A well-planned network is worth the extra effort and expertise required to do the planning, even if that means utilizing outside consulting resources.

Network Configuration

Configuration of a network, a PC, or a mainframe generally consists of two distinct processes. First, the person responsible for the configuration must "understand" and determine what the appropriate configuration parameters should be. The second process ensures that the configuration is properly implemented and/or distributed to the appropriate locations (PC's, mainframes, network components, etc.)

For an NS/3000 network, the configuration values reside in the NMCONFIG file in PUB.SYS and are accessed via NMMGR. (On an MPE/V system, the configuration file is in the NSCONF.NET.SYS file.) Since the job of determining the appropriate values will have a direct impact on the stability of the network, the network administrator needs to take whatever steps are necessary to understand the configuration parameters.

With an NS/3000 network, this first step should probably include understanding some concepts about the TCP/IP protocols. One of the best reference sources for this understanding of TCP/IP is the book Internetworking with TCP (2nd edition) by David Comer. While one may not need to read this book cover-to-cover, it is a good source for looking up various aspects of the TCP or IP protocols. Additionally, if the network utilizes any X.25 links, the book X.25: The PSN Connection from Hewlett-Packard provides an excellent reference on the X.25 Recommendation. In addition to these reference materials, the NS3000/XL NMMGR Screens Reference Manual provides a good description of most of the appropriate values.

As part of the effort to determine the "right" NMCONFIG parameter values, the network administrator needs to consider a few details under each of the following sections:

LINK

The "link" section of the NMCONFIG file is used to define which hardware link(s) will be used by the networking software. For each link used by the network, a corresponding link name must be defined. The only significant parameter under this section is the LINK.linkname field, which has the physical path descriptor for the LANIC or PSI card. This is one parameter that must match the system configuration, so don't guess! The best way to get this value is to ask the Customer Engineer when he installs the card. Otherwise, the network administrator will need to run SYSGEN to get the path or ask someone else.

LOGGING

The best values for the "logging" section are the factory-defined defaults. These values may need to be altered when troubleshooting, which will be discussed later. For normal operations, there may be two exceptions to the default values. First, if the console operator is easily alarmed or very irritated by unusual messages, it may be better to reduce the console logging messages. This may include limiting console logging to the "internal errors" for both Transport (sub0003) and for Network Services (sub0006). These classes of errors should provide adequate notification of network-related problems.

The second situation which may warrant a change of the default logging values is for educational purposes. By setting all values to "Y" for console logging, a significant number of logging messages will be displayed on the console. These messages provide an excellent description of what is happening within the code. Default values are recommended after this "education" is completed.

Logging will be discussed in more detail as part of the troubleshooting section. However, unless one of the two scenarios mentioned above exists, the default logging is appropriate.

NODENAME

Configuring the local nodename is the most simple step in the configuration. However, this field is important if the local system has multiple network links, since this nodename is associated with all of the attached networks. For example, the $BACK

feature is a method of pointing to the "local" system when accessing a "remote" system. File equations on the remote can reference $BACK as a means of accessing the originating (local) system. However, if the "local" system has two links (LAN and X.25, for example) that local nodename is associated with both links. To make the $BACK feature work as expected, the network directory must be carefully constructed. This will be discussed in more detail within the Network Directory (NSDIR.NET.SYS) section.

NETXPORT

The "netxport" screens contain three major sections, including 1) global, 2) gprot, and 3) NI.

The "global" section is used primarily to define the "Name Search Methods". If the network is LAN-based, then the factory default is fine. However, if your network has multiple links and the network directory is defined so that each node is specified, then it is better to change the search order so that the directory is first. This will ensure that a connection takes the intended route.

The "gprot" section can provide the most trouble when configuring the network, for two reasons. First, several TCP-related items are defined under the "gprot" section. These parameters can have a significant impact on network stability if not set properly. The second reason for the trouble serves to complicate the first: these values are shared by all networks on the particular system. Therefore, the TCP-related values configured for the system will be used by LAN, X.25, and point-to-point. These three network types can have drastically different characteristics, so the network administrator must, in essence, configure for the worst case. This might involve setting the TCP retransmission timers to accommodate a very busy X.25 network. These values may be inappropriate for the LAN and could actually cause a slight degradation in throughput. However, setting the retransmission timers to accommodate an efficient LAN could easily result in the X.25 network connections being disconnected. In order to configure a reasonable compromise, the network administrator must understand the various parameters and how his network reacts to those parameters.

The "NI" section allows for the configuration of each "network interface". For MPE XL, this can include one interface for LAN, multiple interfaces for X.25 and point-to-point (router), and an NI for loopback. Configuring the NI's for LAN, loopback, and point-to-point are rather concise. The X.25 NI can have an enormous amount of data associated with it, depending upon the size of the X.25 network. Although the NI is essential, the values are generally straightforward.

In addition to configuring the NMCONFIG file, the network administrator must also configure the network directory. The network directory is a powerful component of the NS network and should receive a thorough analysis before its implementation into the network. All systems will fall into one of two categories: single network systems or multiple network systems. The network directory for single network systems is easy. For example:

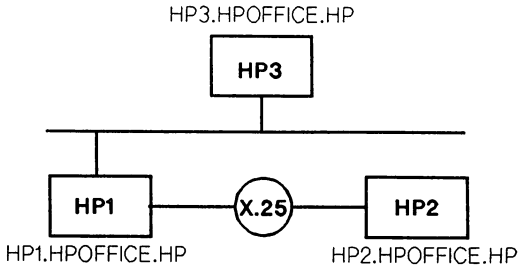
1. HP3000 system is connected to one LAN. The network directory is not required. If present, the directory will probably have one entry for each system on the LAN.
2. HP3000 system is connected to one X.25 network. Although the directory is required, the directory can have one entry per each system on the X.25 network. Each system would most likely have only one node name.
3. HP3000 system is connected to one router network. The directory is required, but would be similar to the simple directory for the X.25 network.

However, for systems with multiple network links, the directories can be complex. The network administrator has a few decisions to make with regards to how each node will be referenced. For example, please consider the three scenarios outlined on the next pages in figures 1, 2, and 3.

1. Figure 1 has one system (HP1) with a LAN link and an X.25 link. HP2 is connected only to the X.25 network, and HP3 is connected only to the LAN network. In this scenario, the network directories can be identical on all three systems and each system can have only one nodename.
2. Figure 2 now has HP2 as a member of both the LAN and X.25 networks, just like HP1. There are now two possible paths between HP1 and HP2. The directories can remain identical, with each system having only one nodename, but the network user has no control over which network is used.

FIGURE 1

LAN Network = C192.006.002 nnn
X.25 Network = C192.006.001 nnn

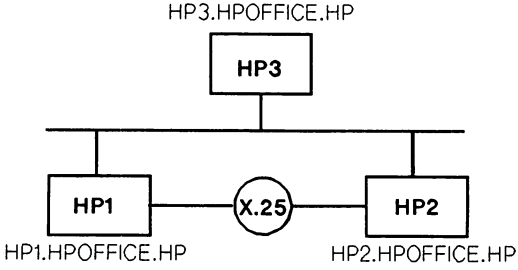


NETWORK DIRECTORY (All Systems)

Node	Type	Address	Address Key
HP1.HPOFFICE.HP	1	192.006.002 001	none
HP1.HPOFFICE.HP	3	192.006.001 001	HP1
HP2.HPOFFICE.HP	3	192.006.001 002	HP2
HP3.HPOFFICE.HP	1	192.006.002 003	none

FIGURE 2

LAN Network = C192.006.002 nnn
X.25 Network = C192.006.001 nnn

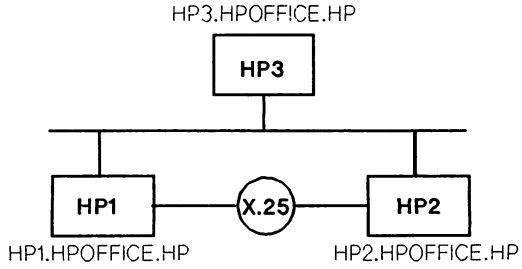


NETWORK DIRECTORY (All Systems)

Node	Type	Address	Address Key
HP1.HPOFFICE.HP	1	192.006.002 001	none
HP1.HPOFFICE.HP	3	192.006.001 001	HP1
HP2.HPOFFICE.HP	1	192.006.002 002	none
HP2.HPOFFICE.HP	3	192.006.001 002	HP2
HP3.HPOFFICE.HP	1	192.006.002 003	none

FIGURE 3

LAN Network = C192.006.002 nnn
X.25 Network = C192.006.001 nnn



NETWORK DIRECTORY (All Systems)

Node	Type	Address	Address Key
HP1.HPOFFICE.HP	1	192.006.002 001	none
HP1X.HPOFFICE.HP	3	192.006.001 001	HP1
HP2.HPOFFICE.HP	1	192.006.002 002	none
HP2X.HPOFFICE.HP	3	192.006.001 002	HP2
HP3.HPOFFICE.HP	1	192.006.002 003	none

3. Figure 3 is similar to Figure 2, but HP1 and HP2 now have multiple nodenames, HP1X and HP2X. The network user on HP2 can specify "HP1" if he wants to access HP1 over the LAN. If, however, he wants to use the X.25 connection, he specifies HP1X. This configuration is particularly valuable when testing or supporting multiple network systems in that it allows one to test applications over both network links and ensures that the correct network is being utilized.

   NOTE: This scenario is where the local nodename becomes significant. For instance, if $BACK is specified between HP1 and HP2, whichever nodename is configured as the "local nodename" will determine the return path for the $BACK. So, if HP1X calls HP2X and specifies a $BACK command, the $BACK will actually use the LAN to return to HP1. (The lab is currently modifying this "feature" to rely upon the originating system's IP address, not its local nodename.)

Once the network directory and the NMCONFIG file have been properly configured, the network administrator must decide how to manage that configuration. There are basically two alternatives: 1) Configure each system uniquely or 2) use the same NMCONFIG and network directory on all systems.

In a small network of maybe five or fewer systems, each node can be realistically configured uniquely. This means the network administrator must either logon to each system and type in the configuration values for the NMCONFIG and the network directory, or he can have someone else perform that task, using the network documentation. Even in a very small network, this method is prone to errors. These configurations have dozens of parameters, many of which are easily mistyped. This decentralized approach will work for small networks, although it is not easy to manage.

For large networks, a centralized method of configuration is essential. The chance of incorrect configurations gets astronomical when the number of network nodes and the number of "network administrators" increases. Also, while various configurations will "work", troubleshooting a network with multiple variations of the NMCONFIG and network directory becomes a support nightmare. (Obviously each node must have its own unique name and address, but all other parameters should be identical whenever possible.)

In order to accomplish this centralized configuration, the network administrator has two options. First, the pattern configuration file can be built (including the directory),

then copied to the various nodes. The changes unique to each node can then be made by hand. The other alternative is to use the maintenance mode interface within NMMGR. This somewhat "unknown" feature allows the configuration of the NMCONFIG and the network directory via batch mode. By using this functionality and a little programming, the entire configuration process can be completely automated. Some programming will be required in order to "customize" each batch job with the unique nodename and address for that system. The effort required to set up this batch process is a small price to pay for the automation of these configuration files.

Configuration management is not necessarily an easy task, but it is essential to the ability to support an NS/3000 network.

NETWORK TROUBLESHOOTING

Even the most well planned, well documented, and thoughtfully configured network is vulnerable to problems. The NS/3000 software does occasionally have a bug, the communication lines occasionally falter, and hardware components have been known to fail. So, until the perfect, totally reliable network is invented, part of our job will continue to be troubleshooting.

Fortunately for those of us who support the networks, our ability to troubleshoot an NS/3000 network has been greatly enhanced during recent months. A recent patch from the Information Networks Division in Cupertino provides the release of a set of troubleshooting tools, called NETTOOL. (These tools are officially released on MPE XL 3.0 and beyond.) NETTOOL resides in NET.SYS and is accessed by typing NETTOOL.NET. This new set of tools comes with an excellent help facility and many useful features. The key functionality for supporting an NS/3000 network resides in the following commands: resource, config, nameaddr, ping, status, nmdump, and x25stat.

RESOURCE

The resource display is similar to an unsupported utility on the MPE/V systems, but provides more documentation on the individual resources. The DISPLAY command will cause the entire resource table to print on the screen. The last column indicates whether this value is acceptable or not acceptable. Acceptable is indicated with a ":)" and unacceptable with a ":(". (When tilting this paper to the right, the markings resemble a smile or a frown.) If the user needs a detailed expla-

nation of a certain item, he can type "detail", followed by "item" and will then be prompted for the item number. Once again, type "DISPLAY" and now a good description of that item will print to the terminal. This is an excellent tool to help determine if certain parameters are configured too low. (NOTE: the GPROT MSG POOL currently displays "unacceptable". This will be fixed in a future release.)

CONFIG

The "config" option allows the user to obtain a listing of the NETXPORT section of the NMCONFIG file without having to execute NMMGR. As most network administrators have learned, NMCONFIG can be easily modified, thus requiring the validation process. The "config" option eliminates this unnecessary access into NMMGR and is an excellent tool to introduce to network operators, without worrying if they might accidentally alter the NMCONFIG file.

"Config" will also let the user display the network directory and provides an option to compare two configuration files.

NAMEADDR

For anyone who has had to restart the network in order to correct an entry in the cache, this new functionality is greatly appreciated. Within the "nameaddr" command, the cache can be displayed in various ways. The "mapping" option will display each IP address and its mapping to an 802.3 hardware address or an X.25 address key.

Within this command is the ability to delete an entry from the cache, which can eliminate the need to restart the network because of an error in the cache or because someone decided to alter a node's address.

When troubleshooting, this "window" into the cache can provide considerable insight as to potential problems on the network.

PING

While rather simple in its functionality, the new "ping" command can be very useful to determine if a device is reachable on the network. If, for example, a router is installed on the network, the HP3000 can "ping" the router using its IP address. If the device responds, the addressing must be correct and the physical connection is working.

STATUS

The "status" command is, obviously, designed to provide status information at the link level, as well as for each network interface configured. This command will provide some useful information regarding the node itself, the hardware links, as well as information relative to each network interface.

NMDUMP

The NM logfiles can be one of the most useful troubleshooting tools available on the system. The information in the logfiles is determined by the values given under "logging" in the NMCONFIG file. The "nmdump" allows the user to read those logfiles. An enhancement has been implemented which provides a summary display of the logfile, without requiring the user to view the entire file. (The NMDUMP file previously resided in PUB.SYS. The version with NETTOOL provides the new summary option.)

During certain problem scenarios, HP support personnel may ask that the customer turn on specific log classes (such as, "informative" or "detailed" events). These messages can then be used by the labs to isolate problems.

X25STAT

For sites that use X.25, the features available with "X25stat" are invaluable for providing the necessary level of support. Data can be obtained on which links are used, which VC's are assigned to the links, the amount of traffic, and which nodes are connected, plus much, much more.

For most network support personnel, the information NETTOOL provides will be all that is necessary to support the NS software. Additionally, this group of tools can provide significant information on various hardware components.

For those networks that include X.25 connections from the MPE XL system, the DTCManager workstation is a useful tool for evaluating the status of the X.25 card (the SNP). Most of this functionality is intended to provide hardware-oriented statistics via the PC. However, for first level troubleshooting, the X25 Site Management (under MONITORING) and the Show Status (under Diagnostics) will provide some useful data. The X25 Level 2 and Level 3 statistics under Show Status provide information that will assist in solving X.25 related problems.

SUMMARY

In order to support an NS/3000 network, numerous factors must be considered. Even with the best planning, solid configurations, and good troubleshooting tools, the unique qualities of each network will dictate that the network administrator (or support personnel) will constantly be searching for ways to improve the support of the NS/3000 network. Also, the items above relate primarily to those networks that use NS/3000 as the communications protocol. As more and more networks expand to include a variety of communications, the tools needed to support those networks will need to expand, as well. Hopefully, the concepts defined above will help in the on-going effort to provide the best possible network support.

Abstract

Paper #3130

Native Mode Spooler - What does it mean to you?

Gary Fletcher
Technical Consultant

Hewlett Packard
24 Inverness Place East
Englewood, CO 80112

(303) 649-5750

With MPE XL release 2.1, the compatibility mode spooler was replaced by a Native Mode Spooler (NMS). Many MPE XL and MPE V/E users are apprehensive about migrating to this new spooler because they do not understand the impact that this new spooler may have on their system. This paper will discuss some of the features and benefits of the NM spooler and tell you what you should be aware of when migrating to the new spooler.

With the native mode spooler, the file system creates spoolfiles as ordinary disk files. A new account, HPSPOOL, will be created to handle spoolfiles and spooler processes. The NMS maintains three kinds of permanent disk files in separate groups:

Type of File	**Group.Account**
Input spoolfiles	IN.HPSPOOL
Output spoolfiles	OUT.HPSPOOL
Checkpoint files	device.HPSPOOL

An input spooler reads data from its device and uses that to create an input spoolfile. The data may consist of one or more batch jobs, data files, or any combination of the two. Input spoolfiles are private files, meaning they are only accessible to a user running in privileged mode. They are not printed, but are used strictly as input for other processes. The system creates input spoolfiles by the STREAM command or by a spooler process controlling a spooled input device. Input spoolfiles have the format Innnn.IN.HPSPOOL where nnnn is the spoolfile identifier (SPOOLID).

An output spooler processes output spoolfiles- files which were created by a user accessing a spooled output device such as a printer or plotter. Output spoolfiles can be either private or non-private files. Output spoolfile names have the format Onnnn.OUT.HPSPOOL where nnnn is the SPOOLID.

Two spoolfile directories, referred to collectively as the SPFDIR, link the spoolfiles to the spooling subsystem. If output spoolfiles are not linked to the SPFDIR they can not be printed or listed with the LISTSPF command.

The checkpoint file is a companion to the output spoolfiles. The checkpoint file help the spooler recover from device problems such as power failures and paper jams. When a spoolfile does not print completely, the next spooler process that prints it, on the same device, uses the checkpoint file. Checkpoint files are kept in device name groups in the HPSPOOL account. Their names have the format Cnnnn.device.HPSPOOL where nnnn is the SPOOLID and device is the device name, such as LP.

INSTALLATION

With MPE XL 2.1 and subsequent releases, the NMS is part of the fundamental operating system (FOS) and is installed on the system during a INSTALL or UPDATE. Additionally, be sure to check with your account assigned SE or the Response Center and install any applicable patches.

The new account (HPSPOOL) will be created as part of the FOS installation. An account manager, MGR.HPSPOOL will also be created. However, it should never be necessary to log on as that user and you should never allow any user to log on to this account. Therefore, both the user and the account should be passworded.

The HPSPOOL account, its groups (OUT, IN, etc.), and its user (MGR) should never be altered or deleted. Doing so may result in an inoperable spooling system or may crash the system. Rebooting the system will rebuild the accounting structure.

CONFIGURATION

Since spoolfiles are normal MPE XL disk files in an ordinary account structure, the configuration for NUMBER OF SECTORS PER SPOOLFILE EXTENT and MAX NUMBER OF SPOOLFILE KILOSECTORS does not apply and has been deleted from the SYSGEN utility. You may control the amount of disk space allocated to spoolfiles by varying the HPSPOOL account file space limit. You may limit input and output spoolfile disk space usage independently by adjusting the IN and OUT group file space limits. The default file space limits set for the HPSPOOL account and its groups is unlimited file space.

If you do limit the directory file space on HPSPOOL or any of its groups, and you encounter this limit at the time you are creating a spoolfile, all spooling queues will be globally disabled, and the following message will be displayed on the console:

> "ALL SPOOLING QUEUES HAVE BEEN GLOBALLY DISABLED DUE TO A FILE SPACE LIMIT ON THE HPSPOOL ACCOUNT OR ITS GROUPS. USE THE OPENQ @ COMMAND TO GLOBALLY ENABLE THE SPOOLING QUEUES WHEN THE CONDITION HAS BEEN CORRECTED."

SUPPORTED DEVICES

The native mode spooler requires a tape drive as the only device supported for input spooling. However, the following output devices are currently supported:

HP 2680 and HP 2688 laser page printers.

HP 256x CIPER protocol printers

Serially connected printers such as the LaserJet series and the HP 293x series.z

Serially connected plotters.

The NMS does not support the HP 2608S.

Note for the OFFSET= page parameter of the SUSPEND, RESUME and RELEASE options of the SPOOLER command, a page is defined as follows:

For the HP2680 and HP2688: a physical sheet (which may contain one or more logical pages)

For CIPER protocol devices: a physical sheet

For serial printers: the OFFSET option is not reliable (except for OFFSET=1 or OFFSET=0, the beginning of file). No error or warning message is generated if it is used on such devices; however, results are unpredictable.

SECURITY

The HPSPOOL account will be built as part of system startup, if it does not already exist. The default security levels are as follows:

HPSPOOL Account:	(R,A,W,L,X:ANY)
IN Group:	(R,A,W,L,X,S:ANY)
OUT Group:	(R,A,W,L,X,S:ANY)
Device Groups:	(R,A,W,L,X,S:GU)

where R is read, A is append, W is write, L is lock, X is execute, S is save, ANY is any user, and GU is group user.

Never allow any user to log on to the HPSPOOL account, password both the HPSPOOL account and its MGR user.

Since spoolfiles are normal MPE XL disk files, non-private spoolfiles can be accessed according to normal MPE security guidelines, as follows:

If you have SM capability, you may access any non-private spoolfiles. This means you can read, delete or alter a spoolfile using either NMS commands and intrinsics or standard MPE XL commands and intrinsics.

If you have AM capability, you may access any spoolfile whose creating user is in your account.

If you are the creating user, you my access spoolfiles that you create.

If you have read access to non-private spoolfiles, you may store and restore them with the STORE and RESTORE commands, respectively. If you have write access, you may purge them using STORE with the ;PURGE option.

Users with sufficient capabilities can use any editor that supports variable length record files to read and edit spoolfiles. The editor should only be used to browse the spoolfile. If you edit a spoolfile and save it, the saved file is no longer a valid spoolfile. The system does not let you overwrite the original spoolfile in OUT.HPSPOOL. However you can overwrite an unlinked spoolfile in a group to which you have access. Note when modifying or saving a spoolfile with a text editor it is possible to corrupt the internal file format. This may cause unexpected results when the file is printed.

You may not browse or edit input spoolfiles.

The ;PRIVATE option generates a spoolfile that may be accessed in privileged mode only. Private spoolfiles may not be saved or copied. They may only be purged, printed or (within limits) altered using the SPOOLF command. The PURGE or COPY commands may not be used on private files. To create a private spoolfile simply add ;PRIVATE onto a file equation for a spoolfile:

```
FILE PRINTME;PRIVATE
```

or, add ;PRIVATE directly to the JOB command:

```
JOB MYJOB;PRIVATE
```

The COPY command allows the copying of non-private output spoolfiles. The new file is not linked to the spoolfile directory.

You may use PURGE to purge a non-private output spoolfile. The PURGE command cannot be used on a private spoolfile nor can it be used on any file which it does not have exclusive access.

You may rename spoolfiles using the RENAME command if you have access to them. This is allowed only with spoolfiles that are not linked to the spoolfile directory (SPFDIR).

PROGRAM, JOB and UDC CHANGES

Normally, no program changes are necessary. If your program simply outputs to the compatibility mode spooler, it should work as always. However, if you are accessing any of the data structures used by the CM spooler or using any of the internal entry points of the CM spooler, your subsystem will require changes. This is because the NM spooler is a native mode implementation, compatibility mode data structures and entry points will not be utilized. If you have specific questions as to how this may effect you, please contact your account assigned SE or the Response Center.

When utilizing 3rd party spooling packages, please contact your 3rd party software support representative for compatibility information. Listed here are most of the 3rd party spooler packages and their suppliers:

Product Name	**Supplier**
UNISPOOL	HOLLAND HOUSE
OMNISPOOL	CAROLIAN
SPOOLMATE	UNISON
MPEX	VESOFT
SPOOLRESCUE	NSD
NBSPOOL	QUEST
RSPOOL	Unsupported

Since the SPOOK utility is obsolete with the NMS, it is important to locate references to SPOOK in job streams and UDC's and use the appropriate NMS commands instead (see SPOOLER AND SPOOLFILE MANAGEMENT).

The JOB command has been enhanced to include two new parameters, insert these where appropriate:

> **;PRIVATE** - Generates a private spoolfile that can be accessed in privileged mode only. See SECURITY for more information.
>
> **;SPSAVE** - If you use this parameter, the output spoolfile is not purged after the last copy of it has been printed. The OUT.HPSPOOL account retains the spoolfile. You can not use the SPSAVE parameter with a private spoolfile.

The system startup file (SYSSTART.PUB.SYS) may contain commands to enable spooling and to start spooling processes. These should be modified to include the new NMS commands. What follows is an example of a system startup file:

```
STARTUP
ALLOW @.@;COMMANDS=LOG
OUTFENCE 14
SPOOLER 6;OPENQ
SPOOLER 18;START
STREAMS 10
HEADOFF 18
LIMIT 5,30
JOBFENCE 7
WELCOME SYSMSG.MESSAGE
VMOUNT ON,AUTO
COMMENT END OF SYSSTART FILE
```

SPOOLER AND SPOOLFILE MANAGEMENT:

With the introduction of the native mode spooler, new commands have been added to enhance the management and control of both the spooler processes and spoolfiles. In addition, two new utilities provide SPOOK command support. Therefore, it is important that the personnel in charge of managing the spooler process be aware of the following changes.

The following MPE XL commands available for spoolfile/spooler control, prior to MPE XL 2.1, are still supported and function almost exactly the same as they always have:

ALTSPOOLFILE	SHOWDEV
DELETESPOOLFILE	SHOWIN
HEADON	SHOWOUT
HEADOFF	SHUTQ
OPENQ	STARTSPOOL
OUTFENCE	STOPSPOOL
RESUMESPOOL	SUSPENDSPOOL

However, the native mode spooler has surpassed the capabilities of these commands by adding three new commands:

SPOOLER - Manages and controls the spooler process. You can use the SPOOLER command to start, stop, suspend and resume spooler processes, and release spoolfiles from a spooler process. Therefore, it replaces the old STARTSPOOL, STOPSPOOL, SUSPENDSPOOL, RESUMESPOOL, OPENQ and SHUTQ commands.

SPOOLF - Enables you to alter the characteristics of the spoolfiles themselves. You can use the SPOOLF command to alter the device, the output priority, the number of copies to print and whether or not the spoolfile should be saved or deferred. You may also use it to print or delete spoolfiles. Therefore, it replaces the old ALTSPOOLFILE and DELETESPOOLFILE commands.

LISTSPF - Lists information about input and output spoolfiles. The LISTSPF command can give you much more information than the old SHOWIN and SHOWOUT commands.

For complete command descriptions and parameter definitions of these new commands, please refer to the MPE XL Commands Reference Manual (32650-90003) and the Native Mode Spooler Reference Manual (32650-90166).

The general migration path from the old CI commands to the new CI commands is shown below:

CI Commands	**NMS Replacements**
ALTSPOOLFILE	SPOOLF filename;ALTER
DELETESPOOLFILE	SPOOLF filename;DELETE
HEADON	HEADON
HEADOFF	HEADOFF
OPENQ	OPENQ
OUTFENCE	OUTFENCE
RESUMESPOOL	SPOOLER filename;RESUME
SHOWDEV	SPOOLER filename;SHOW or SHOWDEV
SHOWIN	LISTSPF
SHOWOUT	LISTSPF
SHUTQ	SHUTQ
STARTSPOOL	SPOOLER;START
STOPSPOOL	SPOOLER;STOP
SUSPENDSPOOL	SPOOLER;SUSPEND

Note that the old CI commands remain, and will continue to perform the same functions as before.

Most of the commands listed above are operator commands (with the exception of SHOWIN and SHOWOUT). In order to allow general users to use these commands,

you must use the ASSOCIATE command. This command links a device class, such as LP, to an individual user on the system. Before you can be associated, the system manager must run a utility program (ASOCTBL.PUB.SYS) in order to create a device class user association table. This table defines which users may be associated with which device classes. More information about the ASSOCIATE command can be found in the MPE XL Commands Reference Manual (32650-90003).

By continuing to support the old spooler commands, the spooler and spoolfile management aspects of your NMS migration can be relatively transparent. The largest impact may be to those users of the SPOOK utility. The SPOOK utility will be obsolete with the native mode spooler. Its functions will be replaced by new spooler commands and utilities or existing MPE utilities.

Two new utilities have been created to assist in the transition from SPOOK:

SPFXFER - The spoolfile tape transfer utility will write NM Spoolfiles to tape in a format which can be read by the SPOOK utility on MPE V/E (for release G.02.B0 and after) and pre-2.1 MPE XL systems. This utility will also be able to read OUTPUT tapes created by SPOOK on these older operating systems. Therefore, it can be used to transport spoolfiles between MPE XL systems that contain the native mode spooler and MPE systems that do not. Its four commands (Input, Output, Help and Exit) are similar to commands seen in the SPOOK utility, and use the same syntax.

PRINTSPF - The print spoolfile utility displays both the data and the special overhead area of each record of a spoolfile. Thus it can be used to look at the forms control options of spoolfiles, such as page eject locations, double spacing, etc. The syntax of PRINTSPF is similar to that of the MPE XL PRINT command.

For complete command descriptions and parameter definitions of these new utilities, please refer to the Native Mode Spooler Reference Manual (32650-90166).

The migration path from SPOOK commands to the new CI commands and the new spooler utilities is shown below:

SPOOK Commands	NMS Replacements
ALTER	SPOOLF filename;ALTER
APPEND	FCOPY
COPY	COPY or FCOPY
FIND	A text editor, or a string search utility
INPUT	RESTORE or the SPFXFER utility
LIST	PRINT or a text editor
MODE	The PRINTSPF utility
OUTPUT	STORE or the SPFXFER utility
PURGE	PURGE or SPOOLF filename;DELETE
SHOW	LISTSPF
TEXT	A text editor

MAINTAINING BACKWARDS COMPATIBILITY

As permanent MPE XL disk files, output spoolfiles can be copied to other MPE XL 2.1 systems through normal STORE/RESTORE.

To maintain compatibility with pre-2.1 MPE XL and MPE V/E systems, the spoolfile tape transfer utility (SPFXFER) must be used. SPFXFER is located in the PUB group of the SYS account.

The SPFXFER utility can do the following:

> Transfer NMS spoolfiles to tape in a format that MPE XL's SPOOK and MPE's SPOOK5 can read. Note for MPE V/E spooler releases prior to G.02.B0, SPOOK5 will not be able to read SPFXFER tapes.
>
> Read SPOOK tapes created from any release of MPE XL and MPE V/E, including those created by releases prior to G.02.B0.

SPFXFER's INPUT command is used to read output spoolfiles from tape into the OUT group of the HPSPOOL account and assigns them new SPOOLIDs. The user and account of the spoolfile owner does not have to exist in the system directory, nor is it created. Note that SPFXFER does not restore user labels because NMS spoolfiles do not have them.

SPFXFER's OUTPUT command is used to store output spoolfiles to tape in a SPOOK compatible format. To maintain backwards compatibility with MPE V/E systems, if the job or session number is larger than 16,383, the system assigns a smaller number. Also, if the number of copies is greater than 127, the number of copies is reduced.

Refer to the Native Mode Spooler Reference Manual (32650-90166) for a complete description of the SPFXFER utility.

Prior to MPE XL 2.1, when spoolfiles were created they were fixed record length files. Now, on 2.1, they are variable length records. This can create a compatibility problem for processes that require fixed record length files. This has been a common concern for customers that have their spoolfiles microfiched. One solution is to use FCOPY to copy the spoolfiles to a fixed record length format. This can be done in the following manner:

```
FILE x;STD;DEV=DISC;CODE=0;DISC=###;REC=###,,F,ASCII
FCOPY FROM=Onnn.OUT.HPSPOOL;TO=*x;NEW
```

> The ;CODE=0 parameter will remove the NMS file code of 1517. The ;DISC= option is mandatory and must be specified when using the default size.

Another solution, is to use the SPFXFER utility to create a tape with fixed record length spoolfiles.

RECOVERY

The NMS uses its checkpoint files to recover devices which were in the act of printing when a device interruption occurs such as a power failure or paper jam. The success of the recovery depends greatly on the type of device:

CIPER protocol printers support page checkpoints and will recover to a specific page.

The HP2680 and HP2688 laser printers perform a silent run from the beginning of the file to the point of recovery, enabling them to resume printing at the correct page.

A serial printer has no feedback to tell the spooler of its page location, so printing may not resume at the correct page.

Since spoolfiles are permanent disk files, spoolfile recovery following a system interruption is no more complicated than it is for other disk files but does depend on the type of system startup and the state of the spoolfile at the time of the interruption:

The system preserves output spoolfiles for all system startups except INSTALL.

The system purges all input spoolfiles whenever an UPDATE or START NORECOVERY occurs.

If the system is booted with a START NORECOVERY, an apostrophe (') is inserted in each output spoolfiles associated job or session number. Ex: J'1234

Output spoolfiles that are in the CREATE state when the system is interrupted may not be completely recovered. Any data not posted to disk before the interruption can not be recovered. If no data was posted, the spoolfile is deleted.

May 1, 1991

HP Proactive Maintenance Services -- Making the Difference in Support

(paper for San Diego Interex Annual Conference in August)

Tamera Stoneburner

with thanks to John Ediger, Michael McCorquodale, Kent Ostby, Lynne Radzykewycz, and Claudia Zornow

Worldwide Customer Support Operations
Hewlett-Packard Company
100 Mayfield Avenue
Mountain View, CA 94043
(415) 968-5600

Introduction

Computers are present in almost every aspect of business today. With more and more applications and critical business functions being managed on computers and computer networks, customers have come to need, and expect, virtually 100% system uptime.

And while hardware reliability is increasing, a single hour of unplanned downtime can cost a business thousands or millions of dollars. This is why it is no longer acceptable for computer vendors to just REACT to computer problems after they happen. Customers are placing increasing value on software tools that can PREDICT and prevent costly system failures.

Hewlett-Packard recognized this situation several years ago and has been addressing the need for proactive services with HP Predictive Support on the HP 3000's and HP RemoteWatch Support on the HP 9000 Series 300 and 400 workstations.

There has been a significant increase in the number of companies creating these kinds of tools for new segments of the marketplace. Gone are the days that the big minicomputer was only product family utilizing proactive support services.

HP is regarded by industry consultants, customers, and competitors alike as the company that sets the standard for high-quality support services in the computer industry. Proactive support services such as HP Predictive and HP RemoteWatch are a key ingredient to this success.
This paper discusses how HP's proactive services continue to high levels of customer satisfaction. We'll explain how the

need for remote-access tools evolved, then explore the HP Predictive and HP RemoteWatch programs, examining how they work in the real world. We'll end with a comparison of the two programs.

Background

To prevent customer downtime, HP has long offered traditional preventive maintenance programs, as well as a variety of diagnostic tools. Although these tools provide significant value, HP engineers wanted to design more proactive diagnostic tools that would highlight potential problems in time to solve them before they occur. Such proactive tools would minimize customer downtime and expense while increasing user productivity.

HP offered its first remote-access proactive service, HP Predictive Support, in 1986 on HP 3000 minicomputers. Predictive Support flags potential problems and prevents or solves them before the problem results in unscheduled downtime -- thus transforming unscheduled downtime into scheduled maintenance. The program has been updated over time as the system platforms have changed and expanded. Late last year the company offered its second remote-access proactive service, HP RemoteWatch, to flag potential problems on HP 9000 workstations. Service contracts today include either HP Predictive support or HP RemoteWatch support software at no additional cost to the customer.

Let's look at how proactive support tools such as HP Predictive and HP RemoteWatch help HP customers.

Predictive Case Study

A large newspaper publisher has four HP 3000 minicomputers running the MPE V/E operating system (one recently was upgraded to MPE X/L). The system administrators generate daily HP Predictive status reports between 6:30 and 7:00 a.m.

chanism. The publisher contends that without the HP Predictive alert, the systems would have crashed before day's end, losing all hard-disk data.

RemoteWatch Case Study

The department of geoscience in a large university currently uses a network of HP 9000 Series 300 workstations involving 30 nodes to run ocean circulation and heatflow models of the earth. None of the ten research team members claim to be true system administrators, but by utilizing HP RemoteWatch they are able to solve minor problems quickly and avoid downtime.

The three professors on the research team typically use the system during the day, and their seven graduate students use it at night. One of those students inadvertently ran a program that re-set all of the device files. The next morning a professor received an electronic-mail message from HP RemoteWatch telling him that his tape drive was malfunctioning, implying that the previous evening's data back-up had not run. He found the student who had used the workstation, learned what happened, and thus was able to troubleshoot and solve the backup failure.

HP Predictive and HP RemoteWatch Compared

Both HP Predictive and HP RemoteWatch support services detect errors and offer proactive system administration. Currently the programs run on different platforms -- HP Predictive Support on HP 3000 CISC and RISC minicomputers, HP RemoteWatch Support on HP 9000 Series 300 and 400 CISC workstations.

HP Predictive and HP RemoteWatch Support focus on the prediction and detection of hardware errors using expert based rules. In addition, HP RemoteWatch Support also provides monitoring of configuration changes which are a major cause of problems in workstation environments.

HP Predictive forwards data via modem to the HP Response Center where it is routed to an available Response Center Engineer.

The Response Center Engineer may first dial into the customer's system via modem in order to run more diagnostics or access files not already sent to the Response Center. He or she then calls the customer back, and attempts to solve the problem by phone. If this fails to solve the problem, the Response Center Engineer sends a Customer Engineer to the site. Thanks to HP Predictive Support, the Customer Engineer will know the exact nature of the problem, and which test equipment and replacement parts to bring along, thus minimizing repair time.

HP RemoteWatch monitors system threshold error configuration for both stand-alone and clustered workstaitons. Each day it reports significant errors, configuration changes and exceeded thresholds to the system administrator via electronic mail. This information is detected through the use of predetermined activity thresholds and filtering of error messages. The messages correspond with instructions in the operating manual that tells the administrator what to do next -- whether it involves a simple configuation change or calling the HP Response Center for assistance.

When installed on a cluster server, HP RemoteWatch configures itself to the client nodes, running without user intervention. When a node or other equipment is added, the program automatically reconfigures itself.

Conclusion

Hewlett-Packard is committed to developing proactive support services to maximize your system uptime on all HP product platforms. We have been, and intend to remain the industry leader in proactive support services. We depend on you, our HP customers to tell us what you need and to let us know how we can make your computing environment more productive. Demonstrations of HP Predictive Support and HP RemoteWatch Support are currently running in the exhibit hall.

** FIRST-LEVEL PERFORMANCE ANALYSIS (USING GLANCEPLUS/XL) **
[Paper #3132]

By: Donna Fountain, HP3000 Performance Specialist
Hewlett-Packard Company, Inc.
Commercial Systems Division
19111 Pruneridge Avenue
Cupertino, CA 95014
(408) 447-1458

Solving performance problems is a valued skill. However, a more impressive accomplishment is to prevent problems from happening in the first place. This article will provide methodologies for first-level troubleshooting of performance problems on your 900 Series HP3000 Computer.

Following this article is a "First-Level Performance Analysis Checklist". The checklist encapsulates all bottleneck indicators that are discussed in this article. You can use the checklist as a handy reference when monitoring the performance of your 900 Series computer. The key to effective performance management is to monitor your system on a regular basis. You will recognize the early symptoms which will help you to avert many performance problems.

Your performance tool can be used for re-active troubleshooting. When slow-downs occur, activate your performance monitoring tool to assess workload imbalances or saturation of critical system resources. Is one particular group of users monopolizing system resources? Is a database lock preventing online users from accessing a database?

As a part of pro-active performance management, observe the resource utilization levels for CPU, memory and disk. The sooner you unveil a problem or potential problem, the more lead-time you'll have to put a solution in place. For example, you can plan for a CPU upgrade or redistribute workloads as your computer approaches capacity rather than when the CPU is saturated.

Real-time performance utilities are not intended to be used for long-term trend analysis or as logging utilities. They cannot be used to address all matters regarding application performance nor can online tools be expected to meet all of your performance

needs. For more in-depth performance analysis and additional functionality, you will need to supplement your performance toolbox with other utilities, such as utilities whose primary function is longer-term data collection/analysis, those designed for analyzing application program, etc.

For the purpose of this article, Hewlett-Packard's GlancePlus/XL is cited. However, these basic performance analysis concepts can be applied to all other real-time performance monitoring utilities as well.

** WHAT IS GLANCEPLUS/XL? **

HP GlancePlus/XL is Hewlett-Packard's online performance monitoring and diagnostic utility for HP3000 MPE XL-based computers. GlancePlus/XL runs on MPE XL Release 2.1 or later. It is incorporated as part of the Master Installation Tape (MIT) beginning with MPE XL Release 2.2. Graphical displays depict how the three primary system resources -- CPU, disk, and memory -- are being utilized. In addition, detailed information is displayed for jobs, sessions, and processes.

** HOW DOES GLANCEPLUS/XL GET ITS INFORMATION? **

GlancePlus/XL gets the majority of its information from the Measurement Interface (MI). The Measurement Interface is a module (i.e., part of the operating system code) contained within MPE XL. Measurement Interface code gathers computer performance information. All performance tools, including those authored by HP as well as our vendors, access the Measurement Interface for the purpose of collecting, formatting, and displaying performance statistics.

The Measurement Interface must be enabled or turned-on before performance data can be collected. Therefore, if your process running GlancePlus/XL turns on the MI, statistics will be initialized at zero and tabulated from that moment forward. However, if another performance tool running on the system (e.g., LaserRX) has already turned on the Measurement Interface, statistics will reflect performance-related activities that have occurred since the MI was originally enabled.

** GLANCEPLUS/XL OPERATIONAL OVERVIEW **

GlancePlus/XL consists of several screen displays including:

. a GLOBAL utilization overview;

. one screen for each of the three primary system resources -- CPU detail, MEMORY detail, and DISK detail;

. detailed data screens for JOBs, SESSIONs, and PROCESSes.

Other screens available are: an in-depth HELP facility and a FILTER screen which permits the user to display information on selected processes.

Upon running GlancePlus/XL, the GLOBAL screen is displayed. Information regarding global utilization of CPU, disk, and memory is depicted in bar format and locked at the top of the screen. Once the utility is executing, the user can freely move among all of the screens going from any one screen to any of the others. As the user moves about in GlancePlus/XL, the global utilization bars continue to be displayed and updated at the top of each core screen. This keeps timely information regarding the three primary system resources at the fingertips of the system manager.

GLOBAL Screen.

The GLOBAL Screen presents an overall view of how the system is performing. Following the global utilization bars is information regarding interesting processes. A process is defined as "interesting" if it exceeds a user configurable threshold value, is newly created, or was terminated during the last interval. Detailed information shown for each interesting process includes: job/session number, logical device number, user logon, process identification number (PIN), program name, scheduling queue, priority, amount of CPU used, disk I/O transfer rate, number of transactions during the last interval, average response time, and wait reason.

CPU DETAIL SCREEN.

The CPU detail screen provides information regarding system CPU metrics. As with other core screens, the Global Bars are displayed at the top. Next, CPU consumption for the current interval is broken down by type (e.g., session, batch, memory manager). Following is a bar which depicts system CPU usage by queue. Then there is a miscellaneous section reflecting various CPU measures, such as switch information, launch rate, and interval compatibility mode percentage. At the bottom of the display, the top CPU consumer is identified.

MEMORY DETAIL SCREEN.

System memory metrics are reflected on the MEMORY detail screen. Following the Global Bars is fault rate information. The total number of memory page faults during the interval is broken down by object type. At the bottom of the screen is miscellaneous memory manager information, such as physical memory size, memory manager clock cycle delta, and System Library Page Faults.

DISK DETAIL SCREEN.

The DISK detail screen reveals the activities of the system disk drives. As usual, the Global Bars are displayed. Next, disk performance measurement statistics are reflected for each individual disk drive (e.g., physical read rate, current queue length, and utilization %). A summary section follows which gives global disk metrics for the interval. Information regarding the pin with the highest level of disk usage appears at the bottom of the screen.

JOB/SESSION Screen.

The JOB/SESSION screen begins with a display of the three Global Bars. Next, two additional bars depict how the selected job or session is using CPU and Disk resources. Totals are accumulated for elapsed time, CPU time, and number of disk transfers completed since the measurement interface was enabled. Finally, interesting processes relating to that job or session are displayed. Details revealed regarding each process include: process identification number, program name, priority, CPU %, disk transfer rate, number of transactions, response time, wait

state, total CPU time, total disk I/O, and PIN of the job/session's parent.

PROCESS Screen.

Following the Global Bars on the PROCESS Screen are the Process Bars. The Process Bars show how the selected process is using CPU and Disk resources. The fully qualified program name is displayed. Total elapsed time, CPU time, and number of disk I/O's is displayed for the process since the MI was turned on. The State Bar shows the the process's wait state over its entire life (i.e. running time). The Last Interval Process State Bar depicts the process's wait state during the last time interval only.

Details displayed for a particular process include: current wait state, scheduling queue, priority, scheduling state, rate of terminal transactions completed during the last interval, compatibility mode percentage, native mode switches, compatibility mode switches, and response time. Additional information that can be displayed for a process includes: names of all open files, family tree, and HPDEBUG procedure trace. If SPT/XL has been purchased, an SPT/XL collection process can be initiated via a softkey to collect metrics on a currently executing process.

For further information regarding operation and functionality of GlancePlus/XL, please refer to the HP GLANCEPLUS/XL USER'S MANUAL, part number B1787-90001.

CPU BOTTLENECK INDICATORS.

The Central Processing Unit (CPU) is responsible for executing the instructions that do the work required by user and system processes. CPU usage is a good first approximation of how well a system is performing. A CPU bottleneck is a situation where demand for CPU exceeds the supply. Processes on the system are requesting more CPU than the system can provide.

There are four CPU bottleneck indicators listed on the GlancePlus/XL checklist: CPU Busy %, average switches per second, interval CM %, and average length of the current ready queue. CPU Busy % is the primary CPU bottleneck indicator. The latter three are secondary indicators which are used to substantiate a CPU bottleneck.

CPU BOTTLENECK INDICATOR: CPU GLOBAL BAR

When probing the system with GlancePlus/XL, the system manager should first note the global CPU Busy % which is the first bar appearing at the top of each core GlancePlus/XL Screen. CPU Busy is a fairly good approximation of how well the system is performing. Be sure to note the composition of the global CPU Busy bar. User activity can be comprised solely of online, solely of batch, or a combination of both.

For example, if CPU Busy is consistently 70% with only interactive users on the system and no batch activity, there is a definite possibility of a CPU bottleneck. On the other hand, if CPU Busy is consistently 100% but there is a lot of batch activity on the system, (e.g., 55%), there probably isn't a CPU bottleneck. In the latter case, batch activity probably consumes surplus CPU, since minimal demand is being placed on the system for CPU by the online users. If the majority of the CPU utilization is batch, that usually implies that there is plenty of CPU capacity left for online sessions.

Here are some RULES OF THUMB for systems where CPU BUSY is composed of predominantly online users:

* When global CPU Busy consistently hits 60%, you should start evaluating your current environment and projecting

future computing needs. At this point in time, begin planning for a CPU upgrade. Keep in mind that it is always best to have a consensus of other indicators before declaring that a CPU bottleneck exists.

* When global CPU Busy % is consistently in excess of 70%, a CPU bottleneck exists. Reexamine workload attributes and processes that are consuming the CPU. If your environment is streamlined for maximum operating efficiency and CPU Busy is in excess of 70%, it is time to order your CPU upgrade. Monitoring and advance planning can prevent many performance problems and is less stressful and more cost effective than managing system resources on a re-active basis.

* When global CPU busy % exceeds 85%, serious performance degradation is inevitable.

CPU BOTTLENECK INDICATOR: CPU UTILIZATION

CPU utilization is one of the most popular measure of system performance. To calculate the specific utilization level, it is important to sum the total amount of CPU consumed by the various system resources and online processes during the current interval: Memory Manager activities, operating system processes (e.g., Network Services), session processes, dispatcher, and Interrupt Control Stack (ICS) activities.

CPU UTILIZATION						GLANCEPLUS/XL THRESHOLD
MemMgr	+ system	+ session	+ ICS	+ disp	= CPU Busy Utilization	70%

CPU BOTTLENECK INDICATOR: AVERAGE # OF SWITCHES PER SECOND

MPE XL has a Switch subsystem which resolves differences between native mode and compatibility mode access. MPE XL transparently handles switches between modes for system routines. Native mode to compatibility mode switches are usually more expensive in terms of resource expenditures than are compatibility mode to native mode switches. When switching from native mode to compatibility mode, 32 bit addresses must be converted before they can be handled by a 16 bit address range. When a compatibility mode procedure calls a native mode procedure, this is not an issue since native mode code can access the full

address range. Thus, CPU consumption is more expensive for native mode to compatibility mode switches.

AVERAGE # OF SWITCHES PER SECOND

GLANCEPLUS/XL THRESHOLD

S/922...150
S/925...150
S/932...240
S/935...300
S/949...550
S/950...300
S/955...450
S/960...600
S/980..1000

_____/sec + _____/sec = _____
switches to CM switches to NM avg # switches per second

CPU BOTTLENECK INDICATOR: INTERVAL COMPATIBILITY MODE PERCENTAGE

The interval compatibility mode (CM) percentage depicts the percentage of time the CPU spent in compatibility mode between switches. Some of the CM time will be time spent in operating system code (e.g., RIO and message files); however, most of the major subsystems are in native mode (e.g., TurboIMAGE, KSAM, VPLUS).

For systems exceeding the threshold of 20% compatibility mode, establish a strategy and appropriate timetable to migrate any remaining compatibility mode programs and applications to native mode. In the interim, be sure to at least translate those remaining program files with the Object Code Translator (OCT).

		GLANCEPLUS/XL THRESHOLD
INTERVAL COMPATIBILITY MODE PERCENTAGE	_____%	20%

CPU BOTTLENECK INDICATOR: CURRENT READY QUEUE

This statistic represents the average number of processes that are ready to execute but must wait their turn to get the CPU. A module of the MPE XL Operating System called the Dispatcher maintains processes requiring the CPU in a priority-ordered list. This list is called the Dispatch Queue in MPE XL pre-2.1 releases.

and the Ready Queue in 2.1 and later releases. Scheduling algorithms dictate how the processes' priorities are maintained and adjusted. The Dispatcher is priority-driven and will give the CPU to the highest priority process that has requested it.

The ready queue is a measure of contention for the CPU during the current interval only. For example, if the average length of the ready queue during the current interval is 16, it exceeds the threshold of 15 and is a CPU bottleneck indicator for the current interval only. If the average length of the ready queue maintains a level of 15 or higher over a duration of intervals (e.g., one hour), it is indicative of a system CPU bottleneck.

	GLANCEPLUS/XL THRESHOLD
CURRENT READY QUEUE = _____	15

CPU BOTTLENECK INDICATOR: "TOP CPU CONSUMER" PROCESSES

A good place to check next is the list of interesting processes on the Global Screen. The interesting processes are those utilizing a large % of one or more of the primary system resources. If the preliminary indication is that a CPU bottleneck exists, the list of interesting processes will provide a good list of candidates for further examination. Details revealed for each of the displayed processes include: job/session number, logical device number, logon, PIN, program name, queue/priority, CPU%, disk I/O transfer rate, number of transactions, average response time, and wait states.

Examine the CPU % column. Is one process, one application, or one group of users monopolizing the CPU during the past interval? The CPU-TOT column in the JOB/SESSION and PROCESS Screen will unveil the total amount of CPU processor time that has been used during the life of the job or session.

If additional details are needed to understand how particular processes are using the CPU, go to the Process Screen. In addition to the typical CPU statistics such as % of CPU used during the last interval and total CPU used during the life of the process, the Process Screen shows the % of time that the process spent in a particular WAIT STATE over its entire life.

Check the PRI (priority) column to see if someone is running at higher than their scheduled queue. For example, if you observe a job with a PRI of D156, you might want to investigate why it is running at such a high priority.

FIRST STEPS IN ALLEVIATING A CPU BOTTLENECK

Improve system performance by reducing the demand for CPU. If the program that you've identified as consuming a lot of CPU is running in compatibility mode, migrate the program to native mode. Remember that optimal program performance is achieved when running in native mode on a PA-RISC system. If factors such as application size, availability of source code, or lack of time prevent immediate recompilation to native mode, the compatibility mode program files should be translated via a utility called the Object Code Translator.

Check the CS queue for such things as running reports and compiling programs. These types of activities should really be accomplished via a job stream in the DS queue. Inappropriate CS queue activities include: compiling programs, large sorts, and reporting programs. To temporarily alleviate the CPU drain, use the GlancePlus/XL command "Q" to dynamically reprioritize the queue and/or priority.

Evaluate batch jobs running in the DS queue. If possible, run some/all of the batch jobs at night instead of during the day. Reschedule those essential batch jobs restricting them to non-critical periods only.

Use the :SHOWQ command to verify the settings of the CS and DS queues (please refer to Performance Note #2 for more details). As of MPE 2.1, there is a new property of the :TUNE command called oscillate/boost. This can be enabled for the CS, DS, and ES scheduling queues to address the problem of poor long transaction response time. By :TUNEing the queues for overlap and enabling oscillation, a system manager can cause the long transactions to be reset to the base of their scheduling queue when they decay to the limit of the queue.

Other ways to reduce the demand for CPU include:

- Optimize UDCs to reduce overhead incurred during logon.
- When using process handling, recycle processes rather than spawning additional processes. ACTIVATE/SUSPEND is much more resource efficient than CREATE/KILL.
- Offload graphics, spreadsheets, etc. to personal computers.
- Streamline applications to eliminate inefficiencies. For example, use the SORT utility to presort large data files. For very small data files, an internal sort would be more efficient. Additional performance tools can assist in diagnosing application performance, such as Hewlett-Packard's Software Performance Tuner (SPT/XL).

MEMORY BOTTLENECK INDICATORS.

The third global bar reveals the overall contents of main memory. Components of the memory bar show how much memory is being used for different types of objects including: resident MPE code, stacks, extra data segments/data objects, and files. The total amount of main memory being used is expressed in megabytes and is displayed to the right of the bar. It is normal for main memory to be fully utilized even when there is little activity on the system.

MEMORY BOTTLENECK INDICATOR: MEMORY MANAGER CPU

A certain amount of CPU resources is needed for operating system management. A typical level of interrupts on the HP3000 Series 900 system consumes approximately 15-20% of total CPU resources. For example, let's say that the "M" or MPE function component of the CPU Global Bar shows 30%. 30% "M" activity minus 15-20% for interrupts leaves approximately 10-15% CPU activity attributable to memory manager/dispatcher activity. This could be indicative of a memory shortage if the majority of that 10-15% is attributable to memory management overhead.

You can use the standalone metric for memory manager CPU utilization that is reported on the CPU DETAIL screen. Memory manager CPU busy % is the first category listed in the Utilization Section which appears right below the Global Bars. The normal guideline or rule of thumb for memory manager overhead is that 8% or over constitutes memory pressure.

	GLANCEPLUS/XL THRESHOLD
MEMORY MANAGER CPU UTILIZATION = _____	8%

MEMORY BOTTLENECK INDICATOR: MEMORY MANAGER CLOCK DELTA

The Memory Manager Clock Cycle Delta is located to the right at the bottom of the MEMORY DETAIL screen. The size of the machine's physical memory (in megabytes) is to the left.

All things remaining the same, the more memory a system contains, the longer it will take to cycle through memory. Conversely, if there is less memory in a computer, it will cycle through memory more quickly.

Note that the Memory Manager Clock Cycle Delta is a new statistic which replaces the more familiar "Clock Cycle Rate Per Minute". This is due, in part, to the slower rate at which the MPE XL Memory Manager recycles through main memory. Memory Manager Clock Cycle Delta is expressed as rate *per hour*. A change in this delta of more than 25 over the period of an hour may indicate possible memory pressure.

		GLANCEPLUS/XL THRESHOLD
MEMORY MANAGER CLOCK CYCLE DELTA =	_____ per hour	25 per hour

MEMORY BOTTLENECK INDICATOR: OVERALL PAGE FAULT RATE

A page fault occurs when a program tries to reference a data object (e.g., file, database) that is not present in memory. The program suspends execution, and the operating system retrieves the needed page(s).

There are 10 different types of page faults reflected in the MEMORY DETAIL screen of GlancePlus/XL. The Overall Page Fault Rate field is a summation of these 10 types of memory faults during the current interval. A fault rate greater than 30 faults per second sustained *over several intervals* indicates memory pressure.

		GLANCEPLUS/XL THRESHOLD
OVERALL PAGE FAULT RATE =	_____	30 per second

MEMORY BOTTLENECK INDICATOR: SYSTEM LIBRARY CODE PAGE FAULT RATE

This metric reveals the total number of faults during the current interval for native mode and compatibility mode system libraries. A sustained rate of greater than two faults per second may indicate a memory bottleneck condition.

		GLANCEPLUS/XL THRESHOLD
SYSTEM LIBRARY CODE PAGE FAULT RATE =	_____	2 per second

FIRST STEPS IN ALLEVIATING A MEMORY BOTTLENECK

Memory bottleneck indicators are less definitive. However, there are a few additional GlancePlus/XL indicators which will assist in identifying memory pressure at the process level.

A wait reason of MEM as reported on the GLOBAL, JOB, SESSION, and PROCESS screens indicates that a process is waiting on either library code, program code, data, or any combination of the three to be brought into memory. These objects are brought into memory in pages. It is normal to occasionally find a process in a MEM wait state. However, if a process is repeatedly observed as waiting on MEM, this can be indicative of poor locality. Application logic may be forcing continuous branchs to other pages.

Multiple processes in MEM waits can be indicative of a memory shortage. As the demand for memory outstrips the supply, objects residing in main memory are kicked out to make room for objects needed by higher priority processes. When your process again becomes active, its needed pages must be brought back into main memory before it can run. This excessive memory management overhead causes system performance to degrade.

Many of the housekeeping rules that apply to alleviating CPU bottlenecks also apply to reducing memory usage. These include:

. Minimizing the use of UDCs and combining UDC files;

. Shutting datacomm lines down when not in use. They tie up memory and freeze it;

. Minimizing file opens, program loads, and user log-ons.

If your 900 Series system is not at maximum memory configuration, you have the option of adding an additional memory board to impact the bottleneck. You also have the option of reducing the demand which will involve application optimization for good data locality.

Always evaluate memory pressure in conjunction with CPU utilization. A saturated CPU will often exhibit memory pressure symptoms. When the CPU bottleneck is alleviated, often the memory pressure indicators will disappear.

DISK BOTTLENECK INDICATORS.

When discussing the subject of a disk bottleneck, we are referring only to physical disk I/Os. Physical disk I/Os are

those that actually require some type of interaction with an external device.

DISK BOTTLENECK INDICATOR: PHYSICAL DISK TRANSFERS PER SECOND

The Global Disk Bar reveals information about physical disk transfers happening on the system. The number to the right of the Disk Bar reports the actual number of physical disk transfers occurring in the current interval. The Disk Bar depicts physical disk transfers by category: memory management (swapping) transfers, system process transfers (e.g., datacomm), interactive process transfers, and batch process transfers.

25 physical disk I/Os per second is the guideline for assessing the disk transfer rate. This is a guideline only, as disk bottlenecks are not very common on MPE XL. PA-RISC architecture was designed to eliminate disk I/O. Thus, this threshold of 25 physical disk I/Os per second is based on limited data.

	GLANCEPLUS/XL THRESHOLD
PHYSICAL DISK TRANSFERS PER SECOND = _____	25 IOs/second

DISK BOTTLENECK INDICATOR: CPU PAUSED FOR DISK

When the CPU is in a Paused for Disk state, no CPU is being used. The system is waiting for a particular process's disk I/O to take place. During this time, the system was not able to perform any other function. This category also measures time which is spent waiting on memory manager disk I/O.

	GLANCEPLUS/XL THRESHOLD
CPU PAUSED FOR DISK = _____	25%

DISK BOTTLENECK INDICATOR: UTILIZATION LEVEL PER EACH DISK DRIVE

Below the Global Bars on the DISK DETAIL screen is the utilization statistics on a per disk drive basis. Utilization

refers to the percentage of time during the current interval that the drive was in use servicing disk I/O reads and disk I/O writes. The indicator for a disk bottleneck at the drive level is 80% utilization over a sustained period.

	GLANCEPLUS/XL THRESHOLD
UTILIZATION LEVEL FOR EACH DISK DRIVE = _____	80%

FIRST STEPS IN ALLEVIATING A DISK BOTTLENECK

The first step in diagnosing disk bottleneck symptoms is to isolate the problem down to its source. GlancePlus/XL will enable you to trace the high level of disk I/O to its origin. Begin by going to the GLOBAL screen. Scan through the list of interesting processes to identify those that are DISC-waited.

Next, look at the TRN column to reveal the rate per minute at which the <return> or <enter> key was pressed during the last interval. Transactions where the think time is less than .1 second are not counted, since they are probably hardware status replies. If the TRN rate is excessively high, investigate modems, etc. for possible broken hardware. Another event that could vastly inflate the TRN rate by causing interrupts is the setting of a manual or other object on the keyboard. The number of transactions is 0 for most batch jobs, because they generally do not read from terminals.

Finally, check the RESP column to get the average time between the <return> or <enter> key being pressed and the computer being ready to accept more input. This number will be 0.0 for batch jobs and for any sessions with no transactions occurring during the last interval. RESP is a calculated value, so don't use it as an absolute number. Instead, use RESP as a relative number and as an indicator.

For example, the value of RESP is normally between .0 and .5 for a particular process. When experiencing system performance problems, the system manager runs GlancePlus/XL and notices that RESP has increased to 2.5 for that particular process. This is a five-fold increase and can be indicative of change. Perhaps there is a bug in the application program, or the system's environment has changed significantly such as adding a new application that shares the same database.

Further investigation will be needed to determine whether or not an I/O bottleneck exists and what disk drives are being impacted. Once you have identified a list of prospective candidates, you can proceed to the JOB/SESSION or PROCESS screens to examine

DISC-TOT, the total number of physical disk transfers that were completed during the life of the job, session, or process. The PROCESS screen will provide even more in-depth clues (e.g., all open files and their level of activity).

There are several housekeeping rules that will insure good throughput on a 900 Series HP3000 system. First, insure that sufficient freespace is available on all drives. Next, archive files that have not been accessed for a specific period of time, and purge them off the system. Other good practices will also affect disk throughput: minimizing file opens and closes, good UDC strategy, PC offloading, eliminating unnecessary log-ons, etc.

LOCKS AND LOGICAL CONTENTION.

GlancePlus/XL will assist you in identifying issues that are related to resource contention. The GLOBAL screen will reveal WAIT reasons for the most resource-intensive processes during the last interval. To investigate locking/logical contention, look for processes that are waited for one of the following: SEM (semaphore), DBMS (database management system), SIR (system identification resource), or RIN (resource identification number).

LOCKING/LOGICAL CONTENTION INDICATOR: IMP % OF STATE/LAST BAR

To gain a perspective of what percentage of time a particular process spends in the various WAIT states, proceed to the PROCESS Screen. The Elapsed Time Process State bar shows the WAIT states for the process over the duration of time it has been running since the Measurement Interface was enabled. The Last Interval Process State bar reflects the same information but for the current interval only.

The DBMS wait state indicates that a process is waiting on a database management system lock (DBMS). User processes as well as the HP SQL subsystem processes will, at times, be reported as DBMS waited. If a process repetitively appears as DBMS waited, the possibility of locking contention exists. TurboIMAGE/XL subsystem locks show up as SEM (i.e., semaphore-waited).

An indicator of contention would be if the percentage of time spent in the "Q" state is greater than or equal to 15% on the Process State/Last Bars. All processes impeded by a syncronizing

mechanism are summed in the "Q" state percentage. This includes both DBMS- and SEM- waited processes.

If the "Q" category in the Process State Bar exceeds 15%, logical contention over time is established. If the "Q" category in the Process Last Bar exceeds 15%, locking/logical contention is presently occurring.

	GLANCEPLUS/XL THRESHOLD
IMPEDED % OF STATE/LAST BAR = _____	15%

FIRST STEPS IN ALLEVIATING LOCKING/LOGICAL CONTENTION

Once a contention issue has been identified, further investigation will be needed. Excessive waits may reveal that an application's locking strategy is not appropriate. Locking strategies should be carefully selected and take place at the lowest level possible to avoid monopolizing resources. For example, TurboIMAGE database locks can take place at the base, set, and item level. After a possible contention issue has been detected with GlancePlus/XL, run DBUTIL.PUB.SYS and issue a SHOW dbname LOCKS command to verify whether or not a problem exists.

Solutions to a locking/logical condition revolve around the application and its design. Possible areas of interest would be to investigate locking strategy on files or datasets. Due to the possible complexity involved additional performance tools, such as Hewlett-Packard's SPT/XL, may be needed.

SUMMARY

For those times when crisis management prevails, a performance monitoring tool is essential for immediate troubleshooting. With regular monitoring and advance planning, you will be using your performance utility to *maintain* control of your 900 Series HP3000 system rather than to *gain* control.

In either case, you need to be aware of the main bottleneck indicators. These indicators will provide you with the necessary data to make informed decisions -- both pro-active and re-active. The "First-Level Performance Analysis Checklist" is a handy one-page summary of the major resource bottleneck indicators. Please feel free to make additional copies of this checklist for yourself and others.

The "bottom line" is to monitor your system regularly to avert performance problems. By examining your computer system's resource usage levels, you will often be able to pro-actively recognize and alleviate bottleneck symptoms before they become full-blown problems. Your reward will be longer lead-times for handling many performance issues.

** FIRST-LEVEL PERFORMANCE ANALYSIS USING GLANCEPLUS/XL **

CPU BOTTLENECK INDICATORS
****

THRESHOLD

(C) *_____ + _____ + _____ + ___ + ____ = _____ 70%
MemMgr system session ICS disp

S922...150
S925...150
S932...240
S935...300
S949...550
S950...300
S955...450
S960...600
S980..1000

(CPU Busy Utilization)

(C) _____/sec + _____/sec = _____
switches to CM switches to NM

(Average # of Switches Per Second)

(C) Interval Compatibility Mode % = _____ 20%

(C) Current Ready Queue = _____ 15

*Examine the "Top CPU Consumer" process(es). Are processes critical to your site being consistently flagged as CPU waited?

MEMORY BOTTLENECK INDICATORS
****

(C) *Memory Manager CPU utilization = _____ 8%

(M) *Memory Manager Clock Cycle Delta = _____ 25/hour

(M) Overall Page Fault Rate = _____ 30/sec

(M) System Library Code Page Fault Rate = _____ 2/sec

DISK BOTTLENECK INDICATORS
****

(D) *Physical Disk transfers/second = _____ 30 IOs/sec

(C) *CPU Paused for Disk = _____ 25%

(D) Utilization level per each disk = _____ 80%

SOFTWARE/CONTENTION INDICATORS
****

(P) "Q" (impeded % of State/Last bar = _____ 15%

*Examine interesting processes that are flagged as "Q" (impeded)

FOOTNOTES * indicates primary bottleneck indicator
********* ( ) maps to GlancePlus/XL screens where data can be retrieved (C=CPU, D=disk, M=memory, P=process)

Paper # 3134

How to Win Memory and Influence CPU: A Look at MPE XL System Performance

D. Scott Pierson
Hewlett Packard Company
Commercial Education Instructor
1421 S. Manhattan Ave
Fullerton, California
(714) 758-5565

With the growth of computer automation within the the business community, it is becoming more important that management address the issue of effective utilization of system resources. When analyzing the performance of a particular machine, MPE XL systems are similar to all computer systems, in that, the factors of CPU, MEMORY, I/O, LOCKS and LATCHES, and system management techniques are usually all added together in a formula. The problem is that there are no formulas in performance. Performance tuning is not to be looked at as a science with formulas and tables to look up the answers, but rather as a form of art that must consider many other pieces of information than would fit in a formula or equation.

So, our focus must be on an understanding and appreciation of the design of the system and how to best utilize its features. Through the use of MPE commands, HP GlancePlus XL, and HP LaserRX there is much that can be done by the system manager to effectively utilize the system to its potential prior to the intervention of a consultant level performance analyst.

WHAT IS PERFORMANCE?

A computer system is one of many resources that a business may utilize to maximize profit. The computer is beneficial if it can contribute to an employee or entity to maximize their income related potential. If that potential is hindered due to the computers inability to service all of the needs placed upon it at a given time, we would consider this to be a problem with performance. An indication of a performance problem might surface in slow response time as seen by the users or delays in batch job completion, to name a few.

Before addressing the subject of performance, the system manager must be able to answer several key questions. Before continuing, it is essential to have these answers. The manager is first asked to characterize the environment. Simple answers of 80% interactive and 20% batch

will not suffice. The manager must explain "what is the function of the computer", "how does information flow through the company and the computer", "what is needed of the computer", and "what prioritized activity must complete at the sacrifice of all other activity". The number one mistake made by system managers attempting performance analysis is to immediately jump to solutions. So, they attempt to learn all that they can about the functionality and control of the computer. But, in actuality, that same effort should have first been put into an understanding of how their company does business. This is the essence to performance analysis being a form of art and not a science. All environments are different. Once the environment is understood, application of that knowledge to the limited resource will maximize its availability and consumption.

We have found the most important point that is the basis of all performance analysis, knowledge of the environment. With this as our base, the principles of performance can be applied to the MPE XL operating system.

When troubleshooting a possible performance problem, it is important to adhere to several basic concepts generic to troubleshooting. First, **NEVER ASSUME THE OBVIOUS.** This is a very common mistake made in the industry. It is easy to jump right into in depth technical solutions to resolve a problem without any success and finally come to the conclusion that the solution was something obvious, directly underneath our nose.

Second, **TO OBTAIN THE CORRECT ANSWERS, YOU MUST ASK THE CORRECT QUESTIONS.** A user's perception of a performance problem is usually completely different from that of the system manager's. If the user's expectations are properly set early, many "false alarms" can be alleviated. Make sure to ask specific "closed end" questions to obtain the needed information. Quantify the information as much as possible. For example, "What was the expected result?", "What is the current result?", and "What is the derivation between the two?". Notice the difference between the answers from these direct closed end questions and a response of "the computer seems to be running slow today." Link this information into the knowledge of the environment.

Third, **LOOK PAST THE PROBLEM WITH POSSIBLE SOLUTIONS.** When implementing a solution it is possible to cause multiple new problems as a result of the solution. In the classic MPE systems, if I/O were a bottleneck, system managers could utilize disk controller caching. This could actually cause two new problems of CPU and MEMORY if those two resources were already bound.

Fourth, **CHANGE ONLY ONE VARIABLE BEFORE RETESTING FOR THE EFFECTS.** After asking the correct questions and looking past the problem with the possible solution, only make one change to the system before retesting for the results of the change. If more than one change is implemented, there is no way to be certain which change corrected the problem or if multiple changes stalemated each other.

Now that there is a procedure in place for resolving performance fluctuations, it is time to identify the contributors and the solutions. Resources that contribute to a degradation of system performance can be characterized as "bottlenecks". The most common performance bottlenecks are CPU, Memory, I/O, Locks and Latches, and that of the system manager. In the way of solutions, all solutions can be broken into one of two categories, "Buy More" or "Use Less".

Buy more or use less has been worded many different ways to disguise what actually is being recommended. It could be called "Increase the Supply" and "Reduce the Demand", but it is the same old thing. These concepts can be applied to many different scenarios. If discussing a household budget and a difficulty in paying the bills, the same solutions are available. The individual can increase the supply of income or reduce the demand on the income. The same dollar can not be used to pay two different bills at the same time. This is one individual who has tried it.

"Use Less" is a broad category to catch all solutions that do not involve the purchase of hardware. But, what if the demand cannot be reduced or the users cannot use less of the resource? In this situation, that resource must be divided between the most important consumers competing for the resource. The system manager is right back to the need to know the environment and prioritize the activity that takes place. Now, the system manager can balance the load and time at which the consumers compete for the resource.

In the HP3000 MPE XL operating system environment, there are several software solutions that will attempt to automate the system performance tuning on behalf of the system manager. Note that the software will only be as effective as the system manager's ability to make the above decisions in balancing the resource load.

Since most environments can neither afford nor need the purchase of hardware, a review of how a system manager may use less of the resources in demand would be in order. When considering buying more or using less, it is important to compare the cost of the solution to the benefit gained from the solution. If it will cost more to implement the solution

than will be gained by the solution, it would be best to consider other alternatives or simply put up with the problem.

There is much that can be done by a system manager prior to paying for the expertise of a performance consultant. A performance consultant is not always needed. System managers will be able to save the department and HP a lot of time and money by performing basic performance analysis.

The next objective is to identify the performance bottleneck. All systems arrive with a "Free" performance tool. It is called the "Look and Listen" method. By looking at the LED display of the computer, a system manager can relatively determine the activity level of the system. The second character position of the display indicates the approximate activity level of the system. A display of "F0FF" indicates the system is 0% active or idle, "F5FF" for 50% active, and "FAFF" indicating a level of 100% active. A system manager could also look at the disk drives. If the disk is busy, the access light will be flashing continuously. With the older 793X drives, the read/write heads could be heard moving back and forth indicating the activity level on a drive. The disk could be seen moving and bouncing across the floor of the computer room. Early warnings of disk imbalances were first noticed through this method. Some system managers actually shutdown the system to re-align the disk units back into a straight line.

This is not to say that a system manager need only walk into the computer room and look at the CPU lights and disk drive movement to analyze performance. Much more is needed. But, this is a first level observation of performance. If the system manager is walking past the computer and notices that an "F8FF" is flashing in the LED display, indicating 80% utilization, it might be wise to determine what activity is taking place on the system. Or, if a user were to call and ask if the system was performing poorly and there was a "F1FF" in the LED, look first to possible causes of the user work station, or application, and not yet to the entire system. The bottom line being that the system manager needs to have a good "gut feel" for the type of activity that takes place on the computer. This will assist in providing a basis with which to compare all activity levels.

This same look and listen method can be applied to the end user. Look at what the users are performing on the system and listen to their comments. The users are what keep the computer center in demand. Eliminate the users and there is no computer center. Listening to the end users and providing the tools to make them more productive will allow the

computer facility to grow and flourish.

LACK OF RESOURCES CREATE BOTTLENECKS

A resource is anything that is in demand and is limited in supply. Performance degradation occurs when one of the system resources of CPU, memory, disk I/O, or locks and latches reaches the point at which the system can no longer perform to its requested level. This symptom is called a bottleneck. Detection of bottlenecks is difficult due to the conditions quickly disappearing or the overlap and interrelation of resources. A shortage of one resource could lead to the misuse or shortage of another, as was the case in the earlier example of MPE caching. In some cases, there could be more than one bottleneck.

The performance of MPE XL systems has been designed to scale directly with respect to processor speed, main memory availability, and workload. This implies that disk I/O be eliminated whenever possible.

THE DISK BOTTLENECK

Disk is the medium on which data is stored until requested for a process. I/O is the physical action of input or output to the disk media. If a process references a piece of data that is not present in memory, then that information must be retrieved from disk. Relatively, this is the slowest operation for the hardware to perform and is measured in milliseconds. It is important to note that in most environments, due to new enhancements to the MPE XL operating system, I/O will not be a bottleneck.

Although disk I/O is not typically a bottleneck on MPE XL systems, much can be done to reduce or optimize I/O and can result in a savings of 5% to 8% in some cases. There are many ways to reduce I/O. The use of fiber-optic disk over HPIB disk, load balancing, and reduction of fragmentation, just to name a few.

Fiber-optic interface is a solution available for several disk drives and an upgrade for others. HP-FL drives offer three main benefits over the HPIB interface. HP-FL can transfer data at 5 megabytes per second in comparison to 1 megabyte per second of HPIB. HP-FL can be located at distances of up to 500 meters from the CPU. HP-FL supports a maximum of eight disks per HP-FL device adapter. The transmission speed, although five times that of HPIB, is only a small part of a physical I/O. Of an average physical I/O of 25 milliseconds, the transfer time only accounts for approximately 1-3 milliseconds. With the transfer being such

a small part of the total I/O, a system manager should not justify HP-FL on the transfer speed alone. Second, a common limitation of HPIB has been the distance at which all disk drives for each channel are supported from the system. This distance of approximately 15 meters is calculated using both the load of the device and the total cable length. The greater the distance, the more the signal can attenuate or weaken. Since fiber optic uses light transmitted through light emitting diodes, LEDs, there is no weakening or signal loss, thus the greater distance. Finally the maximum supported devices on a fiber optic interface is eight compared to six on HPIB. In comparing the two interfaces, HP does not recommend more than four disk drives to an HPIB channel and six disk drives to an HP-FL channel without the possibility of significant performance degradation. This limitation is due to the hardware. Any more disks on the same channel and the hardware might not be able to service the requests. The request will not be lost, it will be placed in queue until it can be completed. In summary, more disk at greater distances with faster transmissions speeds. Today, only disk devices are supported on HP-FL channels.

Sufficient free disk space available on the system is extremely important in reducing I/O demand. The way in which the file system manages free space and candidate locations for the placement of new file extents is completely different from MPE. If there is a lack of sufficient free space, the file system will have to work harder in locating candidate free space locations. The secondary storage manager must fulfill the requests demanded by the volume manager for placement of data on disk. The harder the file system has to work, the greater the delay to the process. The free space must be available to both permanent and transient structures. The free space must also be distributed to multiple disk drives. If the requested disk type is restricted to only one disk drive then all requests will be queued up and processed serially through that drive.

The :DISCFREE command can be used to determine how much space, PERM or TRANS, is available as well as how much is being utilized per drive. An even balance across all drives will potentially maximize the I/O capability for the channel. The system balances disk placement utilizing the configured percentage set within SYSGEN or VOLUTIL. If the disk drives are configured similarly, the system manager will usually see an even distribution of total disk space.

Disk load balancing is most commonly referred to as the balancing of the most commonly accessed files across all volume set drives. Use the scenario of a disk drive being able to process 25 I/Os per second. Not by choice, all of

the most referenced files for an application located on that drive. The use of the application then places a demand of 50 I/Os per second on the drive. It is easy to discern that the disk drive can service only a portion of that demand. As least 25 of the requested I/Os would be waiting. If that same load were equally distributed across two or more drives, each able to handle 25 I/Os per second, then the demand of 50 could easily be met and there would be no processes paused for the I/O to complete. The theory is easy. The application is difficult. How to identify the files and distribute them equally across multiple disk drives is the most difficult part of this process. System managers must rely on their knowledge of the application mix and work load of the system. Partial backups can assist in determining which files or applications are utilized for a given day. But even that is just the files that have been modified, not the ones that have been accessed. To obtain more detailed information, the system manager could utilize system logging. Turning the FILE OPEN event type "ON" would log all accesses of files to disc. The log files could then be read to determine the most heavily accessed files. The system manager needs to know what takes place on the system but not to the granularity of how each application works, just the files it utilizes. Distribution of files and applications across channel adapters, device adapters and disk drives will assist in achieving the I/O potential for the system.

A many times forgotten or overlooked database issue that can have a significant effect on I/O is the concurrence of data within a database. The more dynamic, always changing, a database becomes, the more the data will not be sorted physically but be linked together through chains. A chain logically links a record to the previous and next record. As records are added and deleted from the database, they are no longer in physical order but now in logical order. If a sequential search of the database is requested, the I/O system will use an algorithm to perform a prefetch of additional data to that which was requested. If the ordered data is not all physically located together on disk in one location, multiple I/Os will have to be performed to retrieve what could have been retrieved in only one I/O. Thus, the need for concurrence of data within the database. This can be corrected through the process of unloading/loading the database. Most system managers have the responsibility of database administration without the luxury of database theory or training. A good majority of the escalated performance problem sites resolved found database performance to be the problem. To use another analogy, performing a UNLOAD/LOAD of the database is like changing the oil in a car. Most owners usually know that it needs to be done periodically. The owner also knows it will

be a long time before the engine stops due to the lack of changing the oil. So, many people do not change the oil until it gets black because preventive maintenance is a low priority. Dirty oil contributes to friction. Friction is the major cause of engine wear. Some cars will continue to run while others will eventually slow down and then stop. Improperly managed databases can be a major cause in system degradation but many times not viewed as significant to the system manager until it is too late. Some applications utilizing databases may continue to function just fine while others will eventually slow and become a performance problem. The system manager will have to react with database management. Proactive is certainly easier, and less stressful, than reactive.

MY DISK IS GOING TO PIECES

There are two types of fragmentation: disk and file. Disk fragmentation occurs when the free disk space for a drive is fragmented into such small pieces of disk, that it is difficult for the secondary space manager (SSM) to find candidate locations to satisfy new data storage, both permanent and transient. There is no current HP utility to alleviate this condition. MPE XL uses a concept called paging. All disk is partitioned into 16 sector boundaries (4096 bytes), or a page. Since all files and free disk utilize the same common denominator, there cannot be a segment of disk that is too small to be utilized by the SSM. If a 160 sector piece of free disk is requested and there are no pieces of that size or larger, the SSM simply subtracts 16 sectors, uses a new value of 144 sectors, and looks again. This continues all the way down to the smallest piece of 16 sectors. If none of these are available, the system must be out of free disk. Large contiguous chunks of free space are desirable to prevent this type of activity to be forced on the SSM.

File fragmentation follows the same concept. Based on the availability of free space to the SSM, as a file continues to grow, the information can either be stored in one large chunk of disk, or it can be divided, fragmented, into many small pieces of disk. When it is time to read that information, if it is in one location, fewer I/Os will be needed than if that information were stored in many different locations, smaller in size. DIRUTIL is a TELESUP utility that can be used to identify the extent of fragmentation in a file. Using FCOPY to copy from the old file into one initially allocated file extent is one method of combining multiple small fragments of a file into one contiguous location on disk. The system manager is faced with the dilemma of possibly having large chunks of disk space preallocated to a file. This would

reduce I/O activity at the expense of wasting disk space.

THE MEMORY BOTTLENECK

Before a process can execute on the computer, the CPU (Central Processing Unit) will require that all of the data resources that the process will need to begin, be present in memory. Memory is fast, limited, and expensive. There are three types of memory: CPU cache, main memory, and virtual memory.

CPU cache is the fastest and most expensive memory. This is the small amount of memory that is located on the CPU board. It stores the most recent instructions that have executed on the CPU. These are not the lines of code in your program but the computers translation of that program code. Each line of code translates into many many lines of instructions. RISC architecture, in part, was built on the premise that many of the same instructions are used over and over, thus RISC or reduced instruction set computer. If the CPU can retrieve the next instruction to be executed from the CPU cache and not request it from main memory, execution can continue without a delay. Nothing can be done to improve CPU cache other than upgrade to a larger processor which usually has a larger CPU cache.

Main memory is second in speed and cost to CPU cache. It is not as expensive nor as fast as CPU cache, but it is much faster than virtual memory. Main memory is divided into the same partitions as that of disk. Main memory attempts to keep the most recently referenced data and code available for use by the CPU. The larger the amount of main memory on the system, the better memory management can satisfy the requests placed upon it for data without requesting I/O to take place.

Some things must always remain in main memory. The operating system retains some of its structures in memory and will not allow them to be swapped out to virtual memory on disk. A user can also freeze resources in main memory. This could be very advantageous to a process if it never had to be blocked on an I/O and its resources were always memory resident. But it can also be very damaging to the rest of the system users. The amount of main memory available to be shared by all of the other processes has just been reduced. Their demand will have be satisfied with less of a resource.

HOW MUCH MEMORY IS NEEDED?

As a general rule, to predict the memory requirement of the system, the following formula could be used. Allocate 16 megabytes for the operating system and an additional .5 to 1

megabyte per interactive user. For example, if there will be 100 active users, memory configuration would require 16 meg for the system, plus between 50 meg,(for a low end of .5 meg per user) to 100 meg,(at 1 meg per user). That would be a total of between 66 meg and 116 meg of memory. Which end of this scale is best for your organization will depend on the degree to which your applications are memory intensive, how much parent/child processing is being performed, as well as how much shared code the users will utilize. Some batch applications have been known to be memory intensive and a need for substantially more memory per active batch process has encroached 8 megabytes per job. This is certainly not the norm but is worth mentioning to demonstrate that these are only guidelines and memory requirements differ from application to application. Whatever mix you chose for your environment, the vendor supplying the software will assist with preliminary estimates. If anything, a good portion of the user community is over memory configured at system purchase. This is great. This condition provides for ample growth without needing to address the memory concern too early in the system life cycle.

Virtual memory is the slowest and least expensive of the three. The operating system uses virtual addressing to give the appearance of having a large main memory system through the use of secondary storage on disk. What does that mean? The cost of main memory is high and it is limited in supply. So, we fake out the system by giving the impression that there is an unlimited supply by providing virtual memory on disk. Relatively speaking, disk is cheap. So the most recent data is kept in main memory and the oldest will typically be swapped out to the extended virtual memory on disk. If a process needs some data and the memory manager cannot find it in main memory, a disk request is posted to retrieve the information from virtual memory on disk.

A system manager can view the amount of disk currently being utilized by virtual memory through the DISCFREE command for the TRANSient space utilization. If anything, we use a lot of disk in MPE XL. Each process that logs onto the system, whether it be a session or job, will consume approximately 10,000 sectors of transient space. If parent/child processing is utilized, as in the case of menu driven applications, the child process will utilize an additional 2,000 to 9,000 sectors. To complicate matters more, if upon completion the child suspends instead of terminating, the space will not be returned until the parent process terminates. Note: transient space is only configured and utilized on system domain disk drives, not user private volume domain disk drives. There is a plus and a minus associated with this. The plus is achieved if the user

performs redundant activity while logged into the application. If the user prints many of the same reports, the child process can suspend after the printing of the first report. Thus, the process will not have to reallocate all of the table entries and resources for the code to execute again. This reduces the user wait time for each additional printing of the report. If the child process is terminated upon completion, each subsequent launch of the child will have to reallocate and obtain those entries again for the report to be generated. This minus side of this scenario is realized when the system manager sees the free disk space start slipping away. Remember, if you have free disk, the users will find a way to consume it.

So we use more disk. What problems could that produce? If there is no free space and a new process attempts to log on and cannot get its allocation as listed above, the process will not be allowed to logon until there is sufficient space available. A job will be placed into a WAIT state and a session will be told that the system is unavailable. Imagine the following scenario. A system manager partitions the disk into volume domains leaving only two disk drives to the system domain. The two drives are utilized by the operating system, system log files, software, and transient space which can only reside on the system domain drives. Free space is gradually consumed and goes unnoticed by the system manager. The manager is then requested to add an additional 50 users to the system. Everything might seem in order at first glance. But, multiply the 50 users by the 10,000 sectors minimum required for logon. Hope that the system manager had 500,000 sectors of TRANSient space available. This particular condition has caught many system managers off guard.

Whatever happened to ALLOCATE and AUTOALLOCATE? Due to the differences in the way that the Memory Manager handles the allocation of a program's code segments as well as the amount of memory that the system has available, there is no longer a need to ALLOCATE or AUTOALLOCATE a native mode program. The ALLOCATE command will still provide a benefit to compatibility mode and translated program code. The problem being that there is no command that will provide a list of the programs that have been ALLOCATED. With HP's recommendation that all code be eventually migrated to native mode, this should not have too much importance for the average MPE/XL environment.

THE CPU BOTTLENECK

The Central Processing Unit (CPU) executes programs on behalf

of the users. In a multi-user environment there will be many users all requesting the services of the CPU at relatively the same time. Users believe there is only one person that the computer is servicing. But the system manager realizes that there are multiple users with this same belief. So, the system manager must assist the computer in prioritizing the activity of the system so that all the users are relatively productive.

THE MIP MISCONCEPTION

Users enter commands and run programs. The CPU breaks that activity down into instructions that can be executed by the CPU hardware. The relative speed of the CPU is based on how many of these instructions the computer can process, measured in Millions of Instructions per Second (MIPs). Many performance specialists refer to MIPs as Meaningless Increment of Performance. A MIP rating should not be used as a measurement to compare relative performance of different computers due to the differences in the machine instruction set. Different computers complete code in a different number of instructions. Therefore, what difference does it make how many instructions it can process in a second of time? To use an analogy, it would be like comparing the speed of a car by the RPMs of the engine. If the engine of the car is currently doing 4000 rpms, how fast is the car moving? It is impossible to predict. The car could be in first gear, third gear, or even in park. MIPS is only beneficial in comparing like systems from the same vendor. Any further comparison should utilize one of the industry standard mechanisms shared with multiple vendors such as a TPC-A benchmark.

The CPU bottleneck is one area in which the system manager is afforded a lot of control. Through the use of standard MPE commands and special software products, the system manager is allowed to redistribute the processing of the CPU with many activities. There is a limit to how much processing power the system manager can distribute between the processes. Once that resource runs to capacity, the same two alternatives are available, "buy more" or "use less". An upgrade of the CPU is not always a viable solution in the early stages of system tuning for there is much that can be done to adjust a processes consumption of the available CPU. So we will focus in on the "use less" portion of the alternatives. Remember as stated before, use less many times means balance the load or distribute the consumption.

Processing can be broken into two broad categories of overhead and user. Overhead processing is impacted by the resources of memory, I/O, and management of the CPU. For example, if memory is limited, the processor will be spending

more time managing the limited resource. That time could have been spent processing on behalf of the user process. A system manager will want to monitor the amount of CPU spent on overhead and maintain levels under 20%. Anything higher may indicate other problems to be investigated. Today, this activity can only be displayed through the use of a purchased performance product.

In a multiprocessing environment, there are multiple user processes that will be in contention for the processing power of the CPU. A system manager must be able to rank or prioritize the concurrent activity of the system. It is with this list of prioritized activity, whether formal or informal, that a system manager will base a decision when two user processes are competing for the CPU at the same time. They both can't have sole ownership. It must be divided. The distribution does not have to be even. It can be if they are of equal importance. The most difficult decision is "which process is the most important" or "is it always the same priority list".

SQUEAKY WHEEL GETS THE OIL

Usually when asking a system manager to prioritize the activity that takes place on the system or to identify which process is the highest priority, a manager will respond, "that depends". It depends on the time of day, day of the week, week of the month or year, and who is complaining the loudest. This may be the case in your environment. It doesn't mean that you'll spend your entire career putting out fires. It may mean that either your environment is so dynamically changing that rules cannot be applied or the users need to have their expectations reset. A user calls and states that they need a report immediately, and it is for the president. No problem, the request can be met. Another user calls and states that they need a report, and it is for the president, and so on, and so on, and so on. There will come a time when the system manager will have to decide if the president really needs all of these reports or if the users have learned to say anything that it will take to guarantee that their report is generated quickly.

By default, all processes are considered equal. Interactive sessions and batch jobs are each placed into two separate circular queues. The queues are considered circular because they allow a process to begin with a high priority and decay to a low priority. As the process consumes CPU, its priority will lower so that other processes that are ready with an equal or higher priority may share or compete for the use of the processor. There are five queues on the computer named A, B, C, D, and E, A being the highest to E being the lowest.

The A queue is reserved for system activity. The B queue is mainly used for system activity but can also be utilized as the highest user queue. A system manager might use this queue for a logon when a performance problem is suspected. This will allow the system manager to process at a higher priority than the possible activity causing the CPU bottleneck. Extreme discretion should be used when considering a logon into the B queue. The system manager's activity in this queue is also contributing to the performance problem. In some cases, the use of the B queue to view the current system conditions will justify the degradation associated with the use of the queue. The C queue is the default for all interactive sessions and has a value between 152 and 200. The D queue is the default for all batch activity and has a value between 200 through 238. The E queue has been left as a special nondefault queue that the system manager can utilize for special activity. As processes compete for the use of the CPU, the operating system utilizes a dispatcher to rank and prioritize all requesting processes. The lower the priority number for a process, the higher the priority on the system. The default queues for sessions and jobs (CQ and DQ respectively) are not overlapped. This means that it is possible for sessions to completely monopolize the CPU. If the CQ processes can use 100% of the CPU, then the DQ batch jobs will never have a higher priority than the CQ activity and will be shut out from the CPU. This condition is referred to as process starvation.

The queues can be overlapped by the system manager to allow a DQ batch job the opportunity to share a portion of the CQ CPU. The greater the degree that the queues are overlapped, the more impact the batch activity can affect the CQ sessions. The queues are not overlapped by default for this reason. Interactive activity has the highest user priority on the computer because the CPU is servicing an user at the other end of a terminal. Batch activity is lower because there is no interaction with a end user. The input is received from the batch file. A system manager can modify the characteristics of a queue through the :TUNE command. Existing parameters brought over from the classic MPE environment include the BASE and LIMIT for the queue. This is the upper and lower bounds of the queue. Each time that a process launches on the CPU, the process priority will begin at the base of the queue. The priority will decay toward the limit of the queue as CPU is consumed. The MIN and MAX parameters of the :TUNE command set the minimum and maximum value for the quantum of CPU a process is allowed to consume before the decay takes place. C queue processes use this upper and lower bounds to form a range for calculation of the System Average Quantum (SAQ). All current C queue processes have their quantum consumption averaged into the SAQ allowing

the system to adjust according to activity. The system manager can modify the MIN and MAX range settings to favor either long or short transactions for the C queue. The default values are 200 ms and 2000 ms, respectively. By lowering the MAX, the system will favor short transactions. By raising the MIN, the system will favor long transactions. In many environments, the default settings will allow the system the most flexibility in adjusting to the current demand of the system. The system does not average D queue process quantums. The MIN and MAX parameters are set to the same value. The default is 2000 ms.

When a process is performing a long transaction, one that would require many quantums of CPU to complete, the process priority will eventually decay to the base of the queue where it will remain until completion. Remaining at a low priority could cause process starvation. Starvation occurs when a process is not permitted CPU for execution because the priority is lower than all other requesting processes. A new :TUNE parameter has been added as of operating system version 2.1. A system manager may set the C, D, or E queue to either DECAY or OSCILLATE. The default setting of DECAY will let the process decay to the limit and remain there until completion. Oscillate will boost the priority of the process back to the base once it has reached the limit. The process will then begin a decay to the limit again. The decay and oscillate boost will continue until the process completes.

As of 2.1 version of MPE XL, two new control commands have been introduced. The :SHOWPROC command will allow the system manager to determine at what priority a process is currently executing. It will also show the child processes within the process tree and how much CPU each have consumed. The :ALTPROC command allows a system manager to alter the priority of the process after the execution has started. This is one of the most significant control commands to have been added to the operating system. If a process is either consuming to much CPU or not receiving enough CPU, the process' priority can be altered to achieve the desired result. The new priority can be in one of three categories. It can be placed into a new queue, it can be fixed at a specific value with no decay in priority, or it can be fixed at the queue manager located at the base of the queue also with no decay. The :ALTPROC command can only be issued by a user with OP or SM capability.

There are three modes of program execution. They are compatibility mode (CM), native mode (NM), and translated compatibility mode. The MPE XL operating system has been designed to execute in native mode. Execution in compatibility mode will be much slower than native mode in

almost all programs. A system manager should convert all CM programs to NM through recompiling the source. This process requires some modification of the code in almost all cases. This may not be possible in all environments. In those cases, the compatibility mode programs should be translated through the :OCTCOMP command. This will attempt to optimize the execution of the CM program to take advantage of NM structures. The translation of a program does not require the source, the system manager uses the compiled object for the object code translation (OCT). A translated CM program will occupy 8 to 10 times the original disk space of the CM program. Performance of the program will be at the sacrifice of system disk space.

Excessive logon and logoff activity can also significantly impact the CPU. Some environments require high levels of security and implement inactivity timeout features. If a session has no activity at the terminal, within a period of time set by the system manager, the session is logged off. When the user wishes to resume work, the user will have to log onto the computer again. Logon and logoff is very intensive activity for the computer to perform. The system manager will need to weigh the benefit of security against the negative of overhead associated with extra logons and logoffs.

User defined commands (UDCs) will also delay a user logon. Each UDC that is cataloged for a user will have to be checked when the user logs onto the system. With the use of command files and the implied run feature, a system manager might be able to eliminate many of the UDCs on the system. A command file cannot utilize a UDC option of LOGON. This would be one reason for retaining some UDCs.

Batch activity on the system is another concern. How many batch jobs should the system manager allow to be executing on the system at the same time? When considering the :LIMIT for executing jobs, there are two considerations. First, will the job be considered a "sleeper" job. A sleeper job is one that wakes up periodically to execute and then returns to a "sleeping" stake not competing for CPU. If it is not competing for CPU while it is in this "sleeping" state, the system manager should not consider that job with the same weight as another when setting the job limit. Second, the higher the job limit, the more each of the nonsleeping jobs will have to share the available CPU. This will prolong the time it takes the processor to complete the job and allow a new job to take its place. For example, there are 20 jobs that need to execute on the system. They will each require 1 minute of CPU time to complete. If the job limit is set to 20, all of the jobs will execute at the same time. If the

processor provides 10 seconds of CPU to each job, it would take 200 seconds to provide each job with 1/6 of the required CPU to complete. At this point, no jobs have completed. Under the same conditions but with the job limit set to 2, after the elapse of 200 CPU seconds, 3 jobs will have completed and 2 others will have received 10 CPU seconds each. In many environments, the users batch jobs are short and only require small amounts of CPU. With a low limit, the system can utilize a first in first out method of processing.

PERFORMANCE TOOLS

Hewlett Packard has several performance products available. The two most common tools that will assist system managers in most environments are HP GlancePlus XL and HP LaserRX. They are very different in the information and application of the tool in the problem solving process.

HP GlancePlus XL would be considered a reactive and time of occurrence tool. System managers may react to performance fluctuations by entering the HP GlancePlus XL software and view the activity contributing to the current situation. It is considered a time of occurrence tool because it does not retain activity over long periods of time other than in cumulative totals.

HP LaserRX falls into the category of a reactive/proactive and after the time of occurrence performance tool. HP LaserRX utilizes a PC with a HP LaserRom drive to analyze the data offloading the processing overhead from the computer. System managers may proactively look at data over a period of time to anticipate activity levels. HP LaserRX also stores information about activity that the system manager predetermines as significant into log files. These log files can be analyzed after the time of the performance fluctuation. The information will only be as useful as the items the system manager has viewed as significant. If the fluctuation involves activity that is not within one of these preset categories, then it will be grouped together in a category of "other". The "other" category may not provide enough information to make any determinations. Too many categories will heavily impact the performance of the system and negate the benefit of using the tool.

Either of these tools will provide the system manager with information to evaluate the system load. Hewlett Packard has provided recommendations on the threshold values for each of the bottlenecks discussed above. That information has been published in the Performance Application notes that are included with the Software Status Bulletins (SSB). They are also archived on the HP LaserRom product.

Finally, when addressing the performance bottleneck with possible solutions, always insure that the most current patches for the release of the operating system have been installed.

CONCLUSION

There are many resources available to a system manager for the improvement of performance. As knowledge and comfort level increase, so will the use of MPE commands and performance tools. In summary, there are two critical points to remember in analyzing performance. First, know the environment before attempting any analysis or implementing any solutions. Second, when considering the alternatives to alleviate the performance fluctuation, if it costs a lot to get a little, then look for other alternatives or put up with the problem.

BIOGRAPHICAL SKETCH

Name: **D. Scott Pierson**

Presentation Title: **How to Win Memory and Influence CPU: A Look at MPE XL System Performance**

Paper Number: **3134**

Day/Time:

Scott Pierson is an Education Services Instructor with Hewlett Packard Company. He has been an instructor in the Fullerton area for the past two years focusing in the MPE XL migration and performance material. Scott is requested all over the United States for his classroom delivery of advanced courses.

Scott is frequently utilized by the Response Center and the sales force in providing consulting to the Hewlett Packard customer.

Prior experience includes a Response Center engineer position in the Mt. View Response Center for 1 and 1/2 years, and a system manager position with an aerospace company in the Fullerton area.

Scott graduated from California Polytechnic University of Pomona in 1986 with a BS in Computer Information Systems.

Paper Number : 3135

A Standard Operating System Interface for MPE XL

Rajesh Lalwani

Hewlett-Packard
19447, Pruneridge Avenue 47UH
Cupertino, CA 95014 USA

(408) 447-7456

Abstract

Hewlett-Packard is committed to providing standards across all of its product lines. POSIX is an IEEE standard for a Portable Operating System Interface to support source-level application portability. It forms a basic layer which is a first step towards more extensive sets of standards such as those currently being defined by X/Open. POSIX will provide access to the strengths of MPE XL through a standard interface, improving portability to the HP 3000 and increasing the number of application solutions available. POSIX is significantly different from MPE XL in a number of technical areas like Directory Structure and File System, File Access and Security, Process Management, Signals, and User Identification. The target audience for this paper is application developers and users familiar with the technical concepts of the MPE XL Operating System. The paper presents an overview of POSIX, discusses specifics of its implementation on the HP 3000-series 900, describes the concepts new to MPE XL users, and points out the major differences between POSIX and MPE XL.

---

1. INTRODUCTION

IEEE Std 1003.1-1988 is the first of a group of proposed standards generally referred to as POSIX. POSIX is an acronym for Portable Operating System Interface. The purpose of this standard is to define a standard operating system interface and environment to support source-level application portability. POSIX is intended for system implementors and application software developers. Other areas under active consideration of IEEE at this time are:

- Shell and Utility facilities - P1003.2
- Verification Testing - P1003.3
- Realtime facilities - P1003.4
- Ada Language bindings - P1003.5
- Secure/Trusted System considerations - P1003.6
- System Administration - P1003.7

- Network interface facilities - P1003.8
- FORTRAN Language bindings - P1003.9
- Language-independent service descriptions
- An overall guide to POSIX-based or related Open Systems standards - P1003.0

This paper concentrates mainly on the concepts described in P1003.1. IEEE Std 1003.1-1988 has been published as a paperback book and is available at several technical bookstores.

Conventions Used

Following conventions have been used throughout the paper:

a. POSIX means IEEE Std 1003.1-1988 unless otherwise mentioned.

b. HP 3000 means HP 3000-series 900.

c. MPE XL intrinsic names are written as `HPFOPEN`. POSIX and C language functions are written as *mkdir()*. Appendix A gives a one-line description of all the POSIX and C language functions mentioned in the paper.

d. Various system wide constants defined by POSIX and declared in the C include file <limits.h> are written as {_POSIX_PATH_MAX}. These constants can be used inside programs that use C language interface.

2. AN OVERVIEW OF THE POSIX CONCEPTS

This section will present an overview of the fundamental POSIX concepts. Note that it does not describe specific POSIX interfaces in detail but, instead, focuses on the basic environment to which these interfaces provide access. This section does not describe the MPE XL implementation of these concepts. Section 4 gives the highlights of the POSIX implementation on the HP 3000.

2.1 User Identification

The MPE XL operating system has been designed primarily for use in a departmental setting. All the users in a particular department are typically grouped together and placed in one account. So, an MPE XL system has one or more accounts and each user belongs to an account. All the users are identified by a `user.account` string, for example, `MGR.PAYROLL`. The `user.account` associated with a job or session does not change for the life of the job or session.

On the other hand, POSIX supports individual users. Thus, POSIX does not presume a `user.account` structure. Each user has an associated string of characters as an identification; generally referred to as a **login name**. An example of a login name is `rlalwani`.

Each system maintains a **user database** (similar to the /etc/passwd file on UNIX systems)

which contains at least the following information for each user:

1. login name
2. numerical user ID (UID)
3. numerical group ID (GID)
4. initial working directory
5. initial user program (shell)

Each system user is identified by a non-negative number known as a **user ID** or UID. For example, the user `rlalwani` could have a UID=12. When the identity of a user is associated with a process, a user ID is referred to as a **real user ID**, an **effective user ID**, or an (optional) **saved set-user-ID**.

To facilitate sharing of files among users on the system, POSIX supports grouping of users. This is similar to the file sharing among users in the same account on MPE XL. Each POSIX user is a member of at least one group. A group is identified by a **group ID** or GID which is a non-negative number. For example, the user `rlalwani` could be a member of a group with GID=14. There may be other members in the same group and they will all have the same GID (14). When the identity of a group is associated with a process, a group ID is referred to as a **real group ID**, an **effective group ID**, or an (optional) **saved set-group-ID**. Each system also maintains a **group database** which contains at least the following information for each group:

1. group name
2. numerical group ID
3. list of names or numbers of all users in the group

2.2 Process Management

MPE XL users are familiar with the `CREATE` and `CREATEPROCESS` intrinsics. These two intrinsics are used to create a new process. The process who calls the intrinsic is called the *parent* process and the newly created process the *child* process. Actually, these instrinsics do two things:

1. create a new process, and
2. decide what code the new process is going to execute as specified by the `formaldesig` parameter.

POSIX requires two separate functions *fork()* and *exec* respectively to accomplish the two tasks mentioned above. There are some benefits of this approach. For example, the child process could use the variables set by its parent between tasks 1 and 2 above. The variables are used by the parent process to communicate information to the child process. Also, after *fork()* both the processes are active unlike MPE XL where the parent process must call `ACTIVATE` to start the child process. A process creation example is described in section 3.

The function *fork()* creates a new process which is an almost identical copy or clone of the

process that called the *fork()* function. Both processes execute the same program code. One difference between the two processes is that *fork()* returns the process ID of the child to the parent, and returns zero to the child process. When parent and child processes start executing, they execute the statement just after the call to *fork()*. Typically, the child process either branches to a different path in the same code to take advantage of the variables, file descriptors set by the parent process, or calls one of the *exec* family of functions to replace its image with a new executable file. The child process may choose to terminate by using the *exit()* function, and its parent process may wait for that event by using the *wait()* function.

The *exec* family of functions - *execl(), execv(), execle(), execve(), execlp(), execvp()* replace the current process image with a new process image. The new image is constructed from a regular, executable file called the new process image file. There is no return from a successful *exec*, because the calling process image is overlaid by the new process image. No new process is created as a result of *exec* functions.

The program that calls one of the *exec* functions can pass arguments to the new process image via the *argc* and *argv* arguments of the *main()* function.

Now, let us look at some POSIX definitions.

Each process in the system is uniquely identified during its lifetime by a positive number, called a **process ID** or PID. This is similar to the concept of Process Identification Number (PIN) on MPE XL.

Each process in the system is a member of a **process group** that is identified by a **process group ID**. This grouping permits the signaling (to be explained later) of related processes. A newly created process becomes a member of the process group of its creator. A process whose process ID is the same as its process group ID is referred to as the **process group leader.**

Let us look at the process specific attributes such as real UID, real GID, effective UID, effective GID, and supplementary GIDs that will help explain the concepts of File Access and Security in the later part of this paper.

The **real user ID** is an attribute of a process that, at the time of process creation, identifies the user who created the process. Similarly, **real group ID** is the attribute of a process that, at the time of process creation, identifies the *group* of the user who created the process.

The **effective user ID** is an attribute of a process that is used in determining various permissions, including file access permissions. Similarly, the **effective group ID** is an attribute of a process that is used in determining various permissions, including file access permissions.

Typically, the effective UID and effective GID are used by processes to assume the identity of another user. Let us look at an example. Take the case of the user with the login name `rlalwani` as seen earlier. Initially, the shell runs with real UID = effective UID = 12 and real GID = effective GID = 14. Assume that there is an executable file called `passwd` which `rlalwani` decides to execute. In POSIX, there are two bits associated with every executable file that tell the system to use the UID and GID of the *owner* of the file as the effective UID

and effective GID of the process running that file. (This is in contrast to the normal method of using the UID and GID of the *user* executing the file.) So when `rlalwani` decides to run `passwd`, the effective UID and effective GID of that process will change to those of the owner of `passwd`. Now, if the owner of the file `passwd` had read/write access to the user database file, say `/etc/passwd`, the program would succeed in modifying the file `/etc/passwd` even if `rlalwani` did not have write access to it. This is because the effective UID/GID, not the real UID/GID, is used to determine file access permissions.

The real UID, real GID, effective UID, and effective GID are all subject to change during the process lifetime. The reader should see *setuid()* and *setgid()* for details.

A process also has up to {NGROUPS_MAX} **supplementary group IDs** used to determine file access permissions, in addition to the effective group ID. The supplementary group IDs of a process are set to the supplementary group IDs of the parent process when the process is created. The minimum value of {NGROUPS_MAX} on any implementation of POSIX is 0 which means that a particular implementation may or may not support the concept of supplementary GIDs.

2.3 Signals

For users familiar with MPE XL, signals provide functionality roughly similar to that provided by a combination of intrinsics like `XCONTRAP`, `RESETCONTROL`, `XARITRAP`, `ARITRAP`, `XSYSTRAP`, `XLIBTRAP`, `SUSPEND`, `ACTIVATE`, `KILL` etc.

As an example, `XCONTRAP` is used to arm and disarm a user-written subsystem break trap handling procedure. For most applications, this signal is generated by pressing control-y. If it is armed, the trap handler is invoked when an enabled subsystem break signal is received during an interactive session. Once a process has received a subsystem break trap, it cannot receive another until it calls the `RESETCONTROL` intrinsic to re-enable the subsystem break signal.

The functionality provided by POSIX signals is similar to what was just described. POSIX provides a number of different types of signals, and a process can take one of several actions (default, ignore, or user-written) when a signal is received. So, let us have a look at the POSIX signals.

A **signal** is a mechanism by which a process may be notified of, or affected by, an event occurring in the system. Examples of such events include hardware exceptions and specific actions by processes.

A signal is said to be **generated** for a process when the event that causes the signal first occurs. Examples of such events include detection of an illegal hardware instruction (symbolic constant, SIGILL), erroneous arithmetic operation, such as division by zero (SIGFPE), terminal activity, as well as the invocation of the *kill()* function. See Tables 1 and 2 for other types of signals.

For most signals, a process can take one of the following three actions:

1. Take the signal specific default action (SIG_DFL).
2. Ignore the signal (SIG_IGN). However, a process cannot take SIG_IGN action for SIGKILL or SIGSTOP signals.
3. Execute a user-written signal-catching function. The system does not allow a process to *catch* SIGKILL or SIGSTOP signals.

A signal is said to be **delivered** to a process when the appropriate action, defined by the process for the particular signal, is taken.

Each process has a **signal mask** that defines the set of signals which cannot be delivered to the process. A new process inherits the signal mask from its parent. Each process can manipulate the signal mask using functions like *sigaction(), sigprocmask(),* and *sigsuspend().*

The following tables give the symbolic constant and a brief description for each of the **required** signals and **job control** signals. All POSIX implementations support the required signals. POSIX implementations that provide job control (such as background/foreground jobs) support the job control signals in addition to the required signals.

Table 1 : Required Signals

Symbolic Constant	Description
SIGABRT	Abnormal termination signal, such as is initiated by the *abort()* function.
SIGALRM	Timeout signal, such as is initiated by the *alarm()* function.
SIGFPE	Erroneous arithmetic operation, such as division by zero or an operation resulting in overflow.
SIGHUP	Hangup detected on controlling terminal or death of controlling process.
SIGILL	Detection of an invalid hardware instruction.
SIGINT	Interactive attention signal.
SIGKILL	Termination signal (cannot be caught or ignored).
SIGPIPE	Write on a pipe with no readers.
SIGQUIT	Interactive termination signal.
SIGSEGV	Detection of an invalid memory reference.
SIGTERM	Termination signal.
SIGUSR1	Reserved as application-defined signal 1.
SIGUSR2	Reserved as application-defined signal 2.

Table 2 : Job Control Signals

Symbolic Constant	Description
SIGCHLD	Child process terminated or stopped.
SIGCONT	Continue if stopped.
SIGSTOP	Stop signal (cannot be caught or ignored).
SIGSTP	Interactive stop signal.
SIGTTIN	Read from control terminal attempted by a member of a background process group.
SIGTTOU	Write to control terminal attempted by a member of a background process group.

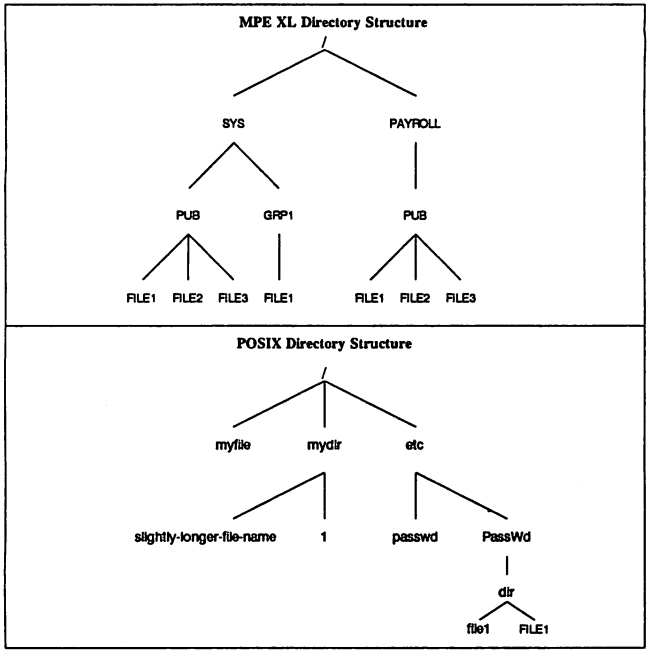
2.4 Directory Structure and File System

MPE XL users are familiar with the `file.group.account` directory structure. In this model, the directory tree consists of one or more accounts. Each account contains one or more groups. Finally, each group has zero or more files in it. Hence, the MPE XL's `file.group.account` directory structure can be looked at as a 3-level hierarchical structure. The first level has account entries, the second level group entries, and the third level the actual files. POSIX's view of the directory structure is an expanded version of MPE hierarchy. The POSIX directory structure has a root directory which is represented by a *slash* (/). The root directory may have files and directories under it. The directories can, in turn, have files and directories under them. The figure on the next page shows the MPE XL and POSIX directory structure.

Now, let us look at some POSIX definitions.

A **file** is an object that can be written to, or read from, or both. A **file** has certain attributes, including access permissions and type. A **directory** is a **file** that contains **directory entries**. No two **directory entries** in the same **directory** can have the same name. The interesting thing to note is that unlike MPE XL, POSIX views a **directory** as a **file** (of **directory** type). Finally, a **directory entry** is an object that associates a **filename** with a **file**. Several **directory entries** or **links** can associate names with the same **file**. Thus, POSIX distinguishes between the file and the filename. This is different from MPE XL, where each file can be referenced only by one name.

The POSIX file naming rules are also different from MPE. On MPE XL, each account, group, and file can have a name of, at most, 8 characters. The valid characters are upper case letters and digits with the restriction that the first character cannot be a digit. Hence, `MPEXL`, `PUB`, `A8000000`, and `NEWUSER` are all valid MPE XL names (for files, groups, or accounts) but `81FNAME` and `LONGFNAME` are illegal names.



A POSIX **filename** consists of 1 to {NAME_MAX} bytes. {NAME_MAX} is implementation dependent, but it cannot be less than 14. For a filename to be portable across conforming implementations of IEEE Std 1003.1-1988, it can consist only of the following characters:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

0 1 2 3 4 5 6 7 8 9 . _ -
```

The hyphen cannot be used as the first character. Unlike MPE XL, POSIX filenames are case sensitive. The filenames consisting of a single dot character (.) and consisting solely of two dot characters (..) have special meaning and are referred to as **dot** and **dot-dot** repectively. Their meaning will be explained below.

Examples of valid POSIX filenames are `myfile`, `MyFile`, `file-20`, `paper.text`, `_system_file`, and `12181990`. The filename `myfile` is different from `MyFile`. One

example of an invalid filename is `-system-file` because it begins with a hyphen.

A **pathname** is a string that is used to identify a **file**. It consists of, at most, {PATH_MAX} bytes, including the terminating null character. It has an optional beginning *slash* (the character /), followed by zero or more **filenames** (also known as pathname components) separated by *slashes*. If the **pathname** refers to a **directory**, it may also have one or more trailing *slashes*. {PATH_MAX} is defined by a particular implementation of POSIX but it cannot be less than 255.

Current Working Directory

On MPE XL, every job and session has an associated *logon group* which is used to qualify an unqualified filename. For example, if a user logs on as follows

```
HELLO RAJESH.MPEXL,PUB
```

the logon group is `PUB.MPEXL` and an unqualified file reference such as `LETTER` would actually refer to the file `LETTER.PUB.MPEXL`. POSIX uses two such references - **current working directory** and **root directory**. It is important to note that these references are defined for each *process*. Thus, two processes in a single session can have two different current working directories. The root directory of a process is usually the same as the root of the system directory.

If the pathname begins with a *slash*, the predecessor of the first filename in the pathname is the root directory of the process, and such pathnames are referred to as **absolute pathnames**. For example, `/usr/mail/rlalwani` is an absolute pathname. An absolute pathname is similar to a fully qualified filename on MPE XL. If the pathname does not begin with a *slash*, the predecessor of the first filename of the pathname is the current working directory (CWD) of the process and such pathnames are referred to as **relative pathnames**. `MyDir/MyFile` is an example of a relative pathname. A relative pathname is similar to a fully *un*qualified filename on MPE XL. The special filename **dot** refers to the directory specified by its predecessor in the pathname, and **dot-dot** refers to the parent directory of its predecessor directory in the pathname.

Other examples of an absolute pathname are `/tmp/rlalwani/file1`, `/tmp/rlalwani/./file1`, `/tmp/rlalwani/../rlalwani/file1`, and `/SYS/PUB/CATALOG`. By the way, the first three pathnames refer to the same file.

An example of a relative pathname is `file1`. If the Current Working Directory is `/tmp/rlalwani`, refering to `file1` will actually mean `/tmp/rlalwani/file1`.

2.5 File Access and Security

The standard file access control mechanism of POSIX uses the file permission bits. It is quite different from the models used by MPE XL: file access matrix based, capability based, file lockword based, and access control definition or ACD based. Let us look at the model used by POSIX.

A process can access a file in the following ways : **read, write,** and **execute/search.** The meaning of read and write access is analogous to the meaning on MPE XL. Search access applies to a directory whereas execute access applies to an executable file like a program or a script file. There is a subtle difference between read (as applied to a directory) and search access. If a process wants to open a directory and read the entries in it (possibly using *opendir(), readdir()*), the process needs read permission for that directory. But if a process wants to access a file, say, `/tmp/rlalwani/file1`, the process needs search permission for all the directories in the path, namely, `/`, `tmp`, and `rlalwani` in this particular case.

Every file in the file system has **file permission bits** associated with it. Since a directory is a *file* of directory type, the same is true for all directories in the file system. File permission bits contain the information about a file that is used, along with other information, to determine whether a process has read, write, or execute/search permission to a file. The bits are divided into three classes: **owner, group,** and **other.** In addition to the file permission bits, there is a user ID (owner UID) and a group ID (owner GID) associated with every file in the file system. The owner UID and owner GID are initialized to the UID and GID of the creator (user) of the file at the time of file creation.

For the purposes of access control, processes are classified as belonging to one of three access classes: file owner class, file group class and file other class. A process is in the **file owner class** of a file if the effective UID of the process matches the user ID of the file. A process is in the **file group class** of a file if the process is not in the file owner class and if the effective GID or one of the supplementary GIDs of the process matches the group ID associated with the file. A specific implementation may define additional members of the file group class. Lastly, a process is in the **file other class** of a file if the process is not in the file owner class or file group class. Let us look at an example.

Suppose, the user `rlalwani` creates a program file called `prog` in his current working directory `/users/rlalwani`. When the file is created, the owner UID and owner GID will be assigned values of 12 and 14 respectively (the UID and GID of `rlalwani`). Now, let us assume that the permission bits for the file `prog` are as follows:

owner : read (r), write (w), execute(x)
group : none
other : execute

Consider two users `tshem` (UID=25, GID=14) and `boconnor` (UID=27, GID=15) on a system that doesn't support any supplementary GIDs. Both try to execute the file `/users/rlalwani/prog`. Let us assume that the directories `/`, `user` and `rlalwani` give the search access to everyone (i.e., owner, group, and other). It can be seen that the user `tshem` will be in the file group class, hence `tshem` cannot execute the file. But the user `boconnor` is in the file other class and hence has execute access to the file. Thus the user `boconnor` will succeed in executing the program file.

The standard file access control mechanism of POSIX uses the file permission bits described above. These bits are set at file creation time and can be changed by the *chmod()* function. These bits can be read by calling *stat()* or *fstat()* functions.

Implementations of POSIX may also provide **additional** or **alternate** file access control mechanisms or both. An additional access control mechanism can only further restrict the access permissions defined by the file permission bits. An alternate access control mechanism, if enabled, is used instead of the standard mechanism. The alternate access control mechanism has some constraints, the chief being that it must be enabled only by explicit user action, on a per-file basis. Lastly, POSIX also allows privilege based security in which access may be granted to a process if it has appropriate privilege. Each POSIX implementation can define what constitutes an appropriate privilege.

Although POSIX 1003.6 (Security) has not been finalized yet, it is worth mentioning that it proposes the access control lists (ACLs) as a file access control mechanism. The concept of ACLs is the same as ACDs (Access Control Definitions) currently available on MPE XL.

3. WRITING AN APPLICATION USING POSIX INTERFACE

The previous section presented a number of POSIX concepts. To see how some of them fit together and how they are typically used in the POSIX/UNIX world, let us design an application.

Problem Statement

Write a shell (command interpreter) which will be put in the user database as the initial user program for the user `rlalwani`. The shell will read a *program name* typed at the terminal and create a process to run that program. The program file resides in the initial working directory and there are no command line parameters. When the program terminates, the shell would again prompt for another program name. The shell will terminate when the user types "bye".

Also, the shell should be able to handle control-c interrupt at the terminal which would be sent by the terminal handler as SIGINT. If control-c is pressed at the shell prompt, it should be ignored. But if control-c is pressed when a program is being executed, the program should be terminated (unless the program takes a specific action for SIGINT) and the shell should prompt for a new program name.

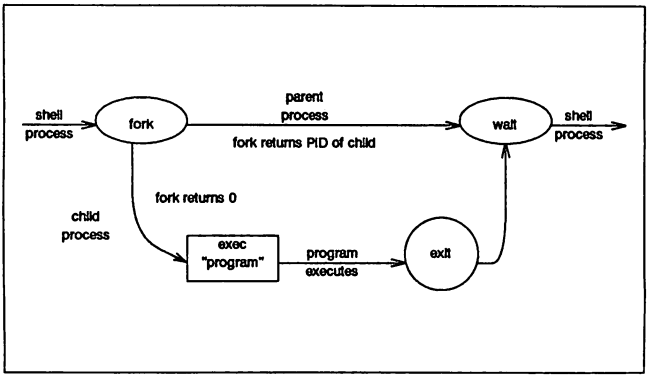
Writing the Shell

The shell program `myshell.c` will make use of the *fork()* and *execlp()* functions to create a new child process and to run a specific program file.

```
/* Simple Shell - myshell.c */

#include <limits.h>
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
```

```
    #define TRUE 1
    main()
    {
        char    program_name[ _POSIX_NAME_MAX]; /* Name of the program that */
                                                /* the user wants to run  */
        int     *stat_loc;    /* Pointer to the status of the terminated */
                              /* child */
        pid_t   child_pid;    /* The process ID of the child process  */

 1:     sigaction(SIGINT, SIG_IGN, NULL); /* Ignore the SIGINT signal */

 2:     while (TRUE)
        {
 3:             printf("\n$");                    /* shell prompt   */
 4:             scanf("%s", program_name);
 5:             if (!strncmp(program_name, "bye", strlen("bye"))
 6:                  break;                       /* quit the loop  */

 7:             child_pid = fork();

 8:             if (child_pid)                    /* parent process */
 9:                 wait(stat_loc);
10:             else                              /* child process  */
                    {
11:                    sigaction(SIGINT, SIG_DFL, NULL);
12:                    execlp(program_name,(char *) 0);
                    }
13:    } /* while loop */

    } /* end main */
```

The following figure shows what happens when `myshell` is executed.



Let us see how the whole thing works. Assume that the system administrator has modified the user database corresponding to `rlalwani` so that when `rlalwani` logs on, the initial program (shell) that will be invoked by the system is `myshell`. There will be a process that will be executing the above code.

When the above program starts executing at line 1, it uses the *sigaction()* function to ignore the control-c signal (SIGINT). So from now on, whenever a control-c is pressed on the terminal, it will be ignored by this shell process. Between lines 2 and 13 is an infinite loop unless an explicit action is taken within the body of the loop to quit. In our program, the `break` statement at line 6 accomplishes this purpose. The program prints a new line character and the "$" prompt (line 3). It reads the name of the program typed by the user (line 4). If it is not "bye", statement at line 7 creates a new process by using *fork()*. From now on there are two processes running exactly the same code. The only way to know whether the process is the parent process or the child process is to see the value returned by the *fork()* function call. If it is some non-zero value, it is the parent process and hence the program decides to wait for the death of the child process (line 9). If it is the child process, the program first uses the function *sigaction()* to associate a default action to be taken when a SIGINT signal is received (line 11). In case of SIGINT, the default action is to terminate the process. Now, the program uses *execlp()* to replace the current image (our shell program) with the program that user specified (line 12). There will be no return from this call to *execlp()*. There are two interesting scenarios: one in which the user program terminates normally (using an *_exit(), exit()* function or return from the *main()* function); and, two, in which the user types control-c. Let us look at both of them.

If the user program terminates normally, the call to *wait()* will complete and the parent process (shell) will start executing. It will go to the beginning of the loop and prompt for next program name.

If the user presses control-c, this signal is sent to both the parent process and the child process. The parent process has chosen to ignore it. But it will be awakened and will no longer be stuck at line 9. It will start executing the loop again and prompt the user for another program name. Let us see what happens to the child process. If the user program didn't use any *sigaction()*, the default action will be taken for it and in this case, it will be terminated. If the user program used *sigaction()* function to associate some other action with SIGINT, that action will be taken when a control-c is typed at the terminal.

The shell program repeats the loop until the user types "bye" at the prompt. At this point, the statement at line 6 is executed and the shell program leaves the loop. Since there are no more statements after the loop, the shell process terminates.

4. HIGHLIGHTS OF POSIX IMPLEMENTATION ON HP 3000

The POSIX implementation on HP 3000 has been architected with two main goals:

- *Backward Compatibility*

  If the system manager elects not to enable POSIX on an HP 3000, or if a user elects not to use the new POSIX features (on a system which has POSIX enabled) at all, the behavior of an MPE XL application will remain the same.

- *Integration*

  MPE XL users will be able to access the new features as transparently as possible. MPE XL users will have access to most of the new features, such as hierarchical directory, through familiar commands and intrinsics. MPE XL interfaces will be able to access files created by a POSIX interface and vice versa.

Let us look at a few specific areas.

• User Identification

Users will continue to log on as they do today on MPE XL using the `:HELLO` command. Each user must belong to an MPE account. When POSIX is enabled, all the users in an account form a *group* and have the same GID.

For example, if POSIX has been enabled on an HP 3000 system and a user `MGR` in account `MPEXL` wants to log on in a group called `PUB`, s/he will issue the following command:

```
HELLO MGR.MPEXL,PUB
```

• File Name Resolution

This sub-section explains file name resolution using the MPE interface. An example of an MPE interface is the intrinsic `HPFOPEN`. File name resolution using a POSIX interface such as *open()* was explained in the sub-section **Directory Structure and File System** of section 2 above. An example, however, is given at the end of this sub-section.

If a file name is presented to an MPE XL interface, either the MPE name server or the POSIX name server will be invoked as follows:

POSIX Name Server

The POSIX name server will be invoked if the filename begins with a *slash* or a *dot*.

MPE Name Server

The MPE name server will be invoked if the filename does *not* begin with a *slash* or a *dot*.

In the case of the MPE name server, the filename will be resolved as follows:

- If the filename is fully qualified, such as A.B.C, it will continue to refer to the file A in the group B in the account C. This is regardless of the CWD and the logon account. See the examples below.
- If the filename is of the form A.B, it will refer to the file A in the group B of the logon account. This is regardless of where the CWD is. Note that the filename A.B refers to the same file as it does today.
- If the filename is of the form A, it will refer to the file A in the CWD. Note that if the CWD were the same as the logon group, filename A will refer to the same file as it does today.

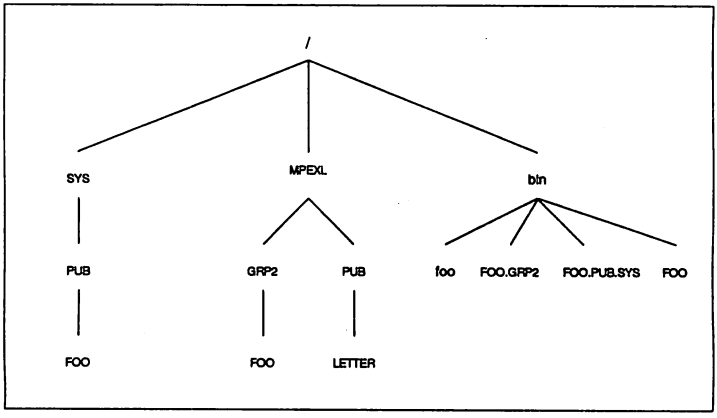
In all the three cases above, MPE name server converts the file, group and account name to upper case letters.

The above set of rules for filename resolution is an excellent example of *backward compatibility* and *integration* mentioned earlier. Let us look at some examples now.

Assume that an HP 3000 system has POSIX enabled and a user logs on as follows:

```
HELLO MGR.MPEXL,PUB
```

where `MPEXL` is an MPE account and `PUB` an MPE group in that account. Now, if the user wants to refer to a file called `LETTER` in the logon group using an MPE interface, s/he could use any of the following names: `LETTER`, `LETTER.PUB`, `LETTER.PUB.MPEXL`. If the user wants to use the new hierarchical directory structure and wants to refer to a file `foo` in directory `bin` under the root, s/he could refer to it by `/bin/foo`.



Now, the user changes his/her current working directory (CWD) to `/bin`. If s/he wants to refer to the file `foo` (lower case letters) in the CWD using an MPE XL interface, s/he cannot use the filename `foo` because the absence of a *slash* or a *dot* in the beginning would invoke the MPE name server. Since the MPE name server upshifts the name, it will refer to the file `FOO` (upper case letters) in the CWD which is different from the file `foo` in the CWD.

Also, assume there are files named `FOO.GRP2` and `FOO.PUB.SYS` in the CWD (`/bin`). Note that these are valid filenames in POSIX syntax because *dot* is a legal character. If the user supplies the names `FOO.GRP2` (or `foo.grp2`), and `FOO.PUB.SYS` (or `foo.pub.sys`) to an MPE interface, they will be considered MPE filenames and will refer to (file `FOO` in group `GRP2` in account `MPEXL`) and (file `FOO` in group `PUB` in account `SYS`) respectively.

So, how can the user access these two files using an MPE interface? According to the above rule, if a filename begins with a *slash* or a *dot*, it is considered to be a POSIX filename. Thus, using `./FOO.GRP2` and `./FOO.PUB.SYS` will do the job.

On the other hand, if the user is accessing files via a POSIX interface such as *open()*, the above two files can be referenced as `FOO.GRP2` and `FOO.PUB.SYS`. To access the file `FOO` in group `PUB` in account `SYS`, the user can specify `/SYS/PUB/FOO`.

5. CONCLUDING REMARKS

POSIX is an IEEE operating system interface standard endorsed by both X/Open and the Open Software Foundation. Since POSIX defines a standard set of operating system interfaces, it will be easy for third party developers to port an application program from one POSIX-compliant vendor platform to another. POSIX also makes it easier to port an application from a UNIX platform to a POSIX-compliant platform. This ease of portability results in an increased number of application solutions available to the customer.

POSIX only defines an interface and leaves the internal implementation to the vendor implementing POSIX. Thus, POSIX applications running on HP 3000 can take advantage of its high availability, reliability and high performance OLTP (on-line transaction processing).

6. TRADEMARKS

Ada is a registered trademark of the U.S. Government - Ada Joint Program Office.
UNIX is a registered trademark of AT&T.
X/Open is a trademark of X/Open Company Limited.

7. ACKNOWLEDGMENTS

I sincerely thank my colleagues who reviewed the paper and provided invaluable feedback. I would like to especially thank Janet Garcia, Brian O'Connor and Jeff Vance who helped me substantially write the paper in its present form.

8. BIOGRAPHY

Rajesh Lalwani received M.Tech. in Computer Science from the Indian Institute of Technology, New Delhi in 1986. In 1988, he received M.S. in Computer Science from the Pennsylvania State University. In 1990, he obtained Computer Science Certificate in Databases from Stanford University.

Rajesh Lalwani joined Hewlett-Packard Company in 1988 as a Software Development Engineer in the MPE Lab. Currently, he is a member of a project providing Open Systems Interfaces on the HP 3000 systems.

APPENDIX A

The following table gives a one line description for each of the P1003.1 and C language functions mentioned in the paper.

Function Name	One-line Description
_exit abort	terminate the calling process abort a process (not a P1003.1 function)
alarm	cause system to send SIGALRM signal after specified time
chmod	change S_ISUID, S_ISGID, file permission bits for a file
exec	a family of functions to replace the current process image with a new process image
exit	terminate the calling process (not a P1003.1 function)
fork	create a new process
fstat	obtain information about a file
kill	send a signal to a process or a group of processes
mkdir	create a new directory
open	open a file
opendir	opens a directory stream corresponding to a directory name
printf	print formatted output
readdir	returns a pointer to the directory entry at the current position
scanf	formatted input
setgid	set real GID, effective GID, and/or saved set-GID
setuid	set real UID, effective UID, and/or saved set-UID
sigaction	examine and/or specify an action associated with a signal
sigprocmask	examine and/or change a process's signal mask
sigsuspend	replace the process's signal mask and suspend the process
stat	obtain information about an open file
strncmp	compare character strings
wait	obtain information about one of the terminated child processes

Paper #: 3136

Transaction Analysis

for Capacity Planning

Ralph T. Kotoski
Hewlett-Packard Company
24 Inverness Place East
Englewood, Colorado 80112
(303) 649-5761

Introduction

Today, more than ever, capacity planning is critical to business success. With competitiveness on the rise, companies have to carefully plan for all types of situations. One of the difficult areas to plan for is in the Information Systems area. Computers, as with any business asset, have finite capacities. The focus of this paper is to address one area of capacity analysis. This paper will illustrate simple, low overhead techniques for determining system resource utilization and response time for specific critical transactions.

It is often difficult to accurately determine specific transaction response time and resource utilization with common tools available today. Most tools provide an response time report averaged over all types of transactions. But what if we want to know about a single, tightly defined transaction? The job has just become more difficult. With the advent of SPT/XL and several new system intrinsics, it is possible to obtain very detailed information regarding any definable transaction, including number and type of intrinsic calls, response time, where a transaction spent most of its time, etc. However, the overhead of capturing and logging this data precludes most from being able to utilize this tool on a continuous basis. It was designed with application optimization in mind, not as capacity planning or trending tool. So, what can be done?

The major point of this paper is to illustrate a method for obtaining certain types of performance data in order to make capacity analysis easier. In this paper, transactions will be defined from several perspectives. It will illustrate how current tools available relate to these perspectives. Finally, a simple, low overhead technique for capturing pertinent transaction information will be presented.

What is a Transaction?

Defining the attributes of a transaction can be difficult and depends upon how the transaction is viewed. From a user perspective, a transaction may include several lines of data entry, one or two lookups, and some more data entry. From the application developer perspective, that user transaction may translate into a transaction for the initial data entry, two transactions for the lookups, and another transaction for the final data entry. Without some sort of logging, in a TurboIMAGE database for example, each DB intrinsic call to the database is treated as a transaction. From the system perspective, the HP 3000 by default, treats each terminal read completion as a transaction. This is known as a terminal transaction. In character mode applications, a terminal transaction occurs each time a user hits Carriage Return. In block mode applications, several terminal transactions occur each time the ENTER key is hit.

Let's work our way back through the list to see the importance of each perspective and its impact on our view of response time. The Measurement Interface in the MPE (both V/E and XL) operating system has no way of knowing what constitutes a user or logical transaction. By default, then, it assumes each terminal read completion to be a transaction. All metrics provided by tools strictly using only the Measurement Interface are based upon terminal read completions. Every time a user hits the Carriage Return key in a character mode application, a terminal read is completed and the Measurement Interface logs the pertinent data for that transaction. In block mode applications, several terminal transactions occur between the system and the sending terminal when the ENTER key is hit by the user. Generally, the handshaking terminal reads are filtered.

The data reported by performance tools can be somewhat misleading, as it is based upon individual terminal transactions. Response times from these tools can appear lower than that perceived by users. This is why it is imperative to understand the difference between what a user perceives as a transaction and the system thinks is a transaction. The graph below, Figure 1, shows a typical representation of terminal transaction data collected from the Measurement Interface in MPE.

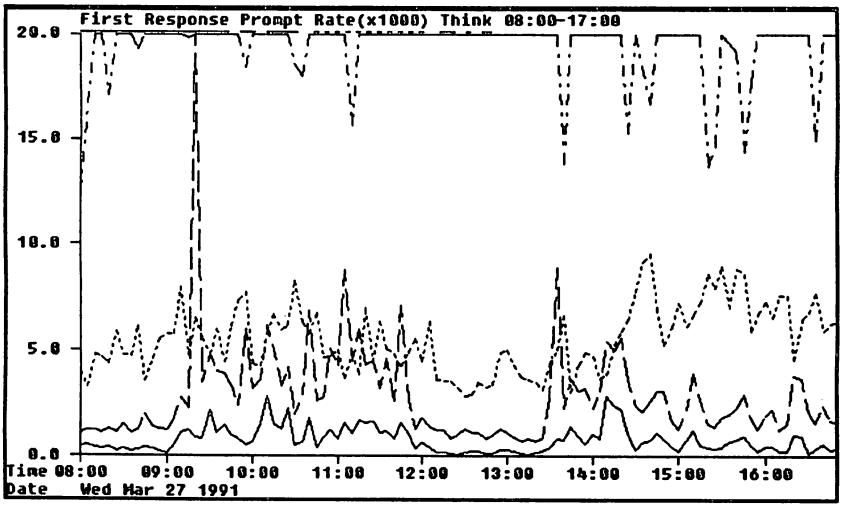


Figure 1 - Transaction Response Graph from LaserRx

In this representation, Think, First Response, Prompt, and Rate are based upon terminal read completions. If the applications on this system were character mode

and each terminal transaction constituted a transaction in the minds of the users, then a direct correspondence could be made between user perceptions and performance tool reporting.

From an application development or processing standpoint, a transaction can be something much different than the completion of a terminal read. A transaction can incorporate several database accesses, output to a report file, and even sorting of data. Generally, an application developer will bracket a transaction with the DBBEGIN and DBEND intrinsics and use TurboIMAGE logging. This way, if some problem should occur, recovery is relatively simple, and the database will not have any incomplete transactions.

A complex transaction, in this sense, may take a relatively long period of time to complete. It may even require additional input from the user. This kind of transaction may appear to the Measurement Interface of MPE as several terminal transactions. Consequently, MPE will record the transaction count, and, the response time for each terminal transaction.

As mentioned earlier, the program developer can use two new intrinsics, STARTTRAN and ENDTRAN, in conjunction with SPT/XL to define and optimize transactions.

Finally, and most importantly, we need to take a look at the definition of a transaction from the user perspective. Most users do not view the work they are doing on the system in terms of transactions. However, they do know when the system is slow, that is, when response time is long.

How is this concept translated into something that can be related to the available metrics from the system? First, and foremost, we need to understand what kinds of transactions users think are receiving poor response time. Many times, users will not even consider relatively short data input transactions as transactions. Users will believe a look-up or a posting of an order as their only transaction (if they even think in terms of transactions). Therefore, it is imperative we concentrate on what users consider a transaction. Once we have done this, we can focus on capturing the relevant data in order to better understand the how the system is responding to the environment.

Having put forth the effort to understand and define transactions from the user perspective, the type of information needed about each transaction should be defined. In order to maintain a low overhead approach, it is important to limit the quantities and types of data collected. It is important to capture and log only the most important data. Capturing and logging too much data could cause

performance problems. The following list illustrates some of the types of performance data generally desired for transaction analysis purposes:

Response Time,

CPU Time,

Nested Response Time, and

Nested CPU Time.

Nested Response and CPU Times may be required when certain smaller transactions are nested within a larger major transaction. It is also desirable to timestamp the transaction data in order to be able to focus on a particular window of time.

It may also be useful to log pertinent user information, such as logon, terminal ldev and a user identification number of some sort. Finally, though it may be more overhead than it is worth, information such as frequency and type of intrinsic calls could be logged. This would be more helpful in optimization rather than capacity analysis, and would be better suited in an environment using SPT/XL.

After defining the type of data to be collected, the next step is to integrate the data capture code into the application. The first step is to define the logfile where the data is to be kept. For this purpose, a Long Mapped file (Long Mapped files can be shared among processes and Short Mapped files cannot) should be used for optimum performance. As an example, let's define our logfile record as follows:

Transaction Name or Type	Transaction Number	Pin Number	Logon Information	Begin/End Flag	Date and Time	CPU Time

Figure 2 - Possible Log File Record Format

It is important to point out that we do not want the logging process to perform any more work, such as calculations, than is necessary. For example, causing the process to calculate transaction response and CPU time would place unnecessary overhead inside the application. It is better to save this type of activity for the reporting process, discussed later. It is critical that the logging activity use minimal system resources in performing its duty.

The best way to access a Mapped file is through Pascal/XL or C/XL. The reason for this is that all access to Mapped files is via a virtual address pointer. Pascal and C are both ideally suited for this type of access. The examples in this paper will be in loose Pascal subprograms (meaning literal descriptions may be used instead of syntactically correct code) and include descriptions of interfaces from COBOL II applications. The focus will be on how to use the method, not on writing actual code.

Two other specifications need to be made at this point. During the logging operation, the process will need exclusive access to the logfile while it is obtaining the pointer and adding the data record. Rather than lock the actual logfile, we will utilize an additional file for locking purposes. This is done because many dirty logfile pages in memory may cause serious performance problems when the process calls FUNLOCK. This file can be empty, since we are using it as a locking semaphore only. Tests have shown using FLOCK requires significantly fewer system resources than using a global RIN as a semaphore.

The other point to be addressed is keeping track of the pointer into the logfile. This is best done with a third file, also a Long Mapped file. As the pointer into the logfile increments, the new value will be stored in the pointer file. In order to start the logging at the beginning of a new logfile, a separate process should be run to store the initial pointer in the ponter file and initialize the logfile.

In the initialization portion of the application, the logfile, the lock file, and the pointer file need to be defined and opened. They need to be opened with Share access, Read/Write, and the lock file needs to be allowed Dynamic Locking. The COBOL application should have the following definitions for the logfile in place:

```
01  Filenum                  PIC  S9(8)     Comp Value 0.
01  HPFOPEN-Status.
    02  HPFOPEN-Status-L     PIC  S9(4)     Comp Value 0.
    02  HPFOPEN-Status-R     PIC  S9(4)     Comp Value 0.
01  Filename-Option          PIC  S9(8)     Comp Value 2.
01  Filename                 PIC  X(26)  Value "%Logfilename%".
01  Foption-Option           PIC  S9(8)     Comp Value 3.
01  Foption                  PIC  S9(8)     Comp Value 1.
01  Shareoption-Option       PIC  S9(8)     Comp Value 13.
01  Shareoption              PIC  S9(8)     Comp Value 3
01  Aoption-Option           PIC  S9(8)     Comp Value 11.
01  Aoption                  PIC  S9(8)     Comp Value 4.
01  Fileptr-Option           PIC  S9(8)     Comp Value 21.
01  Fileptr                  PIC  S9(18)    Comp.
```

The initial call to open the logfile from COBOL would look similar to the following:

```
Call Intrinsic "HPFOPEN" using Filenum HPFOPEN-Status
     Filename-Option Filename Foption-Option Foption
     Shareoption-Option Shareoption Aoption-Option Aoption
     Fileptr-Option Fileptr.
```

The definition of and initial call to open the pointer file will be identical to that of the logfile, with the exception of the name, file number, and file pointer. The pointer file should be built large enough to hold a 64 bit pointer and be in binary format. The following BUILD command should suffice:

:BUILD ***PTRFILE*****;REC=8,1,F,BINARY;DISC=5,1,1.**

The file used for locking can be a flat file of any description. It will also be defined similarly to the logfile and the pointer file with the exception of allowing Dynamic Locking. The additional COBOL definition for Dynamic locking would be:

```
01   Lockoption-Option    PIC  S9(8)   Comp Value 12.
01   Lockoption           PIC  S9(8)   Comp Value 1.
```

The reason separate pointer and locking files are used is due to FUNLOCK causing dirty pages to be posted to disk each time it is called. Naturally, the pointer file will have a dirty page each time the beginning or ending of a transaction is logged. If the two files were combined, an extra disk I/O would be generated for each call to the logging subprogram (two per transaction logged).

Once the files have all been opened, some initialization is in order. At this point, the program will know the PIN and user information. This can be loaded into the record that will eventually be passed to the Pascal logging subprogram later. It should be kept globally so it does not have to be loaded each time a transaction is logged. The information can be obtain using the JOBINFO and PROCINFO intrinsics.

Once the program is ready to call the logging subprogram, either at the beginning or the ending of a transaction, several activities should occur. First, the transaction

name, number and the Begin/End flag should be moved into the record to be passed to the logging subprogram.

Next, the program will need to gain exclusive access to the logfile by issuing an unconditional lock via the FLOCK intrinsic. The file is locked by the calling routine so it doesn't have to pass the file information to the logging subprogram, which would be unnecessary overhead. FLOCK should be called as the last action prior to calling the Pascal logging subprogram.

At this point, the logging subprogram can be called, as follows:

Call "move2log" using Pointptr Rec.

Where **Pointptr** is the pointer into the pointer file and **Rec** is the data record with the information to be stored in the logfile.

The following Pascal/XL code defines **move2log**. It is not entirely complete, however, there is enough to show how to write the routine.

```
$Standard_Level "Ext_Modcal"$
$Subprogram
Program dummy-outer-block;
                                    { Not Complete! }
  Type

    buf_type = record

                    xact_name       :  Packed Array[1..10] of Char;
                    xact_number     :  integer;
                    pin             :  integer
                    user_info       :  Packed Array[1..26] of Char;
                    b_e_flag        :  Packed Array[1..2] of Char;
                    date            :  integer;
                    time            :  integer;
                    cpu_time        :  integer;
                    ;

        { This buffer can be whatever size and combination of types as
          necessary to contain the data to be logged, but be sure the
          COBOL definitions match. }
```

```
        ptr_type = globalanyptr;

Procedure move2log (VAR fileptr     :   globalanyptr;
                    VAR file_rec    :   buf_type);

VAR

    rec-ptr    :   ^$EXTNADDR$ buf_type;

    temp_ptr   :   ^$EXTNADDR$ ptr_type;

BEGIN

    temp_ptr := fileptr;
    rec-ptr := temp_ptr^;   { Gets pointer into logfile }

    { At this point get Date, Time, and CPU Time }

    Calendar( file_rec.date );      { Date & Time stamp }
    file_rec.time := Timer;     { For Elapsed Time }
    file_rec.cpu_time := Proctime;     { For CPU Time }
    rec-ptr^ := file_rec;     { Puts info in logfile }

    rec-ptr := Addtopointer(rec-ptr,78);
    temp_ptr^ := rec-ptr;       { Updates logfile pointer in pointer file }

END;
BEGIN
END.
```

It would be advisable to add end-of-file checking for the logfile prior to attempting to add a record to the logfile.

Finally, don't forget to call FUNLOCK to release the exclusive hold on the fileset after returning to the calling COBOL routine from **move2log**.

Now that we have seen how to perform the logging process itself, several other aspects need to be discussed. First, whether or not the logfile should be attached to the Transaction Manager. The Transaction Manager will ensure the logfile is kept up to date within a minimum of one second before a system crash. Since this

process utilizes Mapped files, many of the logfile records may still reside in memory and not been posted to disk. The Transaction Manager ensures the data will be posted to disk. However, there will be a slight performance penalty for using the Transaction Manager, which will be most noticeable during the Checkpoint process.

The real task is to question the value of this data, especially if only an hour or two of data is lost. It is not, generally, crucial to business operations. The goal is to try to ensure the lowest overhead possible for the logging process. It is also possible to try logging with and without the logfile attached to the Transaction Manager. If there is no noticeable difference in performance, attaching the logfile to the Transaction Manager may be justifiable.

Next, due to the way mapped files exist, it is difficult and somewhat expensive to update the EOF marker after each transaction is logged. To avoid this, use a "fill" method when building the logfile. For example, as part of the logfile creation process, fill the logfile with a meaningless character or series of characters. The other benefit of doing this comes during the reporting process. The program generating the report can look for this meaningless data as an indication of reaching the end of valid data.

It should be noted, at this point, the logfile should be replaced with a newly created logfile after the reporting process has completed.

The final point to be discussed deals with the reporting process. With the logfile created, the information will need to be extracted and reported in some fashion. The frequency of the execution of the reporting process will depend upon several variables, such as length of time to generate a report, necessity of timeliness, and availability of system capacity during off-peak hours.

This paper will not go into great detail regarding the reporting process. Many options exist from home-grown programs to third party report generators. An alternative may be to summarize the data and upload it to a Personal Computer so that it can be represented in a graphical format. Actually, for on-going analysis, this may be the most attractive alternative.

A brief explanation of the calculation of Response Time and CPU Time for transactions is in order. The reporting process must match up the beginning and ending log records for each transaction. This is best done utilizing the Transaction Number stored in the logfile record. Once they have been matched, it is a simple matter of subtracting the beginning time from the ending time to obtain the result.

Given the way the logfile has been constructed, it is possible to window in on certain types of transactions during certain processing periods. It is also possible to gather information about a single user or group of users. The point is this, with careful planning, a large amount of information can be made available while keeping the overhead of collection to a minimum. The information can be used to gauge response time patterns, to understand which transactions are the most intensive, and to plot a course for the future.

Acknowledgement

I want to thank Larry Kemp, HP in Belleview, for answering some Pascal questions. He also provided the basis of the code examples in this paper through his class, *Optimizing Applications Around the MPE XL File System*.

Bibliography

Hewlett-Packard Company — Optimizing Applications Around the MPE XL File System class, 1990.

Hewlett-Packard Company — MPE XL Intrinsics Reference Manual, Third Edition, April, 1990, Chapter 4.

Hewlett-Packard Company — HP Pascal Programmer's Guide, Fourth Edition, October, 1988, Chapters 3, 4, and 10.

Remote Performance Management for HP Systems

Jay Mellman
Application Support Division
Hewlett-Packard Company
100 Mayfield Avenue
Mountain View, CA 94043
(415) 691-5759

Performance management has traditionally been, if not a difficult task, certainly an abstract one. It was characterized by voluminous amounts of data, a tremendous difficulty of mapping this data to user workloads, and a perception by management of performance as a foreign language. If that is true, then, remote performance management was mostly a dream. The thought of being located apart from the systems and users created unease for the performance professional; it implied a total lack of control over the systems environment. When you take these issues together, it's no wonder that most organizations avoided the remote aspect altogether. However, with today's technology, remote performance management can become a part of normal system management activity. This paper discusses the issues involved in planning and implementing remote management.

The Remote Environment

More and more, distributed computing is becoming a forceful trend in organizations. Organizations feel that they can better serve business needs by placing the appropriate computing resources exactly where they are needed. A decade ago, minicomputers fit in to the move toward departmental computing; today, that trend includes moves to the desktop and the smaller operation, creating very complex environments. At the same time, firms are concerned the high and

rising costs of managing these complex environments, especially as the cost of these systems declines. Organizations are centralizing their management effort in order to reduce staffing costs, eliminate duplication of efforts, and leverage expertise across the organization.

Some of these environments may be "lights out," for example, a point-of-sale system in a chain of stores. All aspects of the operation must be handled automatically or by a system manager performing tasks over a network or phone line. The company has no desire to place systems staff at each location, and yet they intend for all necessary activities to occur. Such an environment creates specific needs for all activities including performance diagnosis and planning. That means more challenging work for the systems manager as he or she attempts to manage the performance of these systems from afar.

Other environments also involve what we'll call remote management. Fewer system are located in or managed completely by a single data center. A manufacturing plant may run automatically most of the time, with responsibility for managing that system on a day-to-day basis residing with the staff at that location. But if you are part of a corporate systems staff that needs data for yearly business planning, you may require access to that performance data from a remote location. Likewise, consider a network of workstations and servers dispersed throughout a campus. You may be responsible for managing those servers, including answering user complaints and the management desire for planning and reporting even though these systems support a variety of departments doing a variety of different work. In any of these cases, it may be your job to ensure that the systems serve the users

effectively. Below are some guidelines to follow as you strive to manage these remote environments effectively.

Understand Your Business

The key to successful performance management, and especially remote management, is understanding the business environment. The first step is to investigate the areas of stability and instability within the remote environment. Environments with a great deal of variability or change require different system management strategies. For example, a system responsible for inventory and warehouse management may have a relatively stable workload during the day. On the other hand, the workload on a system running an order processing department could vary greatly. There may also be significant seasonal considerations as well: a point-of-sale system during the holiday season may be quite a bit busier than normal.

The second step in setting up a plan is to characterize the demands placed upon the remote system. You probably have an intuitive sense of which systems, applications, or users are your biggest headaches or are undergoing the greatest changes. In a remote environment, understanding the applications running and characterizing the demands placed upon them is critical. You, the system manager, must understand what business activity drives your users' applications and workloads. Let's look at an example:

Suppose we know that the payroll department tends to complain about their throughput at the end of the month. How are we notified of their problem? Generally, they will call the help-desk complaining about slow systems. We don't

need to invest in a systems planning project. A little experience and investigation tells you that payroll is processing our paychecks and that the payroll application is driven by the number of employees. There are several ways to make good use of that information. Short term, we can work around their need by either freeing up resources or changing other users' behaviors. Longer term, we can plan for equipment needs as the number of employees and the activity of our payroll application grow.

This example demonstrates that to accomplish effective performance management of systems, understanding the drivers of your workload is critical, especially in a remote operation. Your goal is to have your systems operate within acceptable limits, in this case, to reduce complaints. Experts within your organization probably can already characterize the business activity. Combining your knowledge of system performance with your programmers and application users leads to a constructive understanding of the system dynamics.

Given these business needs described above, the third step is to look into setting standards for your system performance. How many transactions per hour does the payroll department need? What kind of CPU power is necessary to support our programming staff? What kind of response time is the order processing team going to expect? These questions have shared owners. As stated above, you need to work with those partners to understand their requirements.

Set Standards

Next, you need to break down these standards into what would be ideal and what is realistic. Perhaps the warehouse needs to process several hundred orders

per day, all with multiple items. Ideally, that organization may want one second response time. Using your knowledge of the systems environment and working with the warehouse team, you may set a realistic goal of three seconds. Or maybe your short term goal is to reduce phone complaints from that group. Some customers, especially those familiar with IBM and VM systems, are looking to guarantee particular response levels to specific users. As long as you are able to distinguish between ideal and realistic, you can set shared standards to manage a system remotely to meet your goals over time.

Second, there are some statistics that you may use personally (e.g. to highlight a potential problem) and others that you will use to communicate effectively. There are some system statistics that you will need for problem solving. You may find it helpful to know whenever there are more than 50 active sessions. There are also indicators which are powerful tools for negotiating with a department to change their batch processing or help sell a system upgrade to your management. Does it make sense to talk to the accounting department about CPU utilization? About memory management? It may make more sense to talk to them about their transactions per hour. Other statistics may be needed to show that same department that they use too much of a given system with their daytime batch jobs. In a complete remote management plan, both of these kinds of indicators will be desirable and should be thought of ahead of time.

Understand Response Characteristics

Another key to successful remote management is to characterize your data center's response to a systems problem. Let's say that you have a system located in another site managing a warehouse. What happens when there is a problem with

that system? How do you find out about the problem? Do you have any operations staff there to address to problem? Are you responsible only for oversight of that system? Planning for upgrades? These questions help define the amount and quality of information you need to gain control over your systems environment.

One scenario might be that you oversee a relatively sophisticated local staff. In this case, reviewing data monthly may be enough. If the staff has some experience with basic issues of performance, you may only be involved on rare occasions. You may be called upon to help with capacity planning, while most issues and user demands are handled by the local operations staff. On the other extreme, there may be no local staff at all: a lights-out environment. As described above, not only do you need to have a clear sense of the application and its dynamic, but you will tend to need a high level of specific information on a regular basis to avoid performance degradation. A lights-out environment forces you to be proactive in your approach.

Even in cases where you have a local operations staff in place, planning is necessary. A staff can easily schedule jobs incorrectly, miss console messages, or simply not have enough training. For example, if you rely on an operations person to call you once the system has ground to a halt, you will be reacting to crisis after crisis. At best, having a well-trained staff means having a system operate within your specifications most of the time. At worst, you may personally assume the fire fighting responsibility, as well as for recognizing and planning to avoid these situations. But overall, the less you have a staff to handle performance issues for you, the more robust your remote management and information needs must be.

Consequences

The final characteristic to consider are the consequences of a performance degradation. For instance, let's look at a local operations person managing a system dedicated to checking out materials at a library. Such an environment is relatively stable (single application, limited throughput), you have some support capability should problems arise, and the consequences of a slowdown are not devastating (a line at the checkout counter). For this system, your major performance duty may indeed be planning for the future needs. Indeed, you may only need monthly performance data to keep this operation running smoothly. On the other hand, let's say that you manage a system in an urgent care facility from the main hospital data center. Here, you may have multiple applications (patient tracking, billing) with much uncertainty about traffic, no local operations staff to help, and larger consequences if things slow down or stop. As above, the more dire the consequences of poor performance, the more robust your plan should be.

Building a Reporting Process

With this framework, we can approach the process of getting you the information to manage the performance of remote systems. First, you need to define the frequency and the amount of detail in your reporting: how often do you want particular information? The library example implies infrequent data reporting needs at a summarized level; for the health center, detailed information is needed quite often. If you know that the center is beginning to process close to a defined number of transactions per hour, you may be flagged to look for potential alternatives prior to the calls complaining about a system slowdown. For this type of environment, besides a warning bell to get your attention, you may want hourly reports at a high level. If you understand the dynamics of the system, daily

summaries of system and application performance will help you stay ahead of potential problems by understanding when you are approaching some of your key standards.

These standards also point you directly to the kinds of data you require. Based on the particular characteristics of the system, is CPU utilization required every five minutes? Are there key numbers of sessions or logons that signal a problem or designate an important trend? Keeping in mind our discussion of personal versus communicated metrics, what kind of data should you be aware of on an ongoing basis and what do you expect your management will need to know? The process of defining your data requirements is an iterative one. In some cases, you will find that there are two or three pieces of information that give you tremendous understanding and control over that system. In other cases, a little bit of data will lead you to need more specific system information on a more regular basis. The key is to know what information you need, for what purpose, and how often you need it.

Technical solutions

There are many issues in planning a remote performance program. Overall, the first effort is to characterize the demands placed on the remote system. Working with your users, you need to understand what business activity, changes, and growth may occur. Only then, can you take other issues into account. To summarize, these issues are:

- How stable is the remote environment? the less stability in terms of system activity, the more proactive you need to be;
- how critical are the applications in the remote environment? the more critical, the more robust your information and tracking needs will be;

- what sort of staffing is in place in the remote environment? the less you can count on a staff to handle performance issues, the further ahead of the problems you will need to be;
- what kinds of standards are you going to apply to the remote systems? are you guaranteeing certain levels of performance to either users or management? will you rely on user complaints to drive your actions? what are your management's expectations about system performance and new purchases of equipment?

With these questions in mind, what data do you need to see? Then, how do you get the information you've identified?

In the beginning, there were only system monitoring tools. Examples might be OPT/3000, ps, or FREE5. To get information, you had to log on to the remote system and be prepared to wait until the problem reoccurred. In some cases, you had to rely on a staff to call you and tell you what was on the terminal screen. The logging capability of a tool like OPT/3000 gave some insight into the past, but the nature of the tool impeded quick analysis. In the UNIX environment, a tool like sar(1) helped, but in general, performance management was an art. The performance tools available did not lend themselves to a remote performance analysis, even if you did invest in a monitoring tool. A monitoring tool generally did not reduce the amount of data you needed to work with. More importantly, it was a sizable effort to meet some other needs described above: relating system workloads to user activities, communicating effectively with management, or reducing problems by staying ahead of performance issues and trends.

Today, the situation is much improved, as tools now help the user deal with these issues. A tool such as HP LaserRX provides trending capability for both systems and application statistics and begins to help in relating system activity to

user behaviors. Vendors recognized that they could present information in a form that management and users could understand. Tools were also designed with logging or continuous collection in mind. These tools could be used to quickly diagnose reoccurring problems and uncover situations that had already resolved themselves (e.g. a slow batch job or a slow print queue on a network server). Tools also began allowing the user to access performance data over a wide variety of network and serial configurations. You now had the power to deal with some issues remotely.

The next generation of tools meets even more of the needs of the remote environment. As stated above, a key part of managing systems remotely is to understand the dynamics of the given systems and devise key measures for flagging and managing problems. In doing so, there ought to be some sort of regular reporting to provide that base of understanding. Incorporating new features, HP LaserRX and HP NewWave agent technology can automatically provide reporting on remote systems. Once you have set your standards for a remote system (e.g. the accounting application should not use more than 20% of CPU or no more than 4 processes should be paused waiting for disk), current technology can easily provide the information. With these reports, you could provide information to a local operations staff, stay on top of user activity and demands, and be prepared to communicate whatever information management desires. You need to provide the business and systems understanding; the tools will provide summarized data broken down by workload where appropriate, easily transferred from the remote location, and displayed effectively without you manipulating the data.

Taking current technology to the next step, your understanding of the remote environment should let you set up exception-based reporting. Using the agent technology, you can set up guidelines within which you need to be notified. In the above example, let's say you need to know when disk queues begin growing; your realistic goal for the system is no more than 6 processes waiting for disk; you know from experience that system performance becomes unacceptable at that point if allowed to continue to too long. You can set up an agent task and an HP LaserRX macro to extract data from the system on an hourly basis and prompt you when the disk queue reaches 4, your personal standard. Hopefully, you will have the time to react to the situation before the users begin calling or the operations grind to a halt. And as has been said, your reporting is based on the characteristics of your remote environment and the standards you need to achieve.

The state of current technology for the HP market is that remote performance management requires human interaction. Some may never go away. In a networked environment, perhaps alarms will register on a network map, allowing system performance to reside alongside of fault management and isolation as a management task. For other situations, the computer might notify you through a beeper or even voice mail. Going one step further, artificial intelligence and rule-based modeling will allow you to define steps for the system to take on its own. For instance, should conditions indicate an imminent slowdown of the order processing application, the system could follow your wishes by sending a message to the programmers as it lowers their program priority in favor of the higher priority activity. These techniques, while relieving you of much day-to-day action, will place an even higher requirement on the need to understand your business and systems environment.

The technology exists today for you to make remote performance management a part of your system management job. From an MIS point of view, this is consistent with pressures to reduce data center management costs and to move toward distributed computing. As stated at the beginning of this article, the key to successful performance management is understanding the business environment and how your users use the systems. By accurately characterizing your remote systems environment, building in and collecting the appropriate measures, and communicating these effectively, you can build your credibility and success. The costs of following these guidelines are not overwhelming, but the benefits can be. You will increase user satisfaction with their systems, meet business goals for MIS in terms of cost reduction or profit improvement, and be better able to meet the changing nature of most businesses.

Developing With The User In Mind

Lisa Burns
Hewlett-Packard
Open Systems Software Division
19447 Pruneridge Ave.
Cupertino, CA 95014

A few years ago I had the opportunity to write an application for Stanford University as a volunteer project. I wrote a data entry program using Lotus 123 and the HP PortablePlus laptop computer. It was a great learning experience -- I learned Lotus and macros, and the Stanford fundraising office learned to use PC's for pledge tracking. However, a very interesting thing happened to me while pilot testing the program: I had to use it! It was an awful shock to realize that my masterpiece was a tremendous pain to work with. The program was easy to write and change, but awkward and clumsy for data entry. A lot of tabbing was required, mistakes were easy to make, and data entry took a long time. Training the Stanford staff was more difficult than I had expected and many of them were reluctant to use the package until it was improved. I had made the mistake of writing the data entry package from a programmer's perspective, not from a user's perspective.

Fortunately for Hewlett-Packard, project teams in our area take a different approach than I took with Stanford. We work closely with users and with user representatives throughout our development process to ensure that the user interface on our business applications fits efficiently into the data entry process and that needs for efficiency, keystroke reduction and productivity are met. In this article, I'll describe the process we use to maintain that user perspective throughout the entire software development lifecycle and give you some things to consider in order to work with users in your own shop. Finally, I'll discuss the results we have had with user involvement in HP's sales systems area.

The first thing we consider setting up a project in our shop is, "Who are our users?" This may seem like a simple question, but sometimes the answer is far from clear. For your own shop, you need to consider your primary users first of all. These are the people who use your system directly, working with your data entry screens, your reports, your online inquiry and other system functions. Next, however, you need to look at those departments and people who use your data, who need summary information, or who read your database directly. These people must also be involved in your design activities. For example, a work order processing system may have work order data entry people and

maintenance staff as primary users. These people enter and retrieve data directly from screens and reports. However, accounting workers who book maintenance billings against the general ledger are indirect users of work order data and should be represented in a system design.

Once you have identified your user base, you need to plan the involvement of representatives of that base in your development process. If you sit next to your users, as many MIS folks are lucky enough to do, you may simply need to plan weekly meetings with them to clarify needs and review specifications. If your users are scattered across several cities or countries, as ours are, this may not be practical. Instead of weekly meetings, you may need to set up telephone conferences, video conferences or perhaps monthly meetings to discuss system plans. Be sure that your plan includes a cross section of your user base. For example, this may mean getting both data entry people and management report users involved in your meetings, or involving one sales rep from each district. For a multi-national situation, you may need to involve users from several countries.

Whatever your plan, however, a key ingredient is management commitment to the plan. If you need two hours of an accounting clerk's time once a week, you must have support from that person's manager. The manager must agree to have someone else answer the phone and perform that person's tasks while they are in your meeting. Similarly, if you are asking a person to attend a monthly meeting hundreds of miles away once a month, you will need travel money as well as management commitment to the time away from the user's job. Even a phone conference requires support. Establish the time and money commitment with user management before you go any further with your project.

Another option to consider when setting up user involvement is to avoid the necessity of borrowing time from a user by hiring them instead! By including former users on the project team itself, you have dedicated resources at your disposal. You need not wait for a meeting or conference to clarify a point or check a specification. Also, these user advocates will likely have many contacts from their old job. They can thus research problems and clarify questions quickly. Most important, because they have lived the life of a system user, they will have a completely different perspective than that of your programmers. This perspective is critical to your success.

Finally, you need to establish authority for specification and implementation phase signoffs. Although you have been working closely with endusers, data entry folks and report users, they probably are not the ones who have authority to approve your system. For small shops, it may be the accounting manager who will review and agree to your system specifications. For larger operations, you may need representatives from each store in your chain, for example, to review and approve the test plan or other document. Whether you need to talk to Fred down the hall or get fifteen signatures from the review committee, set this up up front.

Once you have the groundwork done, it's time to start investigating and designing. Our life cycle, which is probably similar to your own, goes from investigation to external specifications, to internal specifications, followed by coding and testing, and finally user alpha site testing. During your investigation, you will want to fully understand and document existing information flows and system designs. In addition, you will want to ask about functions missing from current systems and procedures. This will probably mean that programmers need to watch users performing their jobs even if you have users on the project team itself. Additionally, you may interview the users you have hired or arranged time with, and have them explain their business process to you.

Once you have completed the investigation and generated whichever documents or flowcharts are needed, design work begins. This is where user involvement is by far the most critical. As you prepare draft screen and report layouts, and as you begin describing the functionality of your system, work closely with the users you identified earlier. Ask them to check the placement of fields on screens and make sure that you do not require the user to tab all over the place to get to a frequently used field. Be sure that reports are organized intuitively and are easy to understand. Above all, ask the user representatives to make sure that programmers have not overlooked a critical function or field. Are add and delete functions adequate, for example, or is a modify function required? Ask the users to test the completeness of your designs by looking at example transactions or inquiries from their jobs.

During the system design phase, you may want to consider different tools to help you check the proposed software with your users. Hands-on demos and prototypes may be especially helpful in this area, as may mockups of reports. We have also used phone conferences and video conferences to check our designs with users from many different geographies and functions. Finally, by involving user representatives in software inspections (also called structured walk-throughs or peer review meetings), you can uncover and prevent system design defects and ensure that all designs are adequately reviewed.

At the end of this design phase, you will probably produce an external specifications document, or at the very least, a set of screens, edits and reports. When this is published to your users, you will undoubtedly be asked to change it -- in fact, if you are not asked to change it, the users may not have read it! Because of the upfront work you have done reviewing and prototyping your system with various user representatives, you will hopefully have a proposed system very close to user needs, and the changes requested will be minor and easy to incorporate. Assuming this is the case, you must now make the changes if appropriate and set expectations firmly that additional changes will become more expensive as time goes on. The phrase, "speak now or forever hold your peace" is probably appropriate about now. While completely frozen specifications are impossible, you should aim for something pretty "slushy" before you get user signoff on the ES.

Now it's time to write the internal specifications for your system. This is mainly a technical activity, so user involvement will be less at this stage. You may be able to give your users a break, or if you have hired user representatives, you can start them writing the documentation for the system. Programmers may still occasionally need to contact them for clarification on edits or system functionality, however.

Once you have completed pseudo-code, modules narratives or hierarchy charts for your system, and you begin coding and testing, user involvement will increase again. You will need to include users in test plan writing and execution. This is another critical task. If you are like me, you are not very good at thinking up realistic test cases. For example, the only HP product number I know off the top of my head is the 2225A, the Thinkjet. Sales orders with 20 line items, all 2225A's, do not make very representative test cases. For this reason, taking sample transactions from your users' production workload is a great idea. User representatives can help you test error conditions, boundary conditions, as well as realistic production transactions. You will probably want to include them in software inspections of your test plans, and you may want to have them actually execute the unit tests themselves.

One caution here: don't leave your programmers completely out of the unit test execution. Having to actually use your own software can have a very sobering effect. By testing your own code, you can begin to see why having to tab over 15 fields to get to the one you need is an annoyance. Programmers working with users is probably the best combination for testing.

As you approach the end of the construction phase and are executing the final unit test plans and fixing minor defects, you will want to involve the users in writing your system test plan. Again, they will help ensure that you are covering all production cases as well as error conditions. In addition, they will help you make sure that your test environment matches a true user environment. For example, they can help you in storing copies of user data files, databases, account structure and user lists, and restoring them onto your test machine or into your test account. This will make sure that your test environment is as close as possible to the one you will encounter in production. Also, users can help you exercise the system's complete functionality -- all kinds of transactions, data flows and interfaces. You will certainly want to have them review your system test plan document, and you may have them run the test itself.

One important thing to keep in mind during the system test is that you may need to involve programmers and users from other functional areas during the test. For example, if you are testing a new invoicing system which updates the accounts receivable files, you may need to involve users from the AR department as well as the invoicing area. Be sure to plan with the AR department well in advance to arrange for these people to help you. State very clearly the time and task requirements so that there are no surprises later.

As the system test winds down, you should be meeting with the users who will actually run the production test of the new software. Testing a new system requires time away from their jobs and will mean extra effort for both the users and the development team. Be sure that you cover how each feature will be tested, whether parallel procedures are involved for an initial time, and how to move from parallel to production. Also, remember that they need not only to test the software itself, but also the training and documentation for the system. They will also test your software installation as well as the installation procedures. The end users probably have better ideas than you do about how to test the system's functionality -- let them decide what will work best.

Next, decide how changes will be made to the new software during the production test. You may need to establish criteria for critical needs vs. enhancements. You will also need to decide how and when to install any changes to the software. For example, will the development team simply move the new code onto the production machine or account? Will tapes be given to the system manager? Will PC users be given new floppies? How will versions be tracked? These things sound simple, but if they are not planned, many bad feelings may result.

Finally, decide how to get signoff on the new system. Will you need to bring in all fifteen user council members for a review meeting, or will buying Fred a beer be enough? How will any open issues be resolved? By establishing these procedures before the start of the production test, you will avoid many misunderstandings.

In our shop, we have found that the steps listed above work very well. There seems to be a very direct correlation between user involvement and the subsequent success of the project.

Several of our project teams have gotten commitments from user management to have users spend time reviewing documents or helping the development team by answering questions. One of these project teams has taken the approach of setting up a user council. Representatives from each of the several user sites meet once a month or so throughout the life of the project to review documents from the development team. At these meetings, they also may view prototypes, suggest test cases, etc. This approach has worked well for the team. Their project is currently in production test at a user site in the Midwest, and only a few defects have been found, all minor. The users are happy with the functionality and with the "look and feel" of the new system.

My own project team has been lucky enough to hire some former users onto our staff. These people, our support team, provide a tremendous resource to the programmers. They have helped us analyze data entry transactions to determine how many VPLUS screens are needed for a given transaction, which fields are needed and how long the fields should be, where the fields should be placed, and how the softkeys should work. They have analyzed report layouts for field length, field placement and overall legibility. They have reviewed prototypes with current users

in our various sites, and worked with the programmers to make changes to our software. Working with our user council, they have researched issues and department procedures which affect us. They have inspected our ES, and have assisted with specifying edits and data flows for our IS. Now that we are writing unit test plans, they are inspecting them and helping us come up with test data. They will do the same for our system test. Since they are so involved, they have been able to write accurate and well-designed training and documentation materials for the new system. Our users are very happy with the team, largely due to the efforts of the support team. We expect the software to be very well received, as other projects we developed this way have been.

I hope that the information in this article allows you to increase user involvement in your own software development process. We have found that programmers, unfortunately, think like programmers. Users can help you see your software through the eyes of someone who works with your data entry screens 40 hours a week. The difference in customer satisfaction is dramatic. Remember, users are the reason for your software, the reason for your job, the reason for your paycheck. Hold on to that paycheck by keeping them involved.

A version of this paper appeared originally in HP Professional magazine.

Paper - 3140

Porting the UNIX System Environment to MPE/XL

Jay Zimmett

Quest Software
610 Newport Center Drive
Suite 890
Newport Beach, CA. 92660
714-720-1434

Porting the UNIX System Environment to MPE/XL

Jay Zimmett
Quest Software

Quest Software recently completed the porting of a major networking package (Portable Netware from UNIX to MPE/XL). Novell's Portable Netware software is designed to run under the UNIX Operating System with its Hierarchical File System, Streams Data Communications Services, Shared Memory Access, and other UNIX services. Since MPE/XL does not currently provide any of these services directly, an environment that provides these services had to be created. This paper will discuss the design strategies used to implement a UNIX environment under MPE/XL.

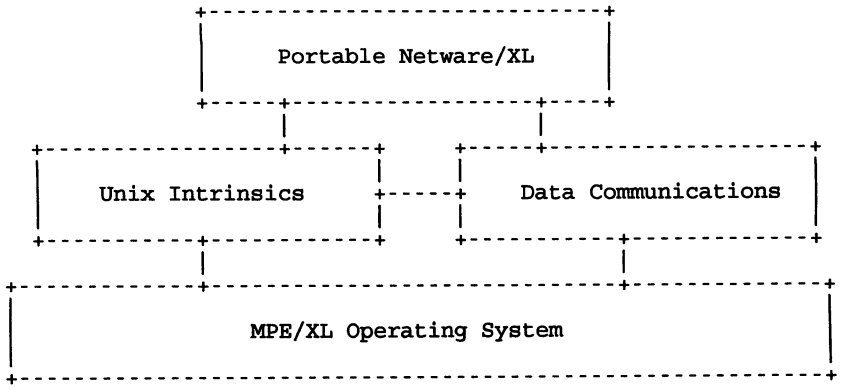
CONVERSION

The project was to convert Portable Netware LAN software to run on the Hewlett-Packard Spectrum line of computers.

Portable Netware consists of several hundred source modules written in "C". The first step was to get the source loaded on the HP system (development was done on a 935 using MPE/XL 2.2). The source modules were delivered in UNIX file format. They were restored to a PC using the Interactive Solutions version of UNIX. The files were then prepared, copied to DOS and then transmitted to the HP-3000. The HP compiler quickly allowed us to compile the Netware source, but the compiled programs had no operating environment in which to execute. Since our number one philosophy regarding the port was DON'T CHANGE THE SOURCE, it was decided to provide these services for the MPE/XL operating system.

UNIX INTRINSICS

HP's UNIX environment, POSIX (Portable Operating System Information Exchange) is still months away, and we needed to provide the same, or almost the same, functionality under MPE/XL that UNIX was providing. It was decided that a "Unix-like" environment would be placed between MPE/XL and Netware/XL.



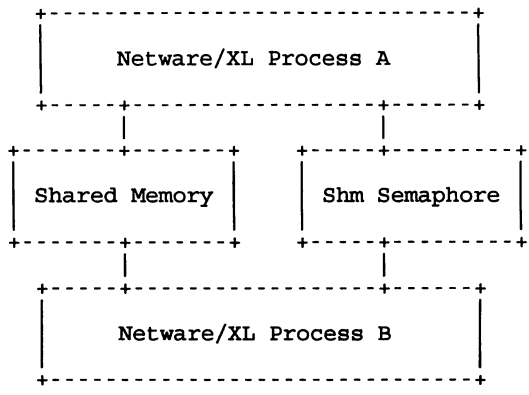
This environment would need to provide the following types of functionality: shared memory access, UNIX directory access, file information, signaling, interprocess communications, process creation, semaphoring and data communications (data communications was done by Jim Kramer and is not discussed in this paper).

SHARED MEMORY ACCESS

UNIX has a feature which allows you to request a segment of memory and to share that memory between processes. In order to use this feature, we needed to provide emulation for the following UNIX calls:

```
SHMGET  - Get a Shared Memory Segment.
SHMAT   - Attach the Segment to the calling Process.
SHMCTL  - Shared Memory Control Operations.
```

Implementation of shared memory operations was accomplished by creating a file of the size specified in the SHMGET call, and then accessing the file with short mapped pointers. This HPFOPEN option allows you to access a file as one huge array.



Mutual exclusion is accomplished by locking and unlocking an auxiliary file. Using an auxiliary file allows a major performance improvement over locking and unlocking the shared file itself. This is because a call to FUNLOCK will cause the MPE/XL operating system to post to disc any pages touched for writing if the file is opened in a modify access mode. This feature of MPE/XL's locking strategy results in a lot of unnecessary I/O when programs accessing the shared memory file are mostly reading the data, and doing very little updating.

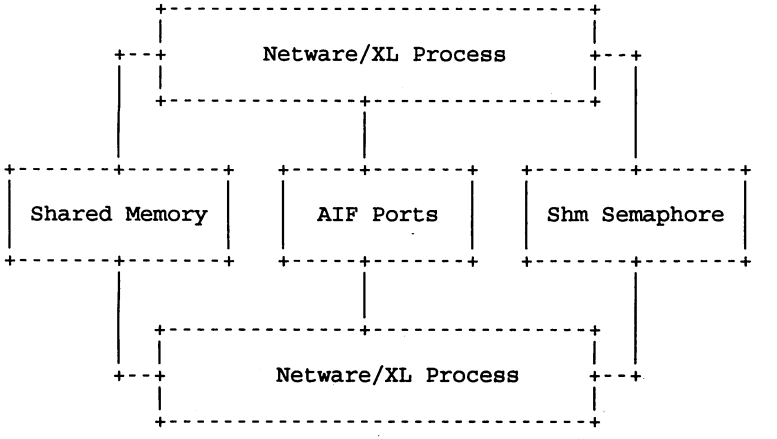
INTERPROCESS COMMUNICATION

Interprocess communication is a feature of UNIX that processes have of communicating between themselves. The UNIX calls that were provided are:

MSGGET - Get a Message Queue
MSGCTL - Message Control Operations
MSGSND - Send a Message
MSGRCV - Receive a Message

UNIX interprocess communication was implemented using HP's new product, the Architected Interface Facility (AIF). AIF's are a set of HP supported intrinsics that allow you "safe" access to MPE/XL internal information. The AIF port functions were used to create, send and receive messages. Message files

could have been used but message file access is still in compatibility mode and is, therefore, much slower.

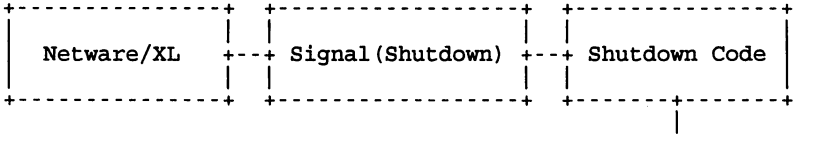


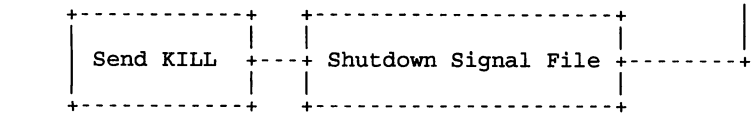
SIGNALS

Another nice feature of UNIX is the use of signals or process interrupts. This allows a process to set a signal function to be executed upon receipt of that specific signal. The signals that needed to be implmented were:

```
ALARM   - Set an Alarm Timer Trap
SIGNAL  - Set a system Signal Trap
SIGSET  - Set a process Signal Trap
SIGHOLD - Hold back Signal Trap
SIGRLSE - Release Held back Signal Trap
KILL    - Trip a Signal
```

Signal Trap





Signals were implemented using MPE/XL message files and using software interrupts that are set using the FCONTROL intrinsic. A process can set a signal by passing a signal number and an interrupt procedure address to the SIGNAL function. When this occurs a message file is created and a software interrupt procedure is attached to the file. Then when the KILL function is called a record is written to the file, thus generating the software interrupt and the execution of the user procedure. Alarm signals were implemented the same way, except that they use a timed read to generate the interrupt.

FILE SYSTEM INFORMATION

UNIX, like MPE/XL, provides information about the files that reside within its file domain. MPE/XL however reports things in words and sectors, while UNIX uses bytes. Numerous functions were needed; here are a few of them:

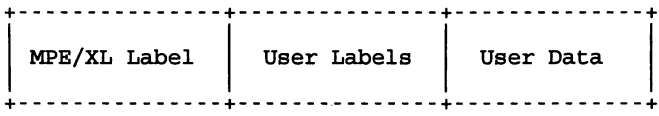
```
CUSERID - Login Name of User.
FSTAT   - File Information on an opened file.
GETUIED - User Capabilities.
GETPID  - Get Process ID.
GETENV  - Get Environment Variable.
PUTENV  - Put Environment Variable.
STAT    - File Information on a closed file.
STATFS  - Disc volume information.
```

CUSERID and GETUIED were easy to implement; a call to the WHO intrinsic satisfied both calls. GETPID is a call to PROCINFO to return the Process ID Number. GETENV is provided in the HP C library and PUTENV is a SETVAR done with the HPCICOMMAND intrinsic. STATFS is implemented using the new HPVOLINFO intrinsic (curiously I could never get the numbers to match the DiscFree command). FSTAT and STAT were not as easy.

FSTAT and STAT needed to return the file size in bytes, and the file's access, modify and creation times. The time had to be returned in Greenwich Mean Time, not MPE calendar format. Also, STAT needed to return this information on

files not currently open. One big item, MPE does not keep track of file size in bytes; there is no such thing as a one byte file!

NETWARE/XL File Format

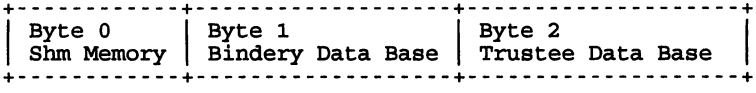


If the file is open, a call to FSTAT has the MPE/XL file number. Information on opened files is abundantly provided by the MPE/XL intrinsics. If file information is requested on a file that is closed, MPE/XL provides few ways of getting this information. Opening and then closing the file will provide the information needed, however there is the cost of opening and closing the file, not to mention the file's access times are then changed (remember, I need to return this information and if it is always changing, that is a problem). The solution is to write the information needed to user label zero at time of file creation. Luckily, the MPE/XL intrinsic FLABELINFO returns user label zero and all the rest of the information that I need without updating the access times. I then convert the dates into their new format and return the information requested.

SEMAPHORES

Netware/XL uses a UNIX function called FCNTL which among other things allows byte level locking of files. This can allow you to define bytes of a file as a process resource semaphore and as long as all processes honor this design a system resource locking mechanism can be devised.

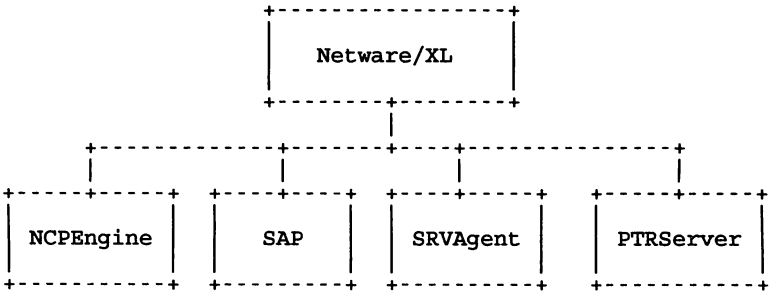
Netware/XL Semaphore File



This scheme of semaphoring was accomplished using MPE/XL file locks on empty files. FLOCK and FUNLOCK are extremely fast on a file open for read access only. MPE/XL internal lock mechanisms will be used at a later time.

PROCESS CREATION

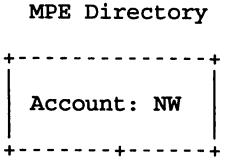
The needs of Portable Netware/XL for process spawning was straightforward. Netware/XL needed to create sons as peer processes. The CREATEPROCESS intrinsic was used to launch sons with the Father not waiting on termination of the sons.

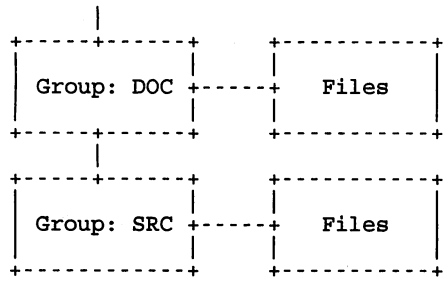


This allows Netware/XL to still function and communicate to its sons using shared memory and AIF ports. Netware/XL will start and stop NCPEngines as needed to services clients.

UNIX DIRECTORY

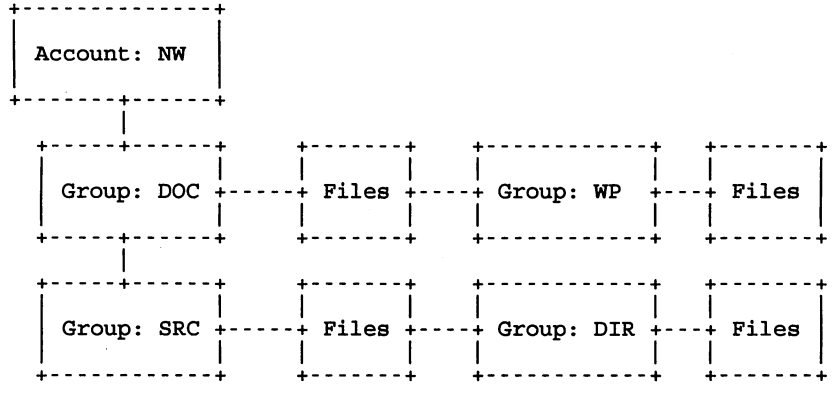
MPE/XL currently only supports a two level hierarchical directory which we know as Group and Accounts.





UNIX on the other hand supports a multi-level hierarchical directory which can support other directories.

Unix Directory

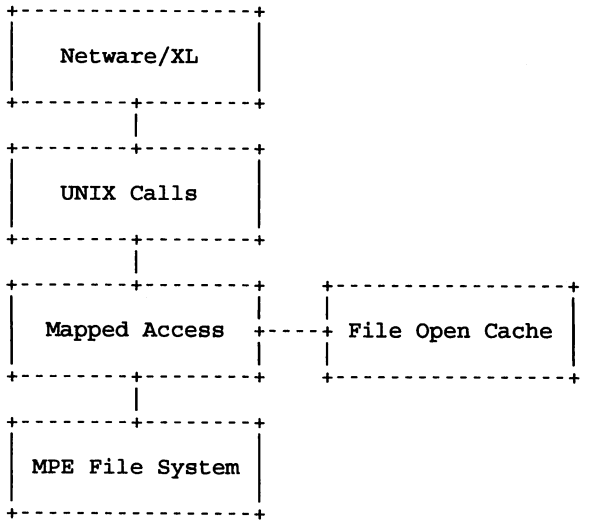


How do you map a Multi-level directory on a Two-level fixed directory? The key word here is map. A translation is needed between file names such as /usr/netware/sys/net$obj.sys and names that MPE/XL will accept. KSAM/XL provides such a mechanism where I can pseudo build a hierarchical directory within the KSAM file that maps to a valid MPE/XL file name.

KSAM DIRECTORY ENTRY

MPE/XL names are generated names such as A0000001.SYSVOL, A0000002.SYSVOL, and are mapped to UNIX path names using the KSAM/XL file.

In order to gain control for performance and 1 byte granularity all I/O functions to Portable Netware/XL files and user data files are done through Quest's own file system access methods using long mapped pointers.



Duplicate information about a file's history is kept with each file as a permanent user label which allows the rebuilding of the KSAM/XL directory in case of corruption. This should not occur as KSAM/XL is tied to the Transaction Management System which protects and automatically recovers

from a system failure.

NETWARE/XL OPTIMIZATION

Four HP tools were extremely helpful in debugging and optimization of the work that we had performed.

XDB	- Symbolic Source Level Debugger
GLANCEXL	- System Monitor
SPT	- Software Performance Tuner
DEBUG	- Old reliable debug

Without XDB I would have killed a couple of trees getting listings. Finally a source level debugger for the HP-3000! XDB is a must for any serious programing effort. GLANCEXL is a replacement for OPT/3000 for the classic and has been much enhanced. SPT traces the execution of your program and lets you know what procedures were executed and what MPE/XL intrinsics were called. Was a great benefit to see where time was being spent and at what cost. Good job HP, how about including monitoring of the AIF's? Debug took a side seat to XDB, but its nice to be able to crawl inside and really take a look at somethings.

CONCLUSION

This has been a most interesting project, and proves that just about anything can be done on the HP, even UNIX applications. As of this writing BETA code has been shipped and is working! By the time of the conference HP should be shipping NetWare for the HP-3000. Future products in the Networking area will be NFS (Network File System), it will allow UNIX systems to attach an HP-3000 account as a native UNIX file system and read/write to these files.

NOTE: UNIX is a registered trademark of AT&T

Getting Over the Hurdles of Oracle Financials on the HP3000's Hardware Platform

Craig P. Albrecht
Technical Systems Analyst
Corporate Information Center
Cray Research , Inc

ABSTRACT

Learn how to make your HP3000 adapt to the constraints of Oracle. This presentation details the information and techniques needed to make Oracle and HP3000 a more positive experience. Detail topics include CRT mapping to database creation. Learn the tools and options available on the HP3000 to monitor and manipulate the Oracle databases. Understand the kernel. Learn how to work with Oracle support staff rather than against them. When Oracle comes out with a major release update, understand how this impacts any customized changes that may have been made to the existing environment. Last of all, learn the potential power behind the "S"pecial "I"nterest "G"roup of MPE XL user of Oracle Financial Applications.

INTRODUCTION

In the early part of 1990, the financial users stated a need for a new financial system since they had outgrown the existing one. The choice was to write one in-house or to purchase one from a third-party vendor.

Corporate Information Center, CIC, has been up until now a traditional HP3000 COBOL shop since the early 80's. The data center managed a network of 8 HP3000's located world-wide with an additional 4 HP3000's residing at corporate.

The users began the search for a new financials software package. By mid-1990, Oracle Financials - Accounts Payable and Purchasing - was decided to be the software of choice by the financial users.

Oracle stated that their RDBMS and Financial Applications would run on the HP3000, 900 series, XL. Oracle was called in to load it and run a demo on our system to prove to CIC and the users that it indeed worked. The demo ended up being brief and incomplete in terms of testing out all the functionality and system specifics, like terminal testing, printing reports, etc. The users said they wanted it even though their testing was incomplete. The choice for the HP3000 hardware platform was chosen because: 1) Our expertise resided on that hardware platform, and 2) We had heard that the Transaction throughput was greater than that of Unix.

Needless to say , the birth of Oracle Financials began at Cray Research along with the CIC role of installing, support, debugging, bitching, headaches, ... started. To ensure the successful implementation of the decision, the Transition team was born. This team consisted of 1 manager/coordinator and 6 existing CIC employees that were the best in a given area:

networking, production control, systems analyst, ... For myself, technical and system knowledge.

This paper is intended for a new and/or existing site that will use/is using Oracle Financials on the HP3000 hardware platform. The issues that will be covered are more towards the technical side, excluding the application details.

SYSTEM : TERMINAL Device

We are dealing with 2 major factors: 1) The HP3000 runs certain applications in a BLOCK mode environment which is unique to HP. 2) Oracle runs based on an ANSI/VT mode.

The problem arises because most HP3000 shops have HP terminals but only have the HP mode option. Only certain HP terminals can function both with Oracle and HP block mode applications. The HP700/92 and HP2392A terminals can do both HP mode and ANSI/VT mode. One of the common problems, is sharing one terminal device between HP block mode and Oracle Financials applications.

Another dilemma was how Oracle maps the keys to the keyboard. Oracle defaults the numeric keypad to single-stoke function keys. They commonly map functionality keys to the numeric keypad. Our accounting users wanted the numeric keypad as such. Now the task of relocating these keys to a different part of the keyboard became the challenge. In total there are about 46 keys that can be used when running the Oracle Financials. However, there are only about 15 commonly used keys used by the end user.

First, the Oracle function keys on the numeric keypad were mapped to the F1-F8 function keys with a second set of Oracle function keys were also mapped to the F1-F8 keys but prefixed by a an <escape>. The users determine which keys they would want as single keystroke keys, the most commonly used keys, these Oracle function keys became the F1-F8 keys while the less commonly used keys are prefixed with an <escape> F1-F8.

Once the end financial users decide on how they want the key board to look like in terms of functionality, the challenge was to map all terminal device that could run Oracle Financials. The following devices could run Oracle Financials and old existing HP3000 software: IBM type PC using Reflections 3.4 or greater, Macintosh running Reflections 2+ or MicroPhones II (these are VT emulation only). Reflections 1+ for the Macintosh emulated HP mode.

When using Reflections with IBM-type PC's , an enhanced keyboard is recommended but not required. To assist in remapping the keyboard, you will have to use the tools supplied with Reflections when purchased: KEYMON, KEYMAP, R1V, VT.KBM to name a few. Note, each PC will need to have a special VT configuration file that contains the remapped keyboard keys, proper VT settings, the user COM# port, and the users choice of colors. Note, if any changes to this file are made, the keys MUST be remapped into this configuration file.

Getting Over the Hurdles of Oracle Financials on the HP3000's Hardware Platform

The Oracle Financial Software can run on IBM type PC running from an XT to an AT with an enhanced to an non-enhanced keyboard. The other key element to note, is what type of BIOS is being used, Standard or Enhanced. The BIOS will determine what codes from the keyboard are being generated for remapping. The BIOS and codes generated by the keyboard are determined by executing the Reflection program KEYMON.

The other problem was to make it easy for the end financial user to run the Oracle Financial software and any old application software that resides on the system. The use of HP command files make this easier.

SYSTEM : PRINTER Devices

Ever ask the Oracle porting group if they ever tested printers on this port? The answer probably will be yes, but I would not believe them. Printing has been a major issue since day 1 and still is an issue as of this writing.

Like terminals, plan on becoming intimate with your printer and what it's capable of doing and how you can programmically alter some of its functionality.

Oracle Financials tend to print reports at 66 lines per page or else x number of lines per page using the <FF> character. The only problem is, most HP printer are set up to print 60 lines per page with 3 blank lines at the top and bottom with automatic perf skipping. This is one option that must be turned OFF. There is an escape sequence to do this so read your printer manuals. Remember to align your paper in the printer at the perf because Oracle will need all 66 lines.

The funny thing, when Oracle shipped their printer routine, they shipped it with a print configuration file for a Desk Top laser printer. Ask yourself, how many production companies print a 500 page report on this type of device? We , the data processing staff, ended up telling Oracle how to set up certain printers to make the reports come out properly, sometimes! This showed their lack of knowledge and thorough testing...

Also, their reports are inconsistent in paging. If a report page is only 60 lines, 6 blank lines are printed before the next page begins. One problem, the reports are not all set up for 66 lines per page so the reports now tend to get misaligned, skewed. Hard to read and/or to misread reports.

Specialty forms are also another headache until you learn how to setup your printer because Oracle does not give any explanation. Examples would be checks, PO forms, ...

Now what about the HP2680A printer, the Laser Page Printer? Oracle did not even know what one was so it was never tested like so many other HP printers. This printer does not operate the same way as the majority of HP printers because it does not understand escape sequences. It

uses environment files and primitive paging done via CCTL characters like in Fortran.

Both HP and Oracle have been aware of this problem, but no resolution has been shared with me. I have ended up creating 4 new ENVironment files purely for Oracle reporting only. The form is setup for 132 columns by 66 lines per page. The only major exception is that a report must be reviewed for page break consistency - always at 66 lines per page.

SYSTEM : DISC Storage

Be prepare to invest in disc drives. Oracle loves to use the word megabytes as a unit of quantity.

The Oracle database runs best when you can allocate a large contiguous space for database files, although, you can have many small ones that look like one big chuck to Oracles RDBMS, although is less efficient.

One of the nicer things about Oracles RDBMS is that you can easily add space to the database if the limit has been reached. The real problem occurs on the HP3000 side of things.

How does one create contiguous spaces on the HP3000? On the pre-900 series HP3000, there was an option to do a disc condense in a relatively short time, without human intervention. Safe!

The only way to allocate a hugh chuck of contiguous space, is to do a full system dump, then purge the files except for the SYStem accounts. Before doing a restore of the full backup, create your Oracle database files. Plan ahead what databases are needed so that the files can be spread out onto different drives to prevent head contention. The more medium size disc drives, the better compared to few large, gigabyte, drives.

HP needs to come up with away of condensing files on disc drives. A reload type procedure is risky and very time consuming especially when you start getting into gigabytes of disc storage.

I have tried to use MPEX to shuffle files around from one drive to another without success. If the original file resides on drive A and I moved it to drive C, I verified the move using MPEX and the MPE LISTF command. The move was successful but when the free space on the drives is to be verified by the MPE DISCFREE command, it did NOT verify. It showed no change on drive A or C. It's as though the file was never moved.

If there is a solution, please let us all know.

SYSTEM : MEMORY

Oracle RDBMS uses what they call SGA, System Global Area. On the HP3000, this is a mapped file. It does not physically occupy all the space in main memory, but is used in conjunction with virtual memory. Up until about 5 months ago, there was a 2 megabyte maximum. This created many problems, especially if you had a large database and/or many users. By

configuring the SGA using the OINIT parameter file, which contains over 50 values that need to be changed for growth or else tuned for performance, caused the SGA to go over the 2 megabyte limit.

As one would think, this would demand a large size for the control area within the SGA. Oracle RDBMS is, in its own way, a separate piece that can be looked at as an addition to the base MPE operating system. It contains 4 system processes that run in the background once Oracle RDBMS is started/booted. These processes are: System monitor, Process monitor, Database writer, Log writer and possible Archiver, if turned on.

The new SGA limit is now 96 megabytes. This is the limit for 1 database and/or all the Oracle databases you have on your system in total. Oracle and Hewlett-Packed have worked together to make this possible. The only noted shortcoming, is to allocate this space. It must be allocated by a program and should be run after a boot of the HP operating system to guarantee the space is available because this space is also used by the MPE operating system. To ensure this space can be allocated, make it part of your system startup procedure, in your "STARTUP" UDC.

SYSTEM : CPU

Is there enough CPU on your system? Good question!

The nature of the Oracle application is to run in character mode. This means that every character that is typed is sent to the CPU to be processed. For the nature and functionality in terms of how it works , this is the way to implement. But, for the nature of the HP3000, this is a severe degradation of the systems performance because the HP3000 system was not designed for this type of IO. The HP3000 was designed for record IO and HP block IO.

Oracle and Hewlett-Packard both recognized this as a problem after many days of telling them of this problem. The solution that has been agreed upon is what they called Field mode. Certain fields will only be sent to the CPU when a special character is entered as the end-of-field terminator. An example would be to use the carriage return as an end-of-field terminator. Other fields may still operate in character mode. Other functions, like painting the screen will be enhanced to use non-character mode painting.

At this time, the RDBMS is ready for field mode, but the applications need to be changed to handle field mode. Also at this time, once field mode becomes part of the application, the functionality of the terminal keyboard will change some. This is due to lack of end-of-field terminators. Oracle is working with HP to have more, so there is more flexibility in terms how one can remap the keyboard.

From what I know, this feature is due to be out in the next release of financials, MPL 8.3.

SYSTEM : Connectivity

I just heard the latest news that this item should no longer be a problem, but I will explain anyway so you can verify this if your company fits into this category.

The problem was, to run Oracle Financials on the HP3000, the terminals used need to be directly connected to the DTC . One the reasons for this is the DTC had a new function that was implemented on the HP3000 900 series. This was called "typeahead". Without typeahead, the Oracle Application would not paint the screen properly and/or would force the end users to type about 1 character a second so that no key strokes would be lost, but the use of the function keys would also be lost if it sent more that 1 character to the CPU . In most cases, all but 4 function keys remained unaffected. This was NOT a acceptable solution.

Nowadays, most companies have more than 1 computer system and tie them together via a network. While networking HP3000's together via NS, you create virtual terminals on the remote machine via the host. Well, if Oracle Financials reside on your remote machine, you're out of luck. The functionality of "typeahead" is bypassed since access is though a network instead of a DTC. So if you wanted to run Oracle Financials, all users must be connected to the host machine, which will have to be the Oracle Financials machine and users MUST be connected via a DTC to the host.

NOTE: Oracle Financials must have "typeahead" when the end users are running the application only. Other applications on your system, host and remote, may be affect by "typeahead" where it could cause problems, for example HPDesk. When invoking Oracle Financials via a command file, turn "typeahead" ON and when ending the application, turn "typeahead" OFF. Typeahead is a system variable called: HPTYPEAHEAD TRUE/FALSE.

ORACLE FINANCIALS : ORGANIZATION of Files

The nature of the HP3000 file structure is: Accounts, Groups and Files. This is not like the Unix file system on which the majority of the ports reside. So this port needed to be reorganized to accommodate the existing file structure. The Unix file system contains many subdirectories, has longer directories/file names, not restricted to uppercase alpha numeric names. In Unix, the use of extension of filenames makes it nice because you can have the same basename in a directory but have different extensions. Example: A SQL script called "tablesz.sql" but the report/output file is called "tablesz.lst" all in the same subdirectory.

When a port is made, it's not quick and mistakes are made easily. The filenames are not always the easiest to convert to recognizable names, but the groups and accounts are. Most Oracle books are based on the Unix Operating system so one must do conversions to the HP3000 environment.

Example: AP7BN000 - This is a group name that refers to the Account Payable subsystem, MPL (Master Porting Library) 7.x, and the BIN (executable) subdirectory.

The current HP3000 Oracle data processing staff keeps on reminding the Oracle Porting team

to be careful to prevent errors.

ORACLE FINANCIALS : EXPERIENCE with HP3000

Oracle has been separated into 2 main groups: the RDBMS kernel group and the porting/support group.

The kernel group has been around for awhile, and is made up of people who used to work for Hewlett-Packard, and have vast knowledge. The RDBMS is sound in most HP data processing staff's opinion, but the Financial Applications are another matter.

The porting people to this hardware platform leave something to be desired. Although they are getting better, but slowly. At least they are getting some HP education.

The latest news that I heard is the 2 groups will be working more closely together instead of as separate identities. The kernel group will educate the porting/supporting groups more on the nature of the HP3000.

ORACLE FINANCIALS : World Wide Support/Contractors

With contractors, assuming they have prior experience, ask to speak to past clients for there recommendation and ask for their work to be guaranteed

Most contractors are fresh out of MBA school and have only theoretical knowledge of Oracle Financials. They lack experience with their own system. Sad! We have ended up educating some of them. If the procedures they create 'work', they end up being incorrect and/or poorly written. Non-professional and it shows.

Be careful. The good contractors are in demand, so book ahead. Be careful of new/inexperience ones. You want one who knows the given application better than yourself as well as actually knows it!

World Wide Support is improving by hiring qualified staff or by getting there current staff trained better. The problems are just getting through to speak to a trained support staff member. Getting solutions on the first try is rare but improving. Since this platform is relative new, the trained support staff is small and lacking experience both in the application and hardware. To get one of these people is a 40/60 chance.

Remember not to close a TAR unless you are satisfied with your answer. If having a problem with a support member in resolving your question, returning calls, etc., ask to speak to the duty manager. Inform your CSS manager of your situation. If you still have problems, go up the ladder of command.

A common problem that we have by being in a different time zone. Qualified support people

don't start work until 8-9am Pacific time. That's roughly 2 hours once our company starts its business day. Then they often return the calls at 5pm or later central time when no one is here. Reminding them of this situation has helped a little but still is a problem.

Your input is advised to Oracle about the support so they can improve. Use constructive criticism.

OTHER

When fix tapes arrive to correct bugs, they usually contain a bug number and instructions on how to run a script. This is NOT fine, the data processing people want to know more information.

The information need is a short description of what the patch is fixing because the bug numbers, do not have meaning to us. Information about what is being fixed is needed because a company may have made customized changes to the Oracle code.

Keep reminding Oracle to give more information on fixes/patches.

Oracle has helped in the conversion process be letting you access your old IMAGE data in read-only mode by using the product SQL*Connect. We have not used the product.

OTHER : Conferences/Users Groups

There are 3 main levels of conference: 1) Local area conferences, 2) Regional area conferences and 3) International Conferences. Also, there are two main types of User Groups: 1) Oracle - broad and 2) Financials.

Once a year, Oracle organizes a week-long conference, (5 days), called ' Oracles International Users Week'. These contains all subjects - from Tips to Techniques, Tuning, Case Studies, mini courses of Oracle Products, Question and Answer sessions, Round tables discussions with Oracle and/or the hardware vendors and many more. Last year alone, about 500 papers were published.

At this conference, they also have a vendor show of 3rd party software vendors and hardware vendors to showoff their speed when running Oracle as well as their compatibility, tools and aids.

This conference is a MUST for those who want to meet the people who are responsible for the RDBMS, port, support, ... and their managers. Here are the people whose names you've heard but never gotten speak with directly. Build up a rapport, but do not threaten them. Just tell them how you are concerned about some of the short comings and/or possible show stoppers. This conference generally brings people together who are running Oracle on the same hardware platform, but more important, people who are using Oracle Financials on the same hardware

platform. Oracle never gave you those names before because they did not want to be a reference at that point in time. Now the birth of the non-official support group of Oracle Financials on your hardware platform is born.

At the Regional level, the size is much smaller and has about the same topics. It is worthwhile in the sense of meeting people who are doing similar things as you are in the same general area.

At the Local level, you meet with people in your general area that are Oracle users. A good place to start to meet people and possibly run into some who have a similar situation as yourself. Much knowledge can be learned here. At times, the can get Oracle to come in an give a demo about a new major release of their software/tool. This is also true with third party software vendors.

A new addition is the Oracle Financial Users Group. The last meeting was held in Washington, DC. The thrust of this conference is Oracle Financials. This is not sponsored by Oracle, but key players from Oracle do show up.

CONCLUSION

Remember that this paper is from my point of view and the experiences that I have. I take no side in saying whether or not to go with Oracle Financials on your HP3000, or with Oracle Financials on any other hardware platform.

This is a technical information paper that you can use when talking with Oracle Corporation and with Hewlett-Packard.

Remember to ask MANY questions when dealing with the 2 vendors. Demand a list of existing HP3000 Oracle Financial users as references. There are about 7 known companies/corporations to date that are running Oracle Financials on a HP3000 hardware platform. I don't know how many are on Oracle's Reference list, but ask. Talk to these people and I would advise trying to spend some time on-site seeing the product in a production/live environment verses just seeing a demo.

ADDITIONAL Information

If you have additional technical questions, or would like copies of how to configure your devices (terminals, printers, ...) please feel free to contact me.

Included are some examples of some Reflections files: keyboard remapping files, procedures how to initially set up the special VT configuration file. Also included are: some common <escape> sequences used by the HP printers..., a keyboard layout that our financials users are using..., some examples of command files that invoke the Oracle Application.

Getting Over the Hurdles of Oracle Financials on the HP3000's Hardware Platform

ANNOUNCEMENT

Cray Research, Inc is no longer running Oracle Financials on a HP3000 960. We have switched over to a HP9000 870 machine about 4 weeks ago. The main reason for the switch is connectivity to the rest of the Cray network that contains many flavors of Unix machines. The rest of Cray will have some sort of access to Oracle Financials and we at Corporate will need to get data from other systems as well. The HP3000 does not fully support our needs!

Craig P. Albrecht
Cray Research , Inc
1440 Northland Drive
Mendota Heights, MN 55120

Direct Phone: (612) 683-7281 (cst)

Example: Reflection Keyboard file for a IBM AT type compute with an enhanced BIOS and keyboard.

```
keyboard-id  = ENHANCED

term = VT

set num-lock ON

       num-lock = vt-pf1
      kp-slash   = vt-pf2
      kp-star     = vt-pf3
       kp-minus = vt-pf4

      kp-plus    = vt-minus
shift kp-plus    =vt-comma

; -- Note: "^[" is an <escape>

f1          = "^[5"  ; pf5
f2          = "^[6"  ; pf6
f3          = "^[7"  ; pf7
f4          = "^[8"  ; pf8
f5          = "^[9"  ; pf9
f6          = "^[0"  ; pf10
f7          = "^[-"  ; pf11
f8          = "^[="  ; pf12
f9           = null
f10          = null
f11          = null
f12          = null

kp-enter  = return

cp-ins      = null
cp-del      = null
cp-home   = null
cp-end      = null
cp-pgup    = null
cp-pgdn    = null
```

Example: Reflection Keyboard file for a IBM AT type computer with an enhanced BIOS but has a standard keyboard.

```
term = VT

set num-lock on

69 hide                = vt-pf1
53 e0-prefixed hide = vt-pf2
kp-start               = vt-pf3
kp-minus               = vt-pf4

      kp-plus          = vt-minus
 shift kp-plus         = vt-comma

f1  = "^[5"  ; pf5
f2  = "^[6"  ; pf6
f3  = "^[7"  ; pf7
f4  = "^[8"  ; pf8
f5  = "^[9"  ; pf 9
f6  = "^[0"  ; pf10
f7  = "^[-"  ; pf11
f8  = "^[="  ; pf12

f9   = null
f10  = null
```

Example: Procedure to create a VT configuration file with Reflection with the remapped keyboard.

1] R1 Execute Reflections in the HP Mode
2] alt-C Configuration keys
3] f7 Global Configuration
4] f1 Next Choice <DEC>
5] f6 Save to disc
6] {new filename.CFG} for the VT Oracle emulation
7] return
8] Adjust the baud rate, proper COM port, ...
9] f6 Save to disc

10] return
11] alt-C Configuration keys
12] Make sure receive and transmit pacing is XON/XOFF, verify the baud rate and COM port.
13] f6 Save to disc
14] return
15] f3 Terminal page 1
16] Initial Label set <User>, User Label lines <0>, Backspace key <BKSP>, Keypad mode <NORMAL>, Cursor Key mode <NORMAL>, Terminal type <VT220-7>, DA response <VT100>
17] f6 Save to disc
18] return
19] f7 Optional (color configuration)
19.1] f1 activate color after you made your selection
19.2] f6 Save to disc
19.3] return
20] COMPLETED

Load the Keyboard map file into the VT Oracle configuration file.

KEYMAP {VT Oracle keyboard file} {VT Oracle Reflection configuration file}

NOTE: When configuring colors, the flowing list must be configured and each must be a unique color patterned for easy of readability and functionality:

1] Normal	2] Bold	3] Underline
4] Inverse	5] Inverse/Underline/Bold	6] Inverse/Bold
7] Underline/Bold	8] Inverse/Underline	

EXAMPLE: HP3000 command file to invoke the Reflection VT Oracle Configuration file:

```
echo "<esc>&j@"              Set the user keys but do not display them
echo "<esc>&oFLoad {VT Oracle configuration file}"
pause 1
setvar HPTYPEAHEAD TRUE
FOUND .....    {invoke Oracle Foundation with the proper CRT definition}
setvar HPTYPEAHEAD FALSE
echo "<esc>&0FLoad R1.CFG"  Load the HP terminal emulator
echo "<esc>&k01\"            Reset the terminal class from DEC to HP
```

EXAMPLE: How to toggle the HP700/92 and HP2392A terminals in and out of VT/ANSI mode.

This example, and many more will be supplied at the conference.

TITLE: "...But We Only Have COBOL! The Real Dilemma."

AUTHOR: Rafael Benitez

Martin Marietta Info. Systems Group

P.O. Box 179; M/S Al6330

Denver, CO 80201

303-790-3615

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING.

PAPER NO. 3202

3203

APPLICATION INSTALLATION

FRANNIE CASELLA
NORTHERN CALIFORNIA CANCER CENTER
1420 HARBOR BAY PARKWAY, SUITE 260
ALAMEDA, CA 94501
(415) 748-6111

OVERVIEW

This paper will address the issues and constraints in installing third party maintained software on an HP3000.

Our organization has the source code to our major application but the bulk of the maintenance of this application is carried out by another organization which owns the source code. We also install our own modifications to this source code due to our own special needs. Thus the nightmare begins.

INTRODUCTION

At one time or another we have all been a slave to two masters. If you are lucky the two want very similar things from you. If you are not so lucky you end up writing a paper like this.

First a brief background on my two masters. The first and primary master is the Federal Government; specifically the National Cancer Institute (NCI). My second one is the State of California; specifically the Department of Health Services. We hold contracts from each to periodically provide them with specific data from our Cancer Tumor Database file. Our application software CANDIS (Cancer Data Information System), which is used

to maintain the data, is owned by the State and the bulk of the maintenance is carried out by them.

We collect cancer data from all hospitals in the five San Francisco Bay Area Counties using Cansur/Net software. This software is also maintained by the State but is totally independent of the CANDIS software. The data is uploaded, edited and reported on bi-monthly by the CANDIS software through a series of COBOL programs which run in batch mode. The data is checked for accuracy and completeness and then modified through a series of View screens and three COBOL programs which run on-line.

The cancer data is then regularly sent to both the State and the Feds, but each requires a different version of the data. For this reason, we enhance the State's software to comply with the standards of the Federal Government to handle those situations when the requirements for the Feds can be easily incorporated into the State's software. Confused? Join the club!!

The standard we follow is that the database holds the data according to the State's specifications. Any modification which is necessary for the Feds but not the State is then handled in a separate program. This program runs on the data after it has been extracted from the database. Any recodes to data or any manipulation which needs to be made to conform to the Federal regulations is done to the extract file only. This way the database stays in sync with the State and the Feds receive their data in the format they stipulate. This keeps both masters happy.

BACKGROUND

After the State installed the first version of CANDIS, we made the modifications necessary to meet the Federal requirements that wouldn't interfere with the State's specifications. The big fiasco didn't happen until the second version release came from the State. After the

installation we found that several of the modifications we had made for the Fed's were gone. After making all of these changes AGAIN, we realized that it was time to sit down and plan our strategy before the next release. Neither the programming staff nor the users wanted to ever go through another few weeks like we had after that last installation.

The following is a list of issues which we put together as a checklist of sorts to make sure that the next installation went quickly, smoothly and, most important, that nothing was forgotten.

- receiving the software
- the accounting structure
- comparing the new release with the old
- modifications
- keeping track
- testing
- installing

Receiving the Software

This seems like a somewhat trivial point to bring up but what it includes is the fact that you should be prepared for the software prior to physically receiving the release.

Try to get an exact date of the release if this is at all possible. Many times it is not. Try to pin down the date to something a little more precise than "spring". At least go for a specific month and once you have that you can usually determine if it will be early in the month or more towards the end. This is important for your initial planning to determine when the software will be available to the end users once the release is in-house.

One must remember that the end users will be impacted by this version release also. If there are major changes perhaps training will have to be arranged. In our shop,

we are on a tight production schedule. We upload approximately 1500 to 2000 new records every two weeks. It is imperative that we keep on this schedule to insure that we stay current and don't let the cases get back-logged. What this means is that we must coordinate the installation with the production calendar. We try to minimize the user down time which usually means working on the weekends.

If you take some time prior to receiving the release to prepare for its installation you can save yourself time and energy. I know this is a new concept for some of you, it was for us. We call it Planning Ahead!!

If you have a large programming staff take the time early to pick the lucky soul(s) who will be doing the install. You might even want to let them know ahead of time so that they can be mentally prepared for it. If your staff is small and the person knows who he or she is, you might want to just sneak up on them, smile politely, whisper the install date and then run. This gives them time to finish current projects and adjust their workload accordingly.

An important and somewhat time-consuming task is creating an "install" environment. This should be created as similar to the production environment as possible. If room permits on the system one can just store the production account and restore it into a test account. If you are like most of us, space on your system is a bit tight. What we did was to create two test databases.

The first is very small, approximately 500 records. It is used for all the initial testing. It is by no means a perfect test database. It does allow the programmers to quickly test out their programs to see if they run to completion. It also lets them work out any bugs in their JCL, e.g. missing file statements, incorrect file names, etc. It is strictly used as a primary check before any thorough testing begins.

We then created a scaled-down version of the database as close to production as possible. A product such as

SUPRTOOL by Robelle is great for this. It allows you to select every "nth" record from a dataset to be copied for loading into a test database.

This can all be done ahead of time so that when the software does arrive it can be immediately put onto the system and nobody has to wonder "now where did I put that tape?".

THE ACCOUNTING STRUCTURE

As mentioned in the previous section it is very important to keep your test environment as close to your production environment as possible. Of course we all know this already. What is also important though is keeping the current software totally separate from the new version release. This can get quite confusing if you aren't careful. Don't mix any new and old software until the actual release. What we did was to set up three groups in the test account. We call them SRCOLD, SRCNEW and SOURCE.

SRCOLD holds all of the current source code; SRCNEW is where the new release is put; and then as each program is reviewed and the necessary modifications made, that source code goes into SOURCE and will become the current version of our software.

The same holds true for the JCL. When receiving updates for running programs in batch mode, we also set up groups to hold these files. We use JOBOLD, JOBNEW and JOBS. The same logic applies as above for the source code. Once the accounting structure is set up and the new release downloaded onto the HP3000 you are ready to start looking at just what it is that you have been sent.

An often overlooked detail is making sure that the install environment has the identical CAPABILITIES and ACCESS as the production environment. You want to ensure that when the application is moved into production that it runs without a hitch.

COMPARING THE NEW RELEASE WITH THE OLD

I don't know about most of you out there but when I install a new version of software I like to know exactly what changes have been made to it before the users come running down the hall screaming "did you know that this program now does this?". This way I can always say "yes, of course I do -- I installed it".

More importantly, by looking at the modifications that have been added to the code before you install them for the users, you have a chance to discuss them with the users and determine whether you in fact should install the change. It really does pay in the long run to take the time and discuss modifications with the people who have the authority to decide whether the application would be better off with or without the new change. This not only makes it easier on the staff who then only has to install it once, but it also makes the users feel that they have some input into the application they are using.

Hopefully, your release comes with some kind of documentation as to which programs have been modified and which have not. This was not always the case for us. Eventually we did start getting some documentation but we had already come up with a way of determining the changes ourselves. We made a list of all programs and JCL in the system. We then compared each of the programs against the version which was currently running. As we reviewed each piece, we checked it off as having been looked at.

A great product for comparing different versions of source code is SCOMPARE by the ALDON Computer Group. What this software does is to compare two versions of the same program and detail every line of code which has been modified. The output is very easy to comprehend. You even have the option of creating an editfile. This file holds all the line numbers for which should be added and

which should be deleted to make the old version match up with the new version. If you decide that the new version is not quite what you want to install, SCOMPARE allows you to pick and choose which new lines of code you do want and easily discard the lines you don't. I don't know of any other software on the HP3000 that does anything similar to this, but I can say that if you need to compare modules or entire programs SCOMPARE can easily save you hours of work.

One standard we follow here is that those changes we receive and do not wish to install are left in the program and just commented out. This way the code stays in the program as self-documentation and is available if the users change their minds in the future.

At this point we are not only looking at the code that was added but looking for code that may not have been added. What I mean here is that if the documentation states that code has been added to perform a specific function, I look to see that somewhere there is code to do that. This also goes for bug fixes. Has the code been modified that is going to take care of all the bugs listed as "fixed" in the documentation? Has any code been removed which shouldn't have been?

I check off all these types of things on my checklist as I go along. This list then becomes part of the documentation of the installation.

MODIFICATIONS

Once the modifications have been identified and everyone has agreed on which ones to install, the process is very simple. We move the newly modified program into the SOURCE group. At this point I like to run another SCOMPARE just to make sure that all the new changes are accounted for and in the right spot in the program.

SCOMPARE will also work on copylibs, so don't forget to take a look at them. It can do the entire copylib or just a member or two.

Once everything has been verified I run my compile. If all goes well, I mark off that program and move onto the next until they're all done.

KEEPING TRACK

Keeping track of which programs have been revised can be as simple as the checklist I mentioned previously. I make a list of all programs which need to be modified and after they are looked at I check them off, then when the changes have been made, another checkmark. The same goes for a clean compile and the final move into the SOURCE group. This checklist can also be used for keeping track of which have been tested later on. This is also how I remember which programs I have made modifications to which were not part of the install from the State. This way I am sure to include them in the new release.

This may seem unnecessary, but if you work in an environment like mine where I am constantly being interrupted, this simple checklist can save your sanity. To tell you the truth, even without interruptions from users I sometimes find it hard to remember what I did yesterday, so you can see why this is a must for me!

TESTING

I am not going to go into the fine art of testing here but I do want to stress the importance of testing each of the programs which have been modified. Even the smallest change to a piece of code can make HUGE errors, as I'm sure you are all aware. We once had a programmer working for us whose answer to the question "did you test it?" was "well, just look at the code - it's only one line". Not a good answer!!! Even the one-line changes need to be fully tested.

As I mentioned earlier in this paper and I'm sure all you programmers out there already know, it is imperative that your testing environment be as close to the real thing as possible. Many bugs that slip through are because the testing environment does not accurately represent what is out there on the live database. Also remember that some things that need to be tested are codes not found in the live database but which are tested for in the programs. You need records that are old as well as more recent. You need some with multiple entries (if this is possible for you) and those that are single. You need a wide variety of codes in different fields so that all possibilities are checked for each new piece of code.

If the record does not exist on the test database which will thoroughly test a piece of code, I use DBEDIT from Robelle to create the record or I use Query to modify one that will force it to fit my specifications. Just remember that taking the time to test thoroughly up front prevents a lot of headaches later on down the road.

After each of the modified pieces of code has been tested I then test it again within the context of the entire program. I select certain records on the test database which I follow through the program. I then check the end results to be sure they match my expected results. When this has been done for all programs I then run a complete application test to verify that all the programs work together the way they were intended.

Testing is very time consuming. I don't let this portion be hurried. I make sure that when we are targeting an installation date that I have given myself plenty of time for testing. If things don't go well when testing, we delay the install date rather than put untested code into production.

INSTALLING

The actual installation of the new software should now be a piece of cake. All of the new code is together in

one group, it has all been tested and now it needs to be moved into the production environment. This can be as simple as storing your executable code from the group in the test account and restoring it into the production group. The same can be done for the source code (if it is to be kept in the production account). We have a "library" account which holds all the new source code as well as the previous version as a backup. This guarantees that if revisions need to be made at a later date, the programmer knows where the current version resides. There should never be any doubt as to which program is the version to be modified!!! After the source code is in the "library" account (or where ever it is kept on your system) it is a very good idea to back it all up onto tape. Then all of the source should be removed from the test environment. This removes the possibility of someone assuming that the code in "test" is the current version and using it when making changes in the future. That is a very easy way to get your software out of sync.

For those programs which will be running in batch mode, be careful that the job card has been modified to point to the production account. Another gotcha here is to be sure that the output files for production are built to the proper size. It's very easy to forget to increase their capacities after they have been built for testing in the test account.

Again, be sure that the users are aware when the installation is to occur. If it is to happen over the weekend make certain that people who occasionally or regularly work on the weekend know that the install is taking place and that the production account will be inaccessible. We have made it a policy that when we are installing new software in the production account that the entire system is off limits to everyone. This saves time in trying to figure out who needs to know and who doesn't. Everyone needs to know and everyone needs to stay off!

CONCLUSION

Congratulations, the new application has been installed and everything is done, right? Wrong! At this point the users are using the new version but everyone should be alert to the fact that things could still go wrong.

We watch our first few live runs very closely. I review all the output to be sure counts look accurate, that the data itself looks correct on all the different reports. Are the codes that are being printed out valid values for each field? This is the type of error that is often overlooked because people assume that if there is data printed out on the report, that the data is correct. This is not always the case.

Users should be extra aware of any discrepancies in the new release. Whether it is a strange looking code, a View screen that doesn't look quite right anymore, or a report that perhaps doesn't total correctly. They should all be on the lookout for anything out of the ordinary. Even if the user turns out to be wrong about a possible bug, it is better that the problem be looked into as early as possible. We really encourage our users to come to us any time they feel that something is out of whack.

If problems do come up or bugs are found in the software we have found that notifying the State immediately was to our advantage. In the beginning we attempted to fix them ourselves and then to notify them. Sometimes the fix we made was not written the same way as their fix and then we were out of sync, AGAIN.

We also realized that when we did call about a potential problem, we had to make sure that we did not call and start screaming. Funny how that immediately puts people on the defensive. It's far better to call and ask them to look into the problem or perhaps offer a possible solution (if asked). Keep the lines of communication open -- you may need them for something in the future!

A few final thoughts are:

- the importance of planning ahead
- the importance of not being rushed
- paying attention to the detail work
- carefully testing all programs
- coordinating the install with the users

Paper #: 3204

DECISION TABLES – MAKING THE COMPLEX SIMPLE

George Federman
George Federman & Associates
6236 Parkhurst Drive
Goleta, CA 93117
(805) 683-3037

Decision tables are overpraised and underutilized. Every text on systems analysis states their virtues. They can clarify complex user specifications. They can aid the analyst in specifying logic to a programmer. When complete, they can remain a part of user documentation, leaving a window into otherwise hidden and intricate logic. More succinct than pseudocode, they offer checks for completeness and consistency that pseudocode can't provide. Yet despite all this capability, this once-popular tool is now relatively unused.

I'd like to review some of the fundamentals of decision tables in an attempt to once again present the power of the tool.

Let's begin with a simple example of translating a company policy into a decision table. We're going to interview the CEO of Mind-Trippers, a data processing consulting firm with a strong '60's orientation. In our interview, Bat D. Fledermaus, the CEO, says:

"We dig our free time, man, and like to plan some groovy things to do when we have time left over from work. But if a project's overdue, we put our bods in gear and go to work. If it's a normal workday, well, work's a grind, but we gotta do it, so it's off to work. Sometimes we have research to do, and then we truck on down to the university library. That's it, man, a simple corporate philosophy and how we earn our daily bread."

Having interviewed the CEO, we can put his policy into a decision table, using a technique borrowed from T. R. Gildersleeve (32-40). First, we isolate all the conditions. Underlining these:

"We dig our free time, man, and like to plan some groovy things to do when we have time left over from work. But if a project's overdue,

we put our bods in gear and go to work. If it's a normal workday, well, work's a grind, but we gotta do it, so it's off to work. Sometimes we have research to do, and then we truck on down to the university library. That's it, man, a simple corporate philosophy and how we earn our daily bread."

Having isolated the conditions, we'll isolate the actions:

"We dig our free time, man, and like to plan some groovy things to do when we have time left over from work. But if a project's overdue, we put our bods in gear and go to work. If it's a normal workday, well, work's a grind, but we gotta do it, so it's off to work. Sometimes we have research to do, and then we truck on down to the university library. That's it, man, a simple corporate philosophy and how we earn our daily bread."

Gildersleeve then recommends that we standardize the language, and if any duplicate conditions or actions exist (like "go to work" when we have an overdue project or on a workday), remove the duplicates (41-44). Standardizing and removing duplicates, we have:

Conditions:

1. Time left over from work?
2. Overdue project?
3. Normal workday?
4. Need to do research?

Actions:

1. Go to work.
2. Go to the library.
3. Plan leisure activities.

Following Gildersleeve's guidelines, we'll eliminate negative conditions (45). These already exist in another form, and just rephrase one or more of the already existing conditions in the opposite way. Thus "time left over from work" can be removed as a condition, since it is implied when the other three conditions have "N" (false) values.

Gildersleeve then lists several other steps to take (listing actions in execution order, including an else rule, eliminating redundancy and contradiction, optimizing searching, and checking for completeness)

(106). We'll deal with some of these as we construct and review our initial decision table.

Defining some terms first, a decision table consists of condition stubs, condition entries, action stubs, and action entries. Any one row shows a condition and its entries or an action and its entries. Any rule column shows what actions should be taken given certain values for the various conditions. Thus, in Figure 1 below, the conditions are in the upper left quadrant of the table (overdue project, normal workday, etc.) The condition entries are in the upper right quadrant. (These are Y/N entries, since each condition is phrased as a Y/N or true/false question. The entry can also be a minus sign or hyphen, which serves as the indifference symbol or "don't care" symbol. If a condition is irrelevant, and could be Y or N without affecting the resulting action, we use a "don't care" symbol.)

The actions appear in the lower left quadrant, and the action entries are marked with an X in the lower right quadrant. (If we wanted the actions to be executed in a specific sequence, we could use 1, 2, 3, etc. as action entries instead of an X. However, if the order in which the action stubs appear is the order in which we want them performed, an X is sufficient.)

Table name or process identification	Rules		
DETERMINE DAILY SCHEDULE	1	2	3
Overdue project?	Y	–	–
Normal workday?	–	Y	–
Need to do research?	–	–	Y
Go to work	X	X	
Go to the library			X
Plan leisure activities			

Figure 1. Determine daily schedule (incorrect rules).

Our first decision table is remarkably simple, and simply wrong. Rule 1 says that if there is an overdue project, company policy is to have employees go to work. Rule 2 says that if it is a normal workday, the policy is to go to work. Finally, Rule 3 says that if research has to be done, company policy is to send employees to the library. Each rule has "don't cares" in it, indicating that the other conditions are irrelevant in determining the action.

Intuitively, we know the table is wrong. It looks like employees will never "plan leisure activities," yet this is an action listed on the table and one we know should be invoked if they have free time. We can also see that if it is a normal workday but employees need to do

research, we have contradictory rules, and don't know whether they go to work or go to the library. We could try to figure out the inconsistencies on our own, but one of the virtues of decision tables is the ability to mechanically arrive at a complete, consistent table.

Let's do some of the simple mechanics. First of all, we have three conditions, each of which can take on a Y or N value. In theory, we should have 2x2x2 simple rules, or 8 altogether. (If we itemized these, we would expect to see YYY, YYN, YNY, YNN, NYY, NYN, NNY, and NNN, reading down each rule column.)

But our table has three mixed or complex rules, containing indifference symbols. How many simple rules are represented in the complex ones?

Since each indifference symbol represents the acceptability of both a Y and a N (since we don't care what value that condition has), each symbol represents two alternatives. Looking at Rule 1 (Y– –), the Y is one value, the first dash represents two possible values, and the second dash represents two more possible values. In short, we have 1x2x2 simple rules combined into one complex rule in Rule 1. Following the same reasoning, we have 2x1x2 simple rules represented in complex Rule 2, and 2x2x1 simple rules in Rule 3. Technically speaking, we have computed "column counts" and discovered that each rule (column) has a count of four. If we sum all three column counts, we find that we have 12 simple rules embedded in our table, when we only expected eight.

Let's expand the complex rules into simple ones, not only itemizing the 12, but finding out where we deviated from the expected count. Taking Rule 1 first:

				1	a	b	c	d
Y	becomes	Y Y	and then		Y	Y	Y	Y
–		Y N			Y	Y	N	N
–		– –			Y	N	Y	N

by expanding each don't care in turn into a Y value and an N value.

Doing the same for Rule 2:

				2	a	b	c	d
–	becomes	Y N	and then		Y	Y	N	N
Y		Y Y			Y	Y	Y	Y
–		– –			Y	N	Y	N

Expanding each "don't care" in Rule 3:

					3	a	b	c	d
–	becomes	–	–	and then		Y	N	Y	N
–		Y	N			Y	Y	N	N
Y		Y	Y			Y	Y	Y	Y

If we compare these 12 simple rules, we immediately detect problems. Rule 1a and 2a are identical: they have the same condition entries (YYY) and the same action. Therefore rule 2a is redundant and can be eliminated.

Rule 1a and 3a have the same condition entries (YYY), but lead to different actions ("go to work" versus "go to the library"). We have a contradiction, and have to go back to the user to determine what action to take. (Let's assume Bat D. Fledermaus says that work is more important than anything else, and if there is a conflict, his personnel go to work. In that case, we would eliminate rule 3a.)

Continuing the comparison, we find that 1b and 2b are identical, so 2b is redundant and can be eliminated. Rules 1c and 3c are contradictions, as are 2c and 3b, and following the CEO's emphais on work, we eliminate 3c and 3b.

If we eliminate 2a, 2b, 3a, 3b, and 3c, we now have seven rules left from our original 12. (See Figure 2.) We expected eight. What's missing?

Table name or process identification	Rules						
DETERMINE DAILY SCHEDULE	1	2	3	4	5	6	7
Overdue project?	Y	Y	Y	Y	N	N	N
Normal workday?	Y	Y	N	N	Y	Y	N
Need to do research?	Y	N	Y	N	Y	N	Y
Go to work	X	X	X	X	X	X	
Go to the library							X
Plan leisure activities							

Figure 2. Determine daily schedule (missing rule).

All work and no play makes Fledermaus a dull boy. Unless we add an eighth rule, NNN, there will never be any free time, and no one will ever plan any leisure activities. We add the missing rule, and finally get the complete table in Figure 3.

The table is easy to read, and reflects the policy described above. Looking at Rules 1 through 4, if there is an overdue project, then

workday or weekend, personnel go to work. Rules 5 and 6 indicate that if there are no overdue projects, but it's a normal workday, it's off to work. On weekends, if research is needed, it's off to the university library (Rule 7). Finally, assuming no critical projects, no workday, and no research to do, it's time to plan some leisure activities (Rule 8).

Table name or process identification	Rules							
DETERMINE DAILY SCHEDULE	1	2	3	4	5	6	7	8
Overdue project?	Y	Y	Y	Y	N	N	N	N
Normal workday?	Y	Y	N	N	Y	Y	N	N
Need to do research?	Y	N	Y	N	Y	N	Y	N
Go to work	X	X	X	X	X	X		
Go to the library							X	
Plan leisure activities								X

Figure 3. Determine daily schedule (complete).

A table is correct and complete if two conditions are met. The first is that the sum of the column counts matches the expected simple rule count. Here, each column has a column count of one (YYY is evaluated as 1x1x1, YYN is 1x1x1, etc.) and the sum of the counts is eight. The expected count is 2x2x2, the product of the possible outcomes for each condition, and the two numbers are the same.

The second condition is that each simple rule is unique. One technique for verifying uniqueness is to take column 1 and compare it to columns 2 through 8, looking for duplicates. If there are none, then compare column 2 with columns 3 through 8, then column 3 with columns 4 through 8, etc. In our table, each column is unique. Thus the table is indeed complete and correct.

At this point, we can consolidate simple rules into complex ones to simplify the table. If a rule differs from another in just one value for a single condition, and the rules have the identical actions, the two rules can be combined and an indifference symbol substituted for that specific value. Thus we could combine Rule 1 (YYY) and Rule 2 (YYN) into (YY–), Rule 3 (YNY) and Rule 4 (YNN) into YN–, and the two new columns (YY– and YN–) into a single rule (Y– –). Similarly, we can combine Rule 5 (NYY) and Rule 6 (NYN) into a single rule NY–. Can we combine Rule 7 and Rule 8? No, because Rule 7 has one action ("go to the library") while Rule 8 has another.

Our final table appears in Figure 4.

Table name or process identification	Rules			
DETERMINE DAILY SCHEDULE	1	2	3	4
Overdue project?	Y	N	N	N
Normal workday?	–	Y	N	N
Need to do research?	–	–	Y	N
Go to work	X	X		
Go to the library			X	
Plan leisure activities				X

Figure 4. Determine daily schedule (final).

This table is a limited entry decision table (LEDT). The way the conditions are phrased, we are limited to two values in the condition entries (Y or N). Similarly, the action entries are a number (1, 2, 3, etc.) or an X, but really limit us to two choices (perform the action or don't). Other types of tables can be created, such as an extended entry decision table (EEDT), where the condition and action entries "extend" the wording in the stubs and can take on more than two values. We could also have a mixed entry decision table (MEDT), a blend of limited and extended entries in one table. (Examples of an EEDT and an MEDT appear below.)

If we wanted to code from this table, it naturally divides into an if..else coding structure. If we bifurcate the table at condition 1 ("overdue project?"), drawing a line down to separate the Y from the N rules, and another line between condition 1 and condition 2, we have the first "if" clause in an if..else structure. (See Figure 5.)

```
If overdue project
     go to work
else ...
```

If we do the same with condition 2, we have the second clause:

```
If overdue project
     go to work
else if normal workday
          go to work
     else ...
```

Finally, doing the same for condition three:

```
If overdue project
     go to work
else if normal workday
          go to work
```

```
else  if need to do research
            go to the library
      else
            plan leisure activities.
```

Table name or process identification DETERMINE DAILY SCHEDULE	Rules 1	2	3	4
Overdue project?	Y	N	N	N
Normal workday?	–	Y	N	N
Need to do research?	–	–	Y	N
Go to work	X	X		
Go to the library			X	
Plan leisure activities				X

Figure 5. Determine daily schedule (bifurcated).

A mechanically-perfect LEDT should still be reviewed. Let's assume we are comparing two numbers, A and B, and want to print a message indicating which is greater or if they are equal. We have three conditions (A>B, A<B, or A=B) and could create the table in Figure 6.

Table name or process identification COMPARE A TO B	Rules 1	2	3	4
A>B?	Y	N	N	N
A<B?	–	Y	N	N
A=B?	–	–	Y	N
Print "A is greater than B"	X			
Print "A is less than B"		X		
Print "A equals B"			X	

Figure 6. Comparison table (LEDT).

Reading the table, it seems to make sense. The table has three conditions, and we expect eight simple rules. We indeed have eight simple rules. Each rule has to be unique, and each is. We know that we have no redundant, contradictory, or missing rules. Thus we have a mechanically-perfect table.

We also have a small problem. Rule 4 (NNN) makes no sense. It is impossible. Similarly, Rule 1 contains four simple rules, but if we expanded the rule Y– – into YYY, YYN, YNY, and YNN, we would find that only YNN makes sense. The remainder are also impossible. Similarly, expanding Rule 2 (NY–), NYY is impossible and only NYN makes sense. Out of our original eight rules in our mechanically-

perfect table, only three make sense, and five are impossible. To be meaningful, our table should be reduced to the one in Figure 7.

Table name or process identification COMPARE A TO B	Rules 1	 2	 3
A>B?	Y	N	N
A<B?	N	Y	N
A=B?	N	N	Y
Print "A is greater than B"	X		
Print "A is less than B"		X	
Print "A equals B"			X

Figure 7. Comparison table (LEDT).

We are now logically correct, but have three simple rules where we expected eight. We could explain the difference by using new notation within the condition entries (like an asterisk or N! to indicate an N by implication) or by additional notes. As an alternative, we could create the extended entry decision table (EEDT) in Figure 8.

Table name COMPARE A TO B	Rules 1	 2	 3
A is	>B	<B	=B
Print "A is "	"greater than B"	"less than B"	"equal to B"

Figure 8. Comparison table (as EEDT).

We could also have blended the limited entry format with the extended entry format to get the mixed entry decision table (MEDT) in Figure 9.

Table name COMPARE A TO B	Rules 1	 2	 3
A is	>B	<B	=B
Print "A is greater than B"	X		
Print "A is less than B"		X	
Print "A equals B"			X

Figure 9. Comparison table (as MEDT).

Now we have both a mechanically-perfect table and a logically-sound one free of impossible rules.

Returning to the idea of passing a table to a programmer as a specification, an EEDT or MEDT immediately translates into a case structure, with case structures or if..else structures nested within it.

There are further advantages from a programmer's point of view. Tables can be factored and nested, and both condition checks and actions can invoke subtables.

As an example, a condition like "overdue project?" will have a condition entry (Y or N) that could be returned as a value from another table. That subtable could later become code which links an employee to each of his projects, compares each project due date with the current date, and returns a Y if any due date exceeds the current date or an N otherwise.

Action stubs could invoke subtables. "Plan leisure activities" could be a call to another table, and ultimately be coded as a PERFORM or a called subroutine.

We can create manager tables and worker tables. We can have condition-only tables and action-only tables, factoring out complex condition checks or actions so that the larger parent table (or manager table) is simpler to read and follow. Tables can be divided and nested like hierarchy charts or leveled data flow diagrams.

As an added benefit, all the constructs of structured programming (sequence, selection, and iteration) are available in decision tables. This one-to-one correspondence could greatly enhance analyst-programmer communication and result in the writing of much clearer specifications.

We've seen the benefits of tables for clarifying policies and procedures and for creating clear programmer specifications. The same clarity is available when tables remain part of user documentation. In a concise, graphic form, tables provide a macro view of all the conditions and all the actions that pertain to a policy or procedure. Moreover, no matter how complex the logic in the table, the user can trace that logic just by following the columns. This standardized way of reading logic frees the user from having to learn different styles of "structured English" and pseudocode. Finally, if user specifications to the analyst consist of tables, and those tables (as is or amplified) become analyst specifications to the programmer, we now have a common form of communication among all three, and a common window into the final system.

Bibliography

CODASYL. A Modern Appraisal of Decision Tables. New York: Association for Computing Machinery, 1982.

Gildersleeve, Thomas R. Decision Tables and Their Practical Application in Data Processing. Englewood Cliffs, N.J.: Prentice-Hall, 1970.

Humby, Edward. Programs from Decision Tables. London: Macdonald, 1973.

Hurley, Richard B. Decision Tables in Software Engineering. New York: Van Nostrand Reinhold, 1983.

London, Keith R. Decision Tables. Princeton, N.J.: Auerbach, 1972.

Metzner, John R., and Bruce H. Barnes. Decision Table Languages and Systems. New York: Academic Press, 1977.

Montalbano, Michael. Decision Tables. Chicago: Science Research Associates, 1974.

Pollack, Solomon L., Harry T. Hicks, Jr., and William J. Harrison. Decision Tables: Theory and Practice. New York: John Wiley & Sons, 1971.

Paper # 3205

DYNAMIC MENU SYSTEMS FOR THE COGNOS PRODUCT

Gene Harmon
AH Custom Software, Inc.
8210 Terrace Drive
El Cerrito, CA 94530
(415) 525-5070

Genesis/Goals

As consultants for a variety of clients, we attempt to design systems which provide the maximum functionality for our client's needs, and which also provide the maximum benefit from the use of a fourth generation language, such as the Cognos product. However, more often than not, many of our best methodologies derive from specific requests from our clients. One of these requests is the subject of my paper today: a dynamic menu system which includes within it a security and printer control system.

The normal setup of an hierarchical menu system which we had designed for many of our clients, meets most requirements of clarity and functionality; however, it requires the user to navigate up and down long chains of menu selections. These navigations can be tedious, lengthy, and confusing.

Goals of the the new system for our client were the following:

1. Allow the user to bring up the screen of their choice, with NO menu navigation; instead allow:

   a. Entry of NEXT;xxxx in the ACTION field to allow the call of the new xxxx screen.

      or

   b. Entry of MENU in the ACTION field to present the user with a screen of allowable menu choices.

2. Establish security at the application program level; also allow definition of users independent of system user signon.

3. Allow assignment of system logical printers to user CRT's. For example, this will allow a requested report or completed form to print out on a laser printer which is located near the CRT on which the requesting user is signed on.

Methodology

To accomplish the goals listed above, it is necessary to create the file structure necessary to support it, and certain 'use' files. The required files are (also see file layouts at end of paper) :

(note: I have not bothered to define Automatic Masters)

1. TBL-FN-MODULE. (Table of Functional Modules)

   Field Definitions:

   1. FUNCTION.
      This is a menu code which will appear on the user's menu and is used as a mnemonic to call the desired function.

   2. MODULE.
      This is the fully defined module name (filename.group.account) which will be executed when the FUNCTION is called.

   3. DESC-MODULE.
      This is the description which will be displayed on the user's menu screen.

   4. FLAG-MODULE.
      This is a field which is not necessary for the generation of the menu system, but which can be used in conjunction with the field FLAG-LEVEL to establish security by function within an application program.

   5. FLAG-LEVEL.
      This is the field which establishes the level of security, and is compared to the level of the user's security when menu functions are assigned to the user.

6. USER-LEVEL.
   This field may be used to discrimate between classes of users; for example: Home-office, Branch, Systems.

7. NEXT-MODULE.
   (obsolete)

2. TBL-PASSWORD (Table of Passwords by user)

   Field Definitions:

   1. EMP-ID.
      This is the sign-on used; usually a first name.

   2. PASSWORD.
      This is the password assigned to the user which must be entered at the first sign-on menu for the user.

   3. NAME.
      The complete name of the user.

   4. HOME-AGENCY.
      Optionally defined field which the user must supply. Will become part of a parameter list to be passed to each called function.

   5. HOME-FACILITY.
      (see HOME-AGENCY)

   6. FLAG-SECURITY.
      This field will be used to determine if the user is allowed to have a particular function of part of his menu.

   7. FLAG-EXIT.
      This will be used to see if the user is entitled to return to the system prompt.

   8. USER-LEVEL.
      This is usually used to assign the user a level of system, home-office, or branch (or some corollary of this structure).

9. FIRST-PROGRAM.
   This is used to designate the first function which will be automatically executed after successful user sign-on.

3. TBL-PW-MODULE. (Table of Modules which have been established for the menus of each user)

   Field Definitions:

   1. PW-MODULE-KEY.
      This key is constructed for each function that is set up for the user.

   2. EMP-ID.
      This is the sign-on ID used in the initial sign-on screen.

   3. FLAG-LEVEL (See above)

   4. SEQ-FUNCTION.
      If used, this will control the sequence in which the menu functions will be displayed to the user.

4. TBL-PRINTERS. (Table of system printers)

   Field Definitions:

   1. PRINTER-ID.
      Use as the identifier which can be referenced when assigning this printer to a specific CRT.

   2. LOGICAL-DEV-NO.
      Use the system logical device number of the physical printer.

   3. DESCRIPTION.
      Use this field to specify where the printer is located.

   4. PRINTER-TYPE.
      Indicate the type of printer (laser, dot-matrix, etc.)

   5. SPOOL-SLAVE.
      Indicate if the printer is slave or spool (sp/sl)

5. TBL-CRTS. (Table of CRT's using the menu system; needed if printers are to be assigned to them)

Field Definitions:

1. CRT-MOD-CLN-KEY.
   Contains the system device-number, the menu function code, and the facility (or corollary designator).

2. PRINTER-ID.
   Contains the PRINTER-ID mnemonic for the printer set up in the file TBL-PRINTERS.

3. TRAY-NO.
   Contains the tray number of the printer (if laser).

Use Files: (see listings at end of paper)

1. NEXT.USE.

   This file must be used in the procedure section of each QUICK program which is on the user menu. Its function is to allow the user to either directly call another screen, or to pull up his own menu screen.

2. TEMP.USE

   This file contains File and temporary statements used by NEXT.USE.

3. STDHDRQK.USE.

   This file contains standard screen header information and is used in each QUICK program which is on the menu system. It is also required by NEXT.USE.

SCREEN SYSTEM:

The preceding files are used by the following QUICK screens (which comprise the MENU system):

(note: see printouts of these screens at the end of the paper)

1. System main menu(PCHMENU).

   Functions:

   1. Prompt for user-id and password and validate on TBL-PASSWORD file.

   2. Determine home agency/facility of signee; possibly allow sign-on to facility other than home; initialize parameter-list to pass on to selected functions (allowing these functions to qualify/select records on screen-files.

   3. Allow for direct call of any allowed (depending on security) function, or a call of screen displaying all allowed functions.

2. Custom User Menu (ALLNEXT).

   Functions:

   1. Display the list of menu functions set up for the user who has signed on.

   2. Allow selection and execution of a function.

3. Maintenance of Menu Functions (TFNMODU).

   Functions:

   1. Maintain all functions which can be executed by a user of the menu system, and assign security levels.

4. Assignment of Users, User-security, and Function-assignmt (TPASSWDU).

   Functions:

   1. Maintain roster of menu users and their security levels.

   2. Allow designation of function to be execution at initial signon

   3. Maintain list of functions which may be executed by user.

5. Printer Maintenance (TPRINTU).

   Functions:

   1. Maintain table of printers which can be assigned to CRT's.

6. CRT Maintenance (TCRTSU).

   Functions:

   1. Maintain table of CRT's by device number and function.

PRACTICAL CONSIDERATIONS:

At the end of this paper, you will find the QUICK code for the programs which are absolutely critical to the correct implementation of this system. Although the menu system presented in this paper will run satifactorily on either a classic or Spectrum machine, the classic version will be subject to stack overflow if there is too much reciprocal calling between functions.

This system also includes code which takes advantage of the DISC Omnidex product; this code is used solely for sorted presentation of the function codes, and may be safely removed without affecting the functionality of the menu system. You will notice that the code for program = TPASSWDU, is set up exactly as an application QUICK execution module MUST be set up (in other words, it has a standard parameter list, and it utilizes the use files: temp.use, stdhdrqk.use, and next.use).

Our experience has been that most users like this method of menu presentation; they quickly learn the 4-character mnemonic function codes, and rarely return to the main menu. From a system administrator's point of view, the system offers both the advantages of security, and the capability to offer the users just the menu functions required by them to do their job; all within the same application.

```
/t
STDHDRQK.USE,UNN
/l all
    1      USE FUNCTION.USE NOL
    2      DRAW THICK 3,1 TO 3,80
    3      ALIGN (,,30) (,,60)
    4      TITLE "|" AT ,25
    5      FIELD PASS-FACILITY-NM DISPLAY REFRESH PREDISPLAY
    6      FIELD PASS-EMP-NM DISPLAY REFRESH PREDISPLAY
    7      TITLE "|" AT ,59
    8      SKIP
    9      ALIGN (,,21) (,,30) (,,60) (,,71)
   10      ;FIELD FUNCTION-ID DISPLAY REFRESH PREDISPLAY
   11      FIELD NEXT-FUNCTION NOENTRY NOID UPSHIFT ;LABEL "Next"
   12      TITLE "|" AT ,25
   13      FIELD FUNCTION-NM DISPLAY REFRESH PREDISPLAY
   14      FIELD SYSTEM-DATE DISPLAY REFRESH PREDISPLAY
   15      FIELD FUNCTION-ID DISPLAY REFRESH PREDISPLAY
   16      ;FIELD SYSTEM-TIME DISPLAY REFRESH PREDISPLAY  PIC "^^:^^:^^"
   17      TITLE "|" AT ,59
   18      SKIP
   19      USE HILITERQ.USE NOL
/t
NEXT.USE,UNN
/l all
    1      ;USE SOMUSE8.USE NOL
    2      PROCEDURE INITIALIZE
    3      BEGIN
    4
    5      LET THIS-FUNCTION = NEXT-FUNCTION
    6      LET NEXT-FUNCTION = " "
    7      DISPLAY FUNCTION-NM
    8      ;DO INTERNAL SETKEYS
    9
   10      END
   11
   12      PROCEDURE INPUT NEXT-FUNCTION
   13      BEGIN
   14        IF FIELDTEXT = " "
   15            THEN
   16            LET FIELDTEXT = "<"
   17      END
   18
   19      PROCEDURE INTERNAL NEXT-INT
   20      BEGIN
   21
   22      IF NEXT-FUNCTION = "<" ;OR NEXT-FUNCTION = " "
   23        THEN BEGIN
   24          LET NEXT-FUNCTION = " "
   25          RUN SCREEN ALLNEXT &
   26            MODE F PASSING &
   27              PASS-EMP-ID, &
   28              PASS-FACILITY, &
   29              PASS-AGENCY, &
   30              PASS-FACILITY-NM, &
   31              PASS-EMP-NM, &
   32              NEXT-FUNCTION, &
   33              FUNCTION-ID, &                    DYNAMIC MENU SYSTEMS
   34              FUNCTION-NM, &                              3205-8
   35              PARAMETER1, &
```

```
36              PARAMETER2, &
37              PARAMETER3, &
38              PARAMETER4, PARAMETER5
39        GET TBL-FN-MODULE VIA FUNCTION USING NEXT-FUNCTION UNIQUE OPT
40        END
41      ELSE BEGIN
42        GET TBL-FN-MODULE VIA FUNCTION USING NEXT-FUNCTION UNIQUE OPT
43        IF NOT ACCESSOK
44          THEN
45            ERROR "Error - This function not currently valid"
46
47      ;   IF FLAG-LEVEL OF TBL-FN-MODULE = "S"
48      ;     THEN
49      ;       LET FUNCTION-KEY = " "
50      ;   ELSE
51      ;   IF FLAG-LEVEL OF TBL-FN-MODULE = "A"
52      ;     THEN
53      ;       LET FUNCTION-KEY = PASS-AGENCY
54      ;   ELSE
55      ;       LET FUNCTION-KEY = PASS-FACILITY
56        LET FUNCTION-KEY = " "
57
58        GET TBL-PW-MODULE VIA PW-MODULE-KEY &
59           USING PASS-EMP-ID + NEXT-FUNCTION + FUNCTION-KEY UNIQUE OPT
60        IF NOT ACCESSOK
61          THEN
62            ERROR = &
63            "Error - Your password does not allow access to this screen"
64
65      END
66
67
68      IF NEXT-FUNCTION <> " "
69        THEN BEGIN
70        LET FUNCTION-NM = DESC-MODULE OF TBL-FN-MODULE
71        LET FUNCTION-ID = "(" + MODULE OF TBL-FN-MODULE &
72                [1:((INDEX(MODULE OF TBL-FN-MODULE,".")) - 1)] + ")"
73        RUN COMMAND FILE-EQUATE
74        RETURN
75        END
76
77
78      END
79
80
81      PROCEDURE DESIGNER NEXT NODATA  &
82          HELP "To Proceed to Next Screen Choice"
83      BEGIN
84
85        IF PASS-FACILITY = " "  AND PASS-EMP-ID = " "
86          THEN ERROR &
87          "Error - Invalid system sign-on"
88      ACCEPT NEXT-FUNCTION
89
90      DO INTERNAL NEXT-INT
91                                                    DYNAMIC MENU SYSTEMS
92      END                                                        3205-9
93
94      PROCEDURE DESIGNER MENU NODATA  &
95           HELP "To Proceed to Menu"
```

```
 96     BEGIN
 97
 98     LET NEXT-FUNCTION = "<"
 99
100       IF PASS-FACILITY = " "  AND PASS-EMP-ID = " "
101         THEN ERROR &
102          "Error - Invalid system sign-on"
103     DO INTERNAL NEXT-INT
104
105     END
106
```

```
/t
ALLNEXT.QKS,UNN
/l all
     1      ;EDITOR NAME : ALLNEXT.QKS
     2      ;AUTHOR      : AH COMPUTER SERVICES, INC (SH)
     3      ;DATE        : 11/89
     4      ;JIMS SYSTEM : MENU TO CHOOSE NEXT FUNCTION
     5
     6
     7      SCREEN ALLNEXT.QKC ACTIVITIES FIND, CHANGE  &
     8          WINDOW ON LINE 49 FOR 23 LINES &
     9           MESSAGE ON LINE 72 RECEIVING &
    10              PASS-EMP-ID, &
    11              PASS-FACILITY, &
    12              PASS-AGENCY, &
    13              PASS-FACILITY-NM, &
    14              PASS-EMP-NM, &
    15              NEXT-FUNCTION, &
    16              FUNCTION-ID, &
    17              FUNCTION-NM, &
    18              PARAMETER1, &
    19              PARAMETER2, &
    20              PARAMETER3, PARAMETER4, PARAMETER5
    21
    22      DESCRIPTION OF SCREEN &
    23      "MENU SCREEN                                                    ALLNEXT" &
    24      " " &
    25      "This is the user menu screen which allows any user access" &
    26      "to the functions allowed by the security system."
    27
    28
    29      ;*****************************************
    30      ;FILE DEFINITIONS:
    31      ;*****************************************
    32
    33
    34      ;*****************************************
    35      ; DATA DEFINITIONS
    36      ;       THE ITEMS LISTED BELOW ARE PRESENT SO
    37      ;       THAT DATA MAY BE PASSED BETWEEN SUB_
    38      ;       PROGRAMS.
    39      ;*****************************************
    40
    41      USE TEMP.USE NOL
    42
    43      FILE TBL-PW-MODULE ALIAS TBL-PW-ALIAS PRIMARY OCCURS 38 OPEN DBMODE 1
&
    44        CLOSE
    45        ACCESS VIA EMP-ID USING PASS-EMP-ID
    46        SELECT IF FLAG-LEVEL OF TBL-PW-ALIAS = "S" OR &
    47                  (FLAG-LEVEL OF TBL-PW-ALIAS = "A" AND &
    48                   FUNCTION-LEVEL OF TBL-PW-ALIAS = PASS-AGENCY) OR &
    49                  (FLAG-LEVEL OF TBL-PW-ALIAS = "C" AND &
    50                   FUNCTION-LEVEL OF TBL-PW-ALIAS = PASS-FACILITY)  OR &
    51                  (FLAG-LEVEL OF TBL-PW-ALIAS GE "1" AND &
    52                   FLAG-LEVEL OF TBL-PW-ALIAS LE "9")
    53
    54                                                          DYNAMIC MENU SYSTEMS
    55      FILE TBL-FN-MODULE ALIAS TBL-FN-ALIAS SECONDARY &                 3205-11
    56        OCCURS WITH TBL-PW-ALIAS OPEN DBMODE 1 CLOSE
```

```
57        ACCESS VIA FUNCTION USING FUNCTION OF TBL-PW-ALIAS
58
59      ;*****************************************
60      ; LAYOUT SECTION
61      ;*****************************************
62
63      USE FUNCTION.USE NOL
64
65      USE HILITEDS.USE NOL
66      DRAW THICK 3,1 TO 3,80
67      ALIGN (,,30) (,,60)
68      TITLE "|" AT ,25
69      FIELD PASS-FACILITY-NM DISPLAY PREDISPLAY
70      FIELD PASS-EMP-NM DISPLAY PREDISPLAY
71      TITLE "|" AT ,59
72      SKIP
73      USE HILITEOP.USE NOL
74      ALIGN  (,15,20) (,,60) (,,71)
75      TITLE "(ALLNEXT)" AT ,1
76      FIELD NEXT-FUNCTION NOENTRY UPSHIFT NOID LABEL "Next"
77      TITLE "|" AT ,25
78      TITLE "Individualized Menu" AT ,30
79      USE HILITEDS.USE NOL
80      FIELD SYSTEM-DATE DISPLAY PREDISPLAY
81      FIELD SYSTEM-TIME DISPLAY PREDISPLAY PIC "^^:^^:^^"
82      TITLE "|" AT ,59
83      SKIP
84
85        SKIP 2
86      ALIGN (,,10) (,,15) (,,44) (,,49)
87      CLUSTER OCCURS WITH TBL-PW-ALIAS VERTICAL FOR 1,34
88      FIELD FUNCTION OF TBL-PW-ALIAS DISPLAY
89      FIELD DESC-MODULE OF TBL-FN-ALIAS DISPLAY
90
91
92
93      PROCEDURE INTERNAL NEXT-INT
94      BEGIN
95
96      IF NEXT-FUNCTION <> " "
97        THEN BEGIN
98        GET TBL-FN-MODULE VIA FUNCTION USING NEXT-FUNCTION UNIQUE OPT
99        IF NOT ACCESSOK
100         THEN
101           ERROR "Error - This function not currently valid"
102
103     ;   IF FLAG-LEVEL OF TBL-FN-MODULE = "S"
104     ;     THEN
105     ;       LET FUNCTION-KEY = " "
106     ;   ELSE
107     ;   IF FLAG-LEVEL OF TBL-FN-MODULE = "A"
108     ;     THEN
109     ;       LET FUNCTION-KEY = PASS-AGENCY
110     ;   ELSE
111     ;       LET FUNCTION-KEY = PASS-FACILITY
.12       LET FUNCTION-KEY = "  "
113
114       GET TBL-PW-MODULE VIA PW-MODULE-KEY &
115          USING PASS-EMP-ID + NEXT-FUNCTION + FUNCTION-KEY OPTIONAL
116     ;    IF NOT ACCESSOK
```

```
117     ;       THEN  BEGIN
118     ;         LET NEXT-FUNCTION = " "
119     ;         ERROR = &
120     ;         "Error - Your password does not allow access to this screen"
121     ;         END
122
123         LET FUNCTION-NM = DESC-MODULE OF TBL-FN-MODULE
124         LET FUNCTION-ID = "(" + MODULE OF TBL-FN-MODULE &
125                 [1:((INDEX(MODULE OF TBL-FN-MODULE,".")) - 1)] + ")"
126         RUN COMMAND FILE-EQUATE
127         RETURN
128         END
129
130         LET NEXT-FUNCTION = " "
131
132     END
133
134     PROCEDURE DESIGNER AUTO NODATA HELP "To Proceed to Next Screen Choice"
135     BEGIN
136
137     LET NEXT-FUNCTION = "<"
138
139     DO INTERNAL NEXT-INT
140
141     END
142
143     PROCEDURE DESIGNER NEXT NODATA HELP "To Proceed to Next Screen Choice"
144     BEGIN
145
146     ACCEPT NEXT-FUNCTION
147
148     DO INTERNAL NEXT-INT
149
150     END
151
152     PROCEDURE DESIGNER MENU NODATA HELP "To Proceed to Menu"
153     BEGIN
154
155     LET NEXT-FUNCTION = "<"
156
157     DO INTERNAL NEXT-INT
158
159     END
160
161     BUILD
```

MODE:F ACTION: | I L S | Gene Harmon
(PCHMENU) Next | Integrated Library System | 05/05/91 18:29:46

Employee ID GENE Password
Gene Harmon

HAYWARD MAIN LIBRARY

1 15

DYNAMIC MENU SYSTEMS
3205-14

MODE:F ACTION: | 5TH FLOOR MALES | CHERYL EVANS
(ALLNEXT) Next | Individualized Menu | 05/05/1991 21:24:19

BNMU BIN MASTER MAINTENANCE
APRP ARCHIVE PROPERTY
TPAS PASSWORD MAINTENANCE
CLST CODE TABLE LISTING
IVMU INVENTORY MASTER MAINT.
OPRG OPEN REGISTER
ITST TEST FOR NEW INMATE SCRN
FAMU FACILITY MASTER MAINT.
JCMU JOB-CODE MASTER MAINT.
AGMU AGENCY MASTER MAINT.
CLRG CLOSE REGISTER
INMU INMATE MAINTENANCE
AIMT ARCHIVE INMATE
RLSE DAILY RELEASE SCREEN
TCRT CRT MAINTENANCE
PNID ASSIGN PROPERTY ID'S
BNAD PROPERTY BIN ASSIGNMENT
TIME TIME-CARD MAINTENANCE
PROP PROPERTY ENTRY

TFNM FUNCTION MODULE
TRAN PROPERTY TRANSFER
AWKR ARCHIVE WORK RELEASE
AREG ARCHIVE REGISTER BALANCE
CODE CODE TABLE MAINTENANCE
PTST PROPERTY ENTRY TEST ONLY
TPRI PRINTER MAINTENANCE
TRRG TRANSFER REGISTER CASH
ACMR ARCHIVE COMMISSARY REQU
TEST TEST FOR SECURITY
ADJC ADJUST CASH ACCOUNT
ADJP ADJUST PROPERTY CASH
BINL BIN/PROPERTY LOOKUP

Menu 1 15

DYNAMIC MENU SYSTEMS
3205-15

```
MODE:F ACTION:          |     5TH FLOOR MALES              |CHERYL EVANS
(TFNMODU)      Next     |     FUNCTION MODULE              |05/05/1991 21:25:17
```

	Fnct	Module	Desc Module	Next	Flag Mod	Flg Lvl	Usr Lvl
01	ACMR	ARCCR01A.QKC.JIMS	ARCHIVE COMMISSARY REQU		O	3	SM
02	ADJC	CSHADJ01.QKC.JIMS	ADJUST CASH ACCOUNT		O	3	SM
03	ADJP	CSHADJ02.QKC.JIMS	ADJUST PROPERTY CASH		O	3	SM
04	AGMU	AGENCYMU.QKC.JIMS	AGENCY MASTER MAINT.		O	3	SM
05	AIMT	ARCIM01A.QKC.JIMS	ARCHIVE INMATE		O	3	SM
06	APRP	ARCPR01A.QKC.JIMS	ARCHIVE PROPERTY		O	3	SM
07	AREG	ARCRG01A.QKC.JIMS	ARCHIVE REGISTER BALANCE		O	3	SM
08	AWKR	ARCWR01A.QKC.JIMS	ARCHIVE WORK RELEASE		O	3	SM
09	BINL	BINPROPL.QKC.JIMS	BIN/PROPERTY LOOKUP		O	3	SM
10	BNAD	PROPBINU.QKC.JIMS	PROPERTY BIN ASSIGNMENT		O	3	SM
11	BNMU	BINMU.QKC.JIMS	BIN MASTER MAINTENANCE		O	3	SM
12	C001	INMTPROP.QKC.JIMS	INMATE PROPERTY		C	3	SM
13	C002	INMATEUA.QKC.JIMS	VIEW ALL INMATES		C	2	SM
14	C003	CSHADJ01.QKC.JIMS	ADJUST CASH BAL		C	1	SM
15	C004	INMRATEU.QKC.JIMS	SERVICE RATES		C	3	SM
16	C005	RELSPROP.QKC.JIMS	RELEASE PROPERTY		C	3	SM

```
  Menu              Add              25    15  Update
```

DYNAMIC MENU SYSTEMS
3205-16

MODE:F ACTION:	HAYWARD MAIN LIBRARY PASSWORD MAINTENANCE	Gene Harmon 05/05/91 (TPASSWDU)

ëëë

01 Employee ID GENE NAME Gene Harmon 02 Initial Funct:

03 Password G Facil MAINUser Level SM Security Flag 1 Exit Y

ëëë

Function	Seq	Description	Level
04 ITEM		ITEM TYPE MAINTENANCE	1
05 DATE		LIBRARY CLOSED DATES	1
06 PATT		PATRON TYPE MAINTENANCE	1
07 CONV		MARC TAG CONVERSION TBL	1
08 CLST		CODE TABLE LISTING	1
09 CODE		CODE TABLE MAINTENANCE	1
10 TCRT		CRT MAINTENANCE	1
11 TPRI		PRINTER MAINTENANCE	1
12 MESS		USER MESSAGING SYSTEM	1
13 CIRC		PATRON ACTIVITY	1
14 CHKO		CHECKOUT PROCESSING	1
15 FINE		PATRON FINES PROCESSING	1

Add Add 25 15 Update

```
MODE:F ACTION:            |     5TH FLOOR MALES                |CHERYL EVANS
(TPRINTU)      Next       |     PRINTER MAINTENANCE            |05/05/1991 21:27:04
ÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈÈ
   Printer     Dev #      Description                     Type           Slave
   ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
01 TRAINING     125       IN BOH TRAINING ROOM            LASER          SP
02 REGISTER     102       CASH REGISTER                   NONE           SL
03 PAINT126     126       PAINTJET IN THE TRAINING ROOM   PAINTJET       SP
04 PAINTJET     113       PAINTJET                        PAINTJET       SL
05 LASER        113       LASER PRINTER                   LASER          SL
06 ABCDEFGH    1111       BY THE WINDOW                   DOT MATRIX     SP
07 LP           113       SYSTEM LINE PRINTER             DOT MATRIX     SL
08
09
10
11
12
13
14
15
16


  Menu             Add              25     15  Update
```

```
MODE:F ACTION:          |     5TH FLOOR MALES          |CHERYL EVANS
(TCRTSU)      Next      |     CRT MAINTENANCE          |05/05/1991 21:23:53
```

	Dev #	Function	Facility	Printer	Tray #
01	102	TRAN	5M	LASER	1
02	102	PNID	5M	LP	1
03	102	INMU	5M	LASER	1
04	115	PNID		LP	1
05	101	TRAN		LASER	1
06	103	PNID	5M	LASER	1
07	101	TRAN	5M	LASER	1
08	116	PNID		LP	1
09	101	PNID	5M	LASER	1
10	102	TRAN		LASER	1
11					
12					
13					
14					
15					
16					

```
  Menu              Add              25    15  Update
```

```
/t
TEMP.USE,UNN
/l all
    1      TEMPORARY NEXT-FUNCTION CHAR*4
    2      TEMPORARY THIS-FUNCTION CHAR*4 RESET AT STARTUP
    3      TEMPORARY FUNCTION-KEY CHAR*2
    4      TEMPORARY PASS-EMP-ID CHAR*8
    5      TEMPORARY PASS-FACILITY CHAR*4
    6      TEMPORARY PASS-AGENCY CHAR*4  RESET AT STARTUP
    7      TEMPORARY PASS-FACILITY-NM CHAR*28
    8      TEMPORARY PASS-EMP-NM CHAR*20
    9      TEMPORARY FUNCTION-ID CHAR*10
   10      TEMPORARY FUNCTION-NM CHAR*28
   11      TEMPORARY PARAMETER1 INTEGER SIZE 4
   12      TEMPORARY PARAMETER2 INTEGER SIZE 4
   13      TEMPORARY PARAMETER3 CHAR*20
   14      TEMPORARY PARAMETER4 CHAR*30
   15      TEMPORARY PARAMETER5 INTEGER SIZE 4
   16      TEMPORARY TEMP-PASSWORD CHAR*8
   17      DEFINE SYSTEM-DATE DATE = SYSDATE
   18      DEFINE SYSTEM-TIME NUM*6 = SYSTIME / 100
   19
   20      FILE TBL-FN-MODULE DESIGNER OPEN DBMODE 1
   21      FILE TBL-PW-MODULE DESIGNER OPEN DBMODE 1
   22
   23      DEFINE FILE-EQUATE CHAR*80 = &
   24                    "FILE NEXTSCRN=" + MODULE OF TBL-FN-MODULE
```

Simplified IMAGE And VIEW Calls Through The Use Of COBOL Copy Libraries

By

Rick Hoover
CIV Software
700 Hanover Dr.
Shelby N.C. 28150

As a consultant, I found that there were all different types of users of HP3000's. The type that I seemed to encounter the most were the programming staffs that did not have a good grasp of the HP inner workings. The tendency to shy away from IMAGE and especially VIEW was rampant in these shops. Most of the time I found that it was a case of lack of training. Without teaching the basics of VIEW and/or IMAGE, new programmers will not be able to fully comprehend how to access form files and/or IMAGE databases.

With this lack of understanding, I decided to create a copy library that would be as generic as possible but still be powerful enough to allow me to use the copy members to create programs for clients. The copy members that are in the handout (and should also be on the contributed library tape at the conference) will allow any user to access VIEW and IMAGE, I believe, easily and without much training. I have developed many programs and systems using the copy members.

A few months ago I met an employee of a site that I had worked at. He told me that he had little exposure (translated: none) to the HP and more importantly, VIEW and IMAGE. He tried to read the documentation in the HP manuals but didn't get very far. He then started reading my programs and copy members and found out how things really worked. This made my day.

Conventions used in the paper:

Throughout the paper I will be referring to the actual names of the copy members. The program examples will be in lower case letters (I always write my COBOL programs in lower case letters to delineate them from other peoples programs). The programs, due to their length, cannot be listed in this paper but are in excerpted form in the body of the paper. The complete IMAGE and VIEW copy member are also too long for this paper. Both the program listings and copy library listing will be available in printed form at the conference and are available at the above address (just send a large manila envelope with return postage...it's about 30 pages),. I would be happy to send these out. The 3 programs used in this paper are actual working

programs. All programs were written using the COBOL85 compiler. If you are not yet on COBOL85 (and if not, why not, see my paper on COBOL85 #3225), don't worry, these same copy members will work just as well under COBOL or COBOLII.

The forms that are used will be discussed briefly at the beginning of each section. They will not be displayed in the paper.

The copy members that will be used are:

IMAGAREA — this is the copy member that contains all the information about the IMAGE data base(s). This copy member should only be used for main programs, since there are value clauses throughout the copy member.

IMAGLINK — this is the copy member that contains all the information PASSED from the main program to the subroutine. This copy member has no value clauses at all.

VAREA — this is the copy member that has all the VIEW communication information for main programs. There are value clauses all though the copy member. Do not use this copy member for CALLed subroutines.

VLINK — this is the copy member used in CALLed subroutines where the VIEW communication area is passed. This copy member should not be used for main programs unless the programmer wishes to initialize all the fields necessary.

VIEWCALL — this copy member contains ALL calls to the VIEW intrinsics necessary to drive on-line screens. Please note that not all VIEW calls exist in the copy member. For example, this copy member does not contain the VIEW intrinsic calls if the programmer wishes to change the attributes of a field.

IMAGCALL — this copy member controls ALL access to an IMAGE data base.

RESULT — this copy member contains all the IMAGE and VIEW resultant checks.

I would like to state at this time that the terms RESULT, VAREA, VLINK, IMAGAREA and IMAGLINK have been used before by PROTOS Software in their COBOL system. The copy members here do NOT conform to the PROTOS copy members at all. I just liked the names. If you are using PROTOS in your shop, please make appropriate changes in the names.

These are the only copy members that are needed for the system.

The only other rule that I have used across the board is that there are no VIEW editing routines in any screen. The fields are all defined as type CHAR.There are a couple of routines that I wrote that are CALLed by the system that take care of numeric changes and date calculations.

Defining A Form:

The first part of the **WORKING-STORAGE SECTION** contains the definition of the form. There are two sections to the definition of the form. The first section defines the form fields:

```
01 ap000-form.
    05 ap000-company-name        pic x(20).
    05 ap000-todays-date         pic x(08).
    05 ap000-user-name           pic x(20) value spaces.
    05 ap000-screen-number       pic x(02).
```

This definition shows each field on the VIEW screen (whether display only or not) and the exact size. Please note that these fields are defined in order on the screen from left to right, top to bottom. VIEW will number the fields in a different order if the programmer adds and deletes fields on the screen as the development cycle runs. Be aware of the field numbers because you will be using them in the **PROCEDURE DIVISION** of the programs.

The second section defines the exact size of the form and the form name.

```
01 ap000-form-size              pic s9(04) comp value 50.
01 ap000-form-name              pic x(16) value "AP000".
```

Even though these two sections do not have to be placed together, it makes it nice to be able to see the two definitions together.

I have always put copy member **COPY** statements at the end of the **WORKING-STORAGE** section and at the end of the **PROCEDURE DIVISION.** That is why you will see just prior to the **PROCEDURE DIVISION** the **COPY** statements.

```
copy imagarea.
copy messrecd.
copy result.
copy varea.
```

There is a new copy member name that was not mentioned in the above list. I have included this one in the copy library only because the Accounts Payable system

has a data set that contains all the VIEW messages used. There is a short routine that will retrieve messages from the data base and place them in the *VIEW message* window.

Opening A Data Base:

The next couple of lines defines the way the data base gets opened through the copy library. There are a couple of defaults that are built into the communications procedures that you may change. The first default is the password. By default, the password is ";" (creator). The field that contains the password in working storage is called **IMAGE-PASSWORD**. The second default field contains the mode that the data base will be opened. This field is called **OPEN-MODE**. The default value is 1. Prior to performing the open paragraph, the programmer could move any other number to the field **OPEN-MODE**. There is a field in the copy library that I have set up to contain the name of the data base. If you wish, you may do the same thing, otherwise you will need to pass the data base name to any CALLed program that needs to access the data base.

The code to open the data base is:

```
move apdb-data-base          to data-base
move "WRITER"                to image-password

perform image-open

move data-base               to apdb-data-base
```

The above code will either open the requested data base or display an error message. Because this is a generic copy library, the last line is required to initialize the first two bytes of the field **apdb-data-base**. There is an 88 level field that you can use to control the program called **IMAGE-NO-ERRORS**. This is set off the **IMAGE-CONDITION-WORD**.

Opening A Forms File:

The code to open a forms file is very easy and short. Only 2 lines of code are needed to open a forms file.

```
move "APFORM"          to v-form-file-name

perform view-open
```

The copy member that actually perform the **VIEW-OPEN** will first check to insure that a forms file name was placed in the **V-FORM-FILE-NAME** field. It will then call **VOPENFORMF** to open the forms file requested. If there is a problem

opening the forms file, the program will display a message and then abort. The paragraph will then open the terminal by calling **VOPENTERM**. If an error occurs, the routine will display an error message and abort the program.

At this point the data base is opened and the forms file is opened. Now the program will go through the processing of the data.

```
move 1                to argument-9999
initialize              ap000-form

perform message-routine
```

These lines of code will initialize all fields on the screen (this is part of COBOL85, you may just as easily say **move spaces to ap000-form**). The paragraph **message-routine** will go out to the data base and move the requested message (based on **argument-9999**) to the field **v-message-line**. Remember, no screen is displayed yet. Everything for the display is set to go however.

The next lines will further ready the form for display.

```
move 2                to v-field-number
move ap00-form-name   to v-next-form-name
move ap000-form-size  to v-data-buffer-length
move ap000-form       to v-data-buffer

perform view-display-form
perform view-display-data
```

These lines of code will place the cursor in the second field on the form (in this case the field **ap000-screen-number**, remember what I said about VIEW numbering fields), then place the actual name of the form in the field **V-NEXT-FORM-NAME**, then set up the size of the form buffer. Finally the form layout gets moved to the generic **V-DATA-BUFFER**. This will allow the program to have all available information set. The performed paragraph **VIEW-DISPLAY-FORM** will first check to insure that the form name was not left blank, then call the VIEW intrinsics **VGETNEXTFORM**. If the call to the intrinsic **VGETNEXTFORM** fails, the program will be aborted with an error message, otherwise the intrinsics **VPUTWINDOW**, **VPUTBUFFER** and **VINITFORM** will be called. The form is still not on the screen so we go to the next step which is to perform the paragraph **V-DISPLAY-DATA**. This paragraph calls the VIEW intrinsics **VPUTWINDOW**, **VINITFORM**, **VPUTBUFFER** and **VPLACECURSOR**. Now everything is in ready to read the screen. The next couple of lines of code will do just that.

initialize v-number-of-errors

perform view-autoread

There are 2 paragraphs that may be performed in the copy library. One is called **view-read** the other is called **view-autoread.** The main difference is this: **view-read** will call the VIEW intrinsics **VSHOWFORM, VPLACECURSOR, VREADFIELDS, VFIELDEDITS, VFINISHFORM** and **VGETBUFFER.** The paragraph **view-autoread** will first perform **view-read** then 'fool around' with the VIEW communication area to trick the communication area into thinking that the enter key was pressed, then call **view-read** a second time, then put the VIEW communication area back together. Why you may ask. Well, this goes back to the way that VIEW works. VIEW will only transfer data from the screen to the data buffer when the ENTER key is pressed, never when function keys are pressed. So, to get VIEW to read data by using function keys, you must trick VIEW into thinking that the ENTER key was pressed. You accomplish this by setting up the communication area with a different value in the **V-TERMINAL-OPTIONS** slot. Notice that I have added 2 to the field then performed the **view-read** then subtracted 2 from the field. This is the way to automatically have VIEW re-read the screen without any key being pressed. I also save the field **V-LAST-KEY-PRESSED** so that I may find out what key was actually pressed. So now not only can programs be controlled by use of function keys, but data can be used to read data (for information...the A/P system does not use the ENTER key for any processing).

The VAREA/VLINK copy members have all nine keys defined as F0-F8 and ENTER-KEY.

All CALLs to subroutines that will use the VIEW area will need the copy member passed. So you will find a lot of:

call "AP011" using image-area, v-area ...

which will pass all information though the **linkage section** using the **vlink** copy member. This will be discussed when we get to subroutines.

Closing Data Bases

The last piece of code in the first example program will close the data base and forms file. First, the data base gets closed:

```
move apdb-data-base          to data-base

perform image-close-db
```

This performed paragraph will close the data base. The only field that will need to be set is the name of the data base to be closed. All other information is already pre-set in the performed paragraph. This will close all access to the user of the data base.

Closing Forms Files:

The closing of forms files is even easier than closing data bases. To close a forms file the program only needs to perform 1 paragraph:

```
perform view-close
```

This paragraph will call the VIEW intrinsics **VCLOSETERM** and **VCLOSEFORMF**. This will shut down all access to the forms file.

The actual flow of a main program for VIEW calls would normally be:

```
open the form file                (view-open)

display the form                  (view-display-form)

display the data                  (view-display-data)

read the screen                   (view-read or
                                   view-autoread)
go back to display the data
until exit

close the form file               (view-close)
```

There are three other perform paragraphs in the copy library. One of these, **VIEW-ERROR-ROUTINE** allows the user to set up a message in the message window. **VIEW-CUSTOM-ERROR-ROUTINE** allows the program to set an error message up and have the field become highlighted. The next one is **VIEW-CUSTOM-MESSAGE** which sets the message in the window and places the cursor in a specific field. The last one **VIEW-PRINT** allows the user to print the form on the screen. I

have not had a real need to use these recently, so I will not go into them in this paper, maybe next year.

This completes all I have to say about VIEW. There are many other intrinsic calls that can be used with VIEW, but, as I said, this copy library was meant to be one that a new programmer could feel comfortable about. One other thing I would like to mention about these routines in VIEW. HP came up with a package called HiLi, which is basically macros for VIEW. This is free and available on ALL HP3000's. I gave a talk last year on COBOL85 and HiLi and the one comment I made then and I will make it now is that HiLi is HiLi and VIEW intrinsics are VIEW intrinsics. They cannot live in harmony! You may not write a program calling HiLi intrinsics and then pass that information to a program using VIEW intrinsics or visa-versa. With this copy library of routines you may (1) pass information from a program using the copy library routines to an existing program that uses VIEW intrinsics and (2) add to the paragraphs with your own clever touches. These routines cannot be used with HiLi either.

On to IMAGE.

I purposely left out any information on IMAGE except for the opens and closes because there isn't as easy a flow as VIEW has. The IMAGE performed paragraphs will do a wealth of processing for the programmer as long as a few ground rules are understood on how the paragraphs work.

First, all calls to the IMAGE routines will need at least the data base name. Most will need the data set name. These should always be put in to insure that the performed paragraphs are initialized before each call. All access of the data base which will return data (DBGET's) will need to have a **MOVE** statement immediately after. Again, the performed paragraphs are generic and, as such, will return the data in a field called **DATA-BUFFER**. I think you will be able to see how all of this ties together over the next examples. One thing to remember, once the above simple rules are understood, you will be able to access any kind of data without knowing a lot about IMAGE (Well, that's not entirely true because the only way to understand what happens in a data base is to understand IMAGE, but, a new user can learn while doing).

The first example that will be shown is a simple inquiry to a manual master. The data set that will be accessed is **VENDOR-MASTER**. The copy member that defines the vendor master data set contains a line in the copy member:

```
01 VENDOR-DATA-SET          PIC X(16) VALUE
                            "VENDOR-MASTER;".
```

To access this data set the program needs the following information:

1. the name of the data base
2. the name of the data set
3. the argument value (in this case the vendor number which is the key)

So the code will look like this:

```
move apdb-data-base          to data-base
move vendor-data-set         to data-set
move ap001-vendor-number     to argument

perform image-calculated
```

The above code will either (1) successfully retrieve the vendor master record from the data base or (2) fail in finding the record with the value stored in the field **ap001-vendor-number**.

The next 2 lines will check out the value of the result of the call to DBGET through an 88 level assigned to **IMAGE-CONDITION-WORD:**

```
if image-no-errors
  move data-buffer          to vendor-record
else ...
```

The above code will check to see if the record was found. If it was found, the initial data would be placed in the field **DATA-BUFFER.** This is another generic field used in the processing the **DBGET.**

As you will se in the follwoing examples **image-calculated** is a powerful paragraph. It will call **DBINFO** to find out what kind of data set you are trying to access, then, if the data set is an automatic or manual master, it will call **DBGET.** However, if the data set is a detail, it will first call **DBFIND** then call **DBGET.**

The next example in the first program is to add a transaction to the data base. This is accomplished by performing first a data base lock then a put to the data base then a data base unlock. The only reason I chose a data base lock was for quick clarity. I would normally have chosen a data set lock for this kind of maintenance. The code that is required is:

```
move apdb-data-base          to data-base

perform image-base-lock

move apdb-data-base          to data-base
move vendor-data-set         to data-set
move vendor-record           to data-buffer

perform image-put

move apdb-data-base          to data-base

perform image-unlock
```

The above example will first call **DBLOCK** with an unconditional lock. There is a field that is initially set to "U" to set up the unconditional lock. If you would rather have conditional locking, set **LOCKING-TYPE** to spaces. It will then set up the data set and move the data into the generic **DATA-BUFFER** for the call to **DBPUT.** Finally, it will release the data base lock by calling **DBUNLOCK.**

The update routine is similar to the add routine in that the same exact code is used except the **perform image-put** is replaced with **perform image-update.** The

update paragraph will call **DBUPDATE** with the data buffer containing all the changed data. Please be aware that the routine will not succeed if the user attempts to change a critical item (sort or search item).

The delete routine is similar to both the add and update. In this routine the user only needs to **perform image-delete**. This routine does not need the data buffer since no data is being manipulated.

To summarize all maintenance to the data base (adds, checnges, deletes and inquires):

1. Give the name of the data base to **data-base**

2. Give the name of the data set to **data-set**

3. If detail data set, give the search item name to **search-item**

4. Give the argument value to either **argument** or **argument-9999**

5. The result data will be palced in the field **data-buffer**

6. Move the result data to your own data record layout

Now that we have laid out accessing a record in the data base, let's expand and get multiple records. We do this by either reading serially through a data set or by reading forward or backward on a chain.

There is one performed paragraph that will allow the user to read serially through a data set. The code to read a data set serially is:

```
move apdb-data-base      to data-base
move vendor-data-set     to data-set

perform image-serial-read

if image-no-errors
  move data-buffer       to vendor-record
```

This small piece of code will probably be in a performed looping paragraph. The data set will be serially read through until the end of data set happens. Then the image condition word will be set to 11.

On the other hand, let's say that you have already performed **image-calculated** on a detail data set and now wish to read the next record on the chain. To do this:

```
move apdb-data-base      to data-base
move detail-set-name     to data-set

perform image-get-next
```

The three lines of code will read the next entry in the chain. The converse of this would be to **perform image-get-previous**. Either perform will read along the chain. This way the user can read multiple records and search for data with just a few lines of code.

The rest of the performed paragraphs will take care of other tasks such as IMAGE logging, rewinding a data set, directed and primary gets and re-reading locations.

To rewind a data set (this is used to insure that the program always starts a serial read at the first record in the data set):

```
move apdb-data-base      to data-base
move vendor-data-set     to data-set

perform image-rewind
```

The data set will now begin at the first record. This routine will call **DBCLOSE** with mode 3.

To re-read the same location (this is used mainly when mass deletes are done to a manual master data set to get rid of migrating secondaries):

```
move apdb-data-base      to data-base
move vendor-data-set     to data-set

perform image-re-read
```

This will return an error 17 (no entry) if there is no record at the location.

To begin/end IMAGE logging:

```
move apdb-data-base          to data-base
move "This is the start of logging" to image-text

perform image-begin-log

do something

move apdb-data-base          to data-base
move "This is the end of logging" to image-text

perform image-end-log
```

Now IMAGE will have logical transactions in the data base.

In conclusion:

I hope this short paper will help out users, especially new users, who are unsure of IMAGE and VIEW. Granted, not everything that you can do is in this copy library, but, given the time, many other functions can be placed in the copy library to allow many different accesses and changes to the data flow of your programs. I tried to give reasonable names to the performed paragraphs so that not only could someone (myself included) could look at the programs months or years from now and still understand what was going on. I have always felt that the clearer the names, the easier to follow. I think that performing a paragraph names **image-get-next** is clearer than calling **DBGET** in mode 5.

Dear Mayor Lee and Members of the Board of Supervisors,

I am writing to express my strong support for the proposed Affordable Housing Bond for the November 2015 ballot. As a resident of San Francisco for over 20 years, I have seen many changes in our city, but few as challenging as the lack of affordable housing for working families like mine.

My family has been fortunate to find a home we can afford, but many of our friends and neighbors have been forced to leave the city they love because they could no longer afford the rent. Our teachers, nurses, and firefighters should not have to commute from hours away just to serve the community they protect.

I urge you to place this measure on the ballot and to support it with your full attention and resources. Thank you for your leadership on this critical issue.

Sincerely,

Jordan Whitfield
1247 Clement Street, Apt. 3
San Francisco, CA 94118

ans cobol 85

or

how i learned to stop worrying and love the bomb

robert a. karlin
karlins' korner
1628 van noord ave.
n. hollywood ca.
91605

(818) 982-9331

INTRODUCTION

In May of 1959, the leading suppliers of computer hardware met with the Department of Defense for the Conference on Data Systems Languages, CODASYL. Charles Phillips, from the Department of Defense, informed the representatives of IBM, Burroughs, Honeywell, General Electric, National Cash Register, Philco, Sperry Rand, RCA and Sylvania that the Pentagon wanted a uniform business programming language, and wanted it as soon as it could get it.

The conferees went to work, and by mid-autumn had developed the basis for a new language, called Common Business Oriented Language, or COBOL. But even before the designers could present the language to the full committee, another group of committee members declared the project dead, and endorsed an entirely different language developed by Honeywell, called FACT (Fully Automated Compiling Technique). In fact, Charles Phillips, who now chaired the executive committee of CODASYL, one day received a heavy crate. Upon opening it, he found a small marble tablet with a recumbent lamb carved at the top. Chiseled into the marble was the single word "COBOL". There was no epitaph.

Premature reports of the death of COBOL have since abounded. From RPG, to PL/1, to PASCAL, to BASIC, to 4GL, to SQL and other "user friendly" report writers and database update packages, to dBASE, to C, the list goes on and on. And yet COBOL still survives, and is the most commonly used language for today's business community. What is it about the language that seems to attract so many people?

First, and probably most important, many programs are written in COBOL precisely because many programs have already been written in COBOL. There is a wealth of experience to draw on, and there is a certain safety in going where everyone has gone before.

Second, COBOL seems to have a very shallow learning curve. Programmers become prolific in COBOL faster than in most other languages. This is partly because of COBOL's strengths, but is also in part because of COBOL's weaknesses. Many errors that plague other languages are impossible in COBOL, because of the lack of such features as implicit variable definition, local variables and parametric procedures.

Third, there is a large pool of trained COBOL programmers available. And, in addition, most of these programmers are also trained in modern development techniques, such as structured design and analysis, data base design and

implementation and user interaction tools and techniques. This is not always true of programmers trained only in Pascal or C.

And, finally, COBOL is an evolving language. Since the initial COBOL specifications were published in April of 1960, there have been three major language revisions, the first in 1968, the second in 1974, and the third in 1985. Each revision has added strength and power to the language. The COBOL committee of the American National Standards Institute (ANSI) is presently working on the fourth revision, to be published sometime in the mid-1990s.

In specific, the 1985 version of COBOL has provided an enormous wealth of new features. This paper describes some of these features, along with examples of how these features can provide software that is both easier to write and easier to maintain. When possible, actual programs have been used to illustrate these features.

Unfortunately, some features described here have not yet been implemented on all hardware platforms. If there is any doubt, check the specific reference manual for the hardware platform that is being used.

SCOPE TERMINATORS

One of the most useful additions made to COBOL by the 1985 ANSI standard was the addition of Scope Terminators, that is, constructs that terminate the scope of a COBOL verb. Consider the following:

COBOL-74

```
IF RECORD-IDENTIFIER EQUALS 'PAYMENT'
   COMPUTE AMOUNT = RECORD-AMOUNT / 10
   IF RECORD-TYPE = 'LOAN'
      COMPUTE AMOUNT = AMOUNT + RECORD-INTEREST
      COMPUTE PAYMENT-DUE = (AMOUNT / 365) * DAYS-PAID
   ELSE
      COMPUTE PAYMENT-DUE = (AMOUNT / 365) * DAYS-PAID.
```

Note that the computation for the PAYMENT-DUE field must be repeated twice. Now, let's look at the same calculation in COBOL-85:

COBOL-85

```
IF RECORD-IDENTIFIER EQUALS 'PAYMENT'
   COMPUTE AMOUNT = RECORD-AMOUNT / 10
   IF RECORD-TYPE = 'LOAN'
      COMPUTE AMOUNT = AMOUNT + RECORD-INTEREST
      END-IF
   COMPUTE PAYMENT-DUE = (AMOUNT / 365) * DAYS-PAID
   END-IF.
```

We can now easily see that the PAYMENT-DUE calculation is the same regardless of what the record type is, a fact that was not evident in the first instance. Also, since the PAYMENT-DUE calculation is in only one place, any modifications to that calculation are made only in that one place, preventing the possibility of missing the second calculation. Note that the final END-IF is not required by most compilers.

Let's take a look at another example:

COBOL-74

```
IF RECORD-IDENTIFIER EQUALS 'PAYMENT'
   COMPUTE AMOUNT = RECORD-AMOUNT / 10
   IF RECORD-TYPE = 'LOAN'
      COMPUTE AMOUNT = AMOUNT + RECORD-INTEREST.
```

One of the most prevalent errors in COBOL coding is the misplaced period. In the above example, a period after the first compute statement would change the calculation in a certain number of cases. Because the particular scenario that produces an error in result may only occur sporadically, it could easily go unnoticed for years. Using scope terminators, however:

COBOL-85

```
IF RECORD-IDENTIFIER EQUALS 'PAYMENT'
   COMPUTE AMOUNT = RECORD-AMOUNT / 10
   IF RECORD-TYPE = 'LOAN'
      COMPUTE AMOUNT = AMOUNT + RECORD-INTEREST
      END-IF
   END-IF.
```

If a period was accidentally placed after the first compute statement, the compiler would reject the second END-IF statement as superfluous, signalling to the programmer that an error of some sort had occurred.

It should be noted that all verbs that contain multiple operands may take a scope terminator. Whereas an END-MOVE statement seems excessive, END-READ statements that terminate an AT END condition on a read statement are extremely useful. Other scope terminators that are useful for documentation purposes are the END-COMPUTE, the END-SEARCH, and the END-[arithmetic]. We will discuss two other scope terminators, the END-PERFORM and the END-EVALUATE, later.

INLINE PERFORMS with TEST BEFORE and TEST AFTER

One of the more frustrating problems with COBOL has been the lack of effective block control structures. Looping through code involved either separating the code into a separate subroutine, or resorting to the excessive use of the GO TO verb. In addition, there was no construct that allowed the programmer to always execute a loop once, since the COBOL perform would always examine the conditional prior to executing the performed subroutine. COBOL-85 has enhanced the PERFORM statement to answer these two problems.

The first enhancement to the PERFORM statement allows the programmer to code his subroutine directly within the perform statement. For example, in COBOL-74:

COBOL-74

```
PERFORM 0100-INITIALIZE-TABLE
        VARYING TABLE-INDEX FROM 1 BY 1
        UNTIL TABLE-INDEX IS GREATER THAN 10.
                    .
                    .
                    .
0100-INITIALIZE-TABLE.
    MOVE TABLE-INDEX TO TABLE-LINE( TABLE-INDEX ).
```

while in COBOL-85

COBOL-85

```
PERFORM VARYING TABLE-INDEX FROM 1 BY 1
        UNTIL TABLE-INDEX IS GREATER THAN 10
    MOVE TABLE-INDEX TO TABLE-LINE( TABLE-INDEX )
    END-PERFORM.
```

There need not be anything within the PERFORM as in the following:

COBOL-85

```
PERFORM VARYING TABLE-INDEX FROM 1 BY 1
      UNTIL TABLE-LINE( TABLE-INDEX ) = LINE-OF-BUSINESS
    END-PERFORM.
ADD 1 TO TABLE-POLICIES( TABLE-INDEX ).
```

In addition to the inline PERFORM, COBOL-85 includes the additional syntax options WITH TEST BEFORE and WITH TEST AFTER. The default is TEST BEFORE, to maintain compatibility with earlier releases.

EVALUATE

Another traditional lack in COBOL has been a CASE structure, that is, a structure that allows a programmer to select alternatives from some form of interrogatory list. The GO TO DEPENDING ON allowed limited branching based on numeric selection, but was insufficient in providing maintainable, easy to read code in any case that had more than a half dozen selections. Extensive use of nested IFs could provide a solution in many cases, but the resultant code could give strong men nightmares.

COBOL-85 has provided the EVALUATE verb, possibly the most powerful case structure verb that exists in third generation procedural languages. In its simplest form it would look like this (I have provided an IF statement in COBOL-74 to illustrate the comparable syntax):

COBOL-74

```
IF RECORD-TYPE = 'A'
    PERFORM PROCESS-ADD
ELSE
    IF RECORD TYPE = 'C'
        PERFORM PROCESS-CHANGE
    ELSE
        IF RECORD-TYPE = 'D'
            PERFORM PROCESS-DELETE
        ELSE
            ADD 1 TO TYPE-ERROR
            PERFORM PROCESS-ERROR.
```

COBOL-85

```
EVALUATE RECORD-TYPE
    WHEN 'A'
        PERFORM PROCESS-ADD
    WHEN 'C'
        PERFORM PROCESS-CHANGE
    WHEN 'D'
        PERFORM PROCESS-DELETE
    WHEN OTHER
        ADD 1 TO TYPE-ERROR
        PERFORM PROCESS-ERROR
    END-EVALUATE.
```

The WHEN clauses need not be in any order. Each WHEN is terminated either by the next WHEN, the END-EVALUATE, or a period. WHEN OTHER is used to select all conditions not explicitly specified. If there are no statements between WHEN clauses, the EVALUATE falls through and executes the first executable statement it finds within the EVALUATE. To process a null option, the CONTINUE place holder must be used:

COBOL-85

```
EVALUATE RECORD-TYPE
    WHEN 'a'
    WHEN 'A'
        PERFORM PROCESS-ADD
    WHEN 'c'
    WHEN 'C'
        PERFORM PROCESS-CHANGE
    WHEN 'd'
    WHEN 'D'
        PERFORM PROCESS-DELETE
    WHEN 'I'
        CONTINUE
    WHEN OTHER
        ADD 1 TO TYPE-ERROR
        PERFORM PROCESS-ERROR
    END-EVALUATE.
```

The strength of the EVALUATE is in the fact that the conditional comparison may be as complex as necessary. For example:

COBOL-85

```
EVALUATE (LOAN-AMOUNT * INTEREST) / 100
    WHEN 0 THRU PRINCIPLE * .20
        PERFORM NEW-LOAN
    WHEN PRINCIPLE * .20 THRU PRINCIPLE * .80
        CONTINUE
    WHEN OTHER
        PERFORM MATURE-LOAN
    END-EVALUATE.
```

The EVALUATE command may also contain a second conditional comparison as well:

COBOL-74

```
IF RECORD-TYPE = 'L' AND RECORD-ACTION = 'A'
  PERFORM NEW-LOAN
ELSE
  IF RECORD-TYPE = 'A' AND RECORD-ACTION = 'A'
        PERFORM NEW-ACCOUNT
ELSE
    IF RECORD-TYPE = 'L' AND RECORD-ACTION = 'C'
      PERFORM CHANGE-LOAN
    ELSE
      IF RECORD-TYPE = 'A' AND RECORD-ACTION = 'C'
        PERFORM CHANGE-ACCOUNT
      ELSE
        IF RECORD-ACTION = 'D'
          PERFORM DELETE-RECORD.
```

COBOL-85

```
EVALUATE RECORD-TYPE ALSO RECORD-ACTION
    WHEN 'L' ALSO 'A'
        PERFORM NEW-LOAN
    WHEN 'A' ALSO 'A'
        PERFORM NEW-ACCOUNT
    WHEN 'L' ALSO 'C'
        PERFORM CHANGE-LOAN
    WHEN 'A' ALSO 'C'
        PERFORM CHANGE-ACCOUNT
    WHEN ANY ALSO 'D'
        PERFORM DELETE-RECORD
    END-EVALUATE.
```

And the EVALUATE command may take a completely generic conditional, that is, a construct that can choose from among many diverse and posibly unrelated choices:

COBOL-85

```
EVALUATE TRUE
....WHEN HEADER-RECORD
        PERFORM LAST-RECORD
    WHEN PRINCIPLE = 0
        PERFORM PAID-OFF-LOAN
    WHEN PRINCIPLE > INTEREST
        PERFORM MATURE-LOAN
    WHEN PRINCIPLE NOT > INTEREST
        PERFORM YOUNG-LOAN
    END-EVALUATE.
```

Note that the EVALUATE command processes the conditional statements in the order that they are expressed. In the above example, when PRINCIPLE equals zero, the second condition will be executed, and then the EVALUATE verb is exited, even though the last condition may also seem to apply. It should be noted that FALSE is also a valid generic conditional for the EVALUATE verb.

SETTING CONDITIONALS

One of the more interesting features of COBOL was the 88 level conditional data. This feature allowed programmers to provide meaningful descriptions of codes and switches in programs. Unfortunately, the programmer still needed to know what the switch setting and the switch name was in order to set the proper value. COBOL-85 provides a new method of setting 88 level conditionals using the SET verb.

```
01  SWITCHES.
    05  END-OF-FILE-SWITCH              PIC X VALUE 'N'.
        88  END-OF-FILE                       VALUE 'Y'.

COBOL-74
    READ INPUT-FILE
        AT END MOVE 'Y' TO END-OF-FILE-SWITCH.

COBOL-85
    READ INPUT-FILE
        AT END SET END-OF-FILE TO TRUE.
```

Note that not only is the programmer insulated from the value of the switch itself, but the switch is being set with the same name that it will be interrogated with later. If there were multiple values coded for an 88 level, the COBOL compiler will choose the first VALUE to move to the switch:

```
01  SWITCHES.
    05  RECORD-TYPE-FLAG                PIC X.
        88  DEPOSIT                     VALUE 'D'.
        88  LOAN PAYMENT                VALUE 'L'.
        88  WITHDRAWAL                  VALUE 'W'.
        88  MONEY-IN                    VALUE 'D' 'L'.
        88  MONEY-OUT                   VALUE 'W'.

COBOL-85
    SET MONEY-IN TO TRUE.
```

In the above example, RECORD-TYPE-FLAG would be set to 'D', since that was the first value in the value list. Unfortunately there is no way to SET an 88 level to FALSE, since the compiler would not know which value to chose.

INITIALIZATIONS

COBOL-85 has provided a number of interesting features to facilitate some common programming chores. One of the most useful is the INITIALIZE verb. In its simplest form, the INITIALIZE verb will set all subordinate levels for a data item to zeroes or spaces, depending on the type of data being initialized. The INITIALIZE statement will also initialize any class of data subordinate to a data item to any characters that are appropriate to that data class.

```
01  DATA-STRUCTURE.
    05  FILLER                          PIC X.
    05  NUMERIC-ITEM                    PIC 9.
    05  ALPHA-ITEM                      PIC X.
    05  FILLER.
        10  NUMERIC-EDITED-ITEM         PIC 9.9.
```

COBOL-85

```
INITIALIZE DATA-STRUCTURE.
```

Is equivalent to

COBOL-74

```
MOVE ZEROES TO NUMERIC-ITEM,
               NUMERIC-EDITED-ITEM.
MOVE SPACES TO ALPHA-ITEM.
```

Note that the INITIALIZE verb does not move spaces to elemental FILLER items.

COBOL-85

```
INITIALIZE DATA-STRUCTURE REPLACING ALPHANUMERIC BY '*'.
```

This construct will move asterisks to ALPHA-ITEM. NUMERIC-ITEM and NUMERIC-EDITED-ITEM will be unchanged.

The INITIALIZE verb will work on all occurrences of table items declared with the OCCURS clause, but will not affect items that are INDEX items, and items that contain or are subordinate to a REDEFINES clause (though DATA-STRUCTURE may contain a REDEFINES clause or be subordinate to one).

MOVE ENHANCEMENTS

COBOL-85 finally corrects a deficiency that has plagued software designers since the language was originally developed , that being the ability to dynamically access part of an alphanumeric field. The STRING and UNSTRING commands, added by the COBOL-74 standard, allowed the programmer to parse a field, if there were a limited number of delimiters, if the number of fields were known, if the resulting field sizes were known, if ..., if ..., if ... . COBOL-85 provides Reference Modification, the ability to specify a starting byte position (relative to one) within a field, and the number of bytes to move (i.e. [START]:[LENGTH]):

```
COBOL-85
01  ALPHAMERIC-LINE                    PIC X(80).
01  ALPHAMERIC-LINE-2                  PIC X(40).

MOVE ALPHAMERIC-LINE ( 21:10 ) TO ALPHAMERIC-LINE-2.
MOVE 'LITERAL' TO ALPHAMERIC-LINE ( 32:7 ).
```

The move takes place using the rules for moving simple alphanumeric fields. The fields must be defined as USAGE DISPLAY (the COBOL default), and, if the sending field is numeric or numeric edited, it is treated as if it had been redefined a s simple alphanumeric field. The starting position and/or the length may be any arithmetic expression.

```
COBOL-85
PERFORM VARYING POSITION FROM 1 BY 1 UNTIL POINTER > 10
    MOVE ALPHAMERIC-LINE ( POINTER:1 ) TO
        ALPHAMERIC-LINE-2 ( POINTER * 2:1 )
    END-PERFORM
MOVE SPACES TO
 ALPHAMERIC-LINE-2((POINTER * 2) + 1:32 - (POINTER * 2))
```

The last line translates as MOVE SPACES to the beginning of ALPHAMERIC-LINE-2 plus 1 plus (POINTER times 2) bytes, for a length of 32 minus (POINTER times 2).

Another enhancement to COBOL provides the DEEDITED move, that is, the ability to move from a numeric edited field to a computational numeric field.

COBOL-85

```
01  NUMERIC-EDITED           PIC ZZZ9.99CR.
01  NUMERIC-CALCULATED       PIC S9(4)V99 COMP-3.

MOVE -35.42 TO NUMERIC-EDITED.
MOVE NUMERIC-EDITED TO NUMERIC-CALCULATED.
```

Note that NUMERIC-EDITED cannot be used on the right side of a computation as in: NUMERIC-CALCULATED = NUMERIC-EDITED + 1.

NESTED SOURCE PROGRAMS

Earlier in this text it was mentioned that COBOL does not provide the ability to use parametric procedures. This is not entirely correct. COBOL-85 has provided a method of local variable storage and parametric procedures: the nested program. A nested program must occur at the end of the procedure division, and is treated very much like an external called program. Like a main program (see Miscellaneous Enhancements), a nested program does not need to contain all four divisions. A program may optionally declare data or files that may be referenced by all programs subordinate to it, or may declare data or files that can be shared by any program in the run unit, that is, the aggregate code file produced by the compile and link.

```
COBOL-85
IDENTIFICATION DIVISION.
PROGRAM-ID. CALLER.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILE-1 ASSIGN TO ZZZZZ.
    SELECT FILE-2 ASSIGN TO YYYYY.
DATA DIVISION.
FILE SECTION.
FD  FILE-1 IS GLOBAL.
01  RECORD-1.
    03  FIELD-11          PIC X(10).
    03  FIELD-12          PIC X(10).
FD  FILE-2.
01  RECORD-2 IS GLOBAL.
    03  FIELD 21          PIC X(10).
WORKING-STORAGE SECTION.

77  STATUS-FIELD          PIC X(1).
    88  GOOD                      VALUE 'Y'.
    88  BAD                       VALUE 'N'.

PROCEDURE DIVISION.

0000-MAINLINE.
    OPEN INPUT FILE-2
```

```
          OUTPUT FILE-1.
      READ FILE-2
          AT END MOVE HIGH-VALUES TO RECORD-2.
      PERFORM UNTIL RECORD-2 = HIGH-VALUES
          CALL 'SUBPROGRAM-1' USING STATUS-FIELD
          IF GOOD
              CALL 'SUBPROGRAM-2' USING STATUS-FIELD
              IF BAD
                  DISPLAY 'BAD RECORD ' RECORD-2
                  END-IF
              END-IF
          READ FILE-2
              AT END MOVE HIGH-VALUES TO RECORD-2
              END-READ
          END-PERFORM.
      CLOSE FILE-1
            FILE-2.
  IDENTIFICATION DIVISION.
  PROGRAM-ID. SUBPROGRAM-1.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  COUNT                PIC S9(9) VALUE 0 EXTERNAL.
  LINKAGE SECTION.
  77  STATUS-FIELD              PIC X(1).
      88  GOOD                         VALUE 'Y'.
      88  BAD                          VALUE 'N'.
  PROCEDURE DIVISION USING STATUS-FIELD.
  0000-MAINLINE.
      IF RECORD-2( 1:2 ) = 'OK'
          SET GOOD TO TRUE
          ADD 1 TO COUNT
      ELSE
          SET BAD TO TRUE.
      EXIT PROGRAM.
      END PROGRAM SUBPROGRAM-1.
  IDENTIFICATION DIVISION.
  PROGRAM-ID. SUBPROGRAM-2.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  COUNT                PIC S9(9) VALUE 0 EXTERNAL.
```

```
LINKAGE SECTION.
77  STATUS-FIELD            PIC X(1).
    88  GOOD                        VALUE 'Y'.
    88  BAD                         VALUE 'N'.
PROCEDURE DIVISION USING STATUS-FIELD.
0000-MAINLINE.
    MOVE FIELD-21 TO FIELD-11.
    MOVE COUNT TO FIELD-12.
    WRITE FILE-1.
    SET STATUS-FIELD TO GOOD.
    EXIT PROGRAM.
    END PROGRAM SUBPROGRAM-2.
    END PROGRAM CALLER.
```

The GLOBAL keyword allows all subordinate programs to reference the file and/or data item that contains it, as well as all data items subordinate to the GLOBAL item. The EXTERNAL keyword defines a data area that is common to all programs that include the definition. Note that if there are subordinate items to the EXTERNAL item, they must be defined exactly the same in all referenced cases, but the data area may be subsequently redefined.

A nested program may also include the keywords COMMON and INITIAL on the PROGRAM-ID line. The COMMON keyword specifies that the program may be called by any program in the run unit. The INITIAL specifies that all items are to be reset to their initial state, that is, to either the values specified in the VALUE clauses, or to an undefined state if there is no VALUE clause specified.

```
COBOL-85
IDENTIFICATION DIVISION.
PROGRAM-ID. SUBPROGRAM-3 IS COMMON PROGRAM.
```

```
COBOL-85
IDENTIFICATION DIVISION.
PROGRAM-ID. SUBPROGRAM-4 IS INITIAL PROGRAM.
```

```
COBOL-85
IDENTIFICATION DIVISION
PROGRAM-ID. SUBPROGRAM-4 IS INITIAL COMMON PROGRAM.
```

Note also that multiple COBOL-85 programs may follow one another in a compilation stream. Each program is terminated by an END PROGRAM statement, or by the termination of the input stream.

MISCELLANEOUS ENHANCEMENTS

COBOL-85 has provided a number of minor enhancements that are useful to know about. One of these is enhancement to the CALL verb to allow data to be passed by value instead of by reference.

```
COBOL-85
01  FIELD-1          PIC X VALUE 'A'.
01  FIELD-2          PIC X VALUE 'B'.
01  FIELD-3          PIC X VALUE 'C'.

CALL 'SUBPROG' USING BY REFERENCE FIELD-1
                                  FIELD-2
                     BY CONTEXT   FIELD-3.
```

Another enhancement included in COBOL-85 is the implied FILLER statement.

```
COBOL-74
01  FIELDS.
    03  FILLER                          PIC X.
    03  FIELD-1                         PIC XX.
    03  FILLER REDEFINES FIELD-1.
        05  FILLER                      PIC X.
        05  FIELD-12                    PIC X.

COBOL-85
01  FIELDS.
    03                                   PIC X.
    03  FIELD-1                         PIC XX.
    03  REDEFINES FIELD-1.
        05                               PIC X.
        05  FIELD-12                    PIC X.
```

And yet another enhancement found in COBOL-85 is the reduction in the minimum program. The following illustrates the minimum compilable program.

```
COBOL-74
IDENTIFICATION DIVISION.
PROGRAM-ID. COBOL74.
ENVIRONMENT DIVISION.
```

```
DATA DIVISION.
PROCEDURE DIVISION.
    STOP RUN.
```

COBOL-85

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBOL-85.
```

The above also illustrates a fourth enhancement to COBOL-85. The compiler will automatically generate a program exit after the last line of the program if the program falls through it. The compiler will also automatically close all open files when the program is exited.

COBOL-85 provides another arithmetic operator, **, for exponentiation. 3 ** 2 will result in three squared, or 9. The exponent may be fractional, 4 ** .5 will result in the square root of 4, resulting in 2. The exponentiation operator may appear anywhere that any other arithmetic operator may occur.

COBOL-85 allows fields that contain OCCURS clauses to be initialized using a VALUE clause. This would be equivalent to coding each entry separately with a VALUE clause.

COBOL-74

```
01  FIELDS.
    05  FIELD-A.
        10  FILLER                  PIC 9 VALUE 0.
        10  FILLER                  PIC 9 VALUE 0.
        10  FILLER                  PIC 9 VALUE 0.
    05  FILLER REDEFINES FIELD-A.
          10  FIELD-1                    PIC 9 OCCURS 3
TIMES.
```

COBOL-85

```
01  FIELDS.
    05  FIELD-1    PIC 9 OCCURS 3 TIMES VALUE 0.
```

COBOL-85 allows the substitution of BINARY and PACKED-DECIMAL for COMP and COMP-3, respectively.

In COBOL-85, the CONTINUE statement acts as a NO OPERATION and may

occur anywhere a COBOL-85 procedure division statement may occur.

COBOL-85 enhances the INSPECT verb, adding the INSPECT CONVERTING option:

```
COBOL-85
INSPECT FIELD-A CONVERTING 'ABC' TO 'DEF'.
```

COBOL will examine each byte of FIELD-A, comparing it to each byte of the string 'ABC'. If a match is found, COBOL will replace it with the corresponding byte from the string 'DEF'. If FIELD-A contained 'CAT', the above would convert it to 'FDT'.

COBOL-85 provides two new class tests, ALHABETIC-UPPER and ALPHABETIC-LOWER.

```
COBOL-85
IF FIELD-A IS ALPHABETIC-LOWER
    PERFORM UPSHIFT-FIELD-A.
```

COBOL-85 allows the programmer to define a SYMBOLIC to identify a particular character in an alphabet.

```
COBOL-85
SPECIAL NAMES.
    SYMBOLIC CHARACTER BEL IS 07.

MOVE BEL TO FIELD-A.
DISPLAY BEL 'WAKE UP'.
```

COBOL-85 allows the programmer to specify his own class test for use in conditionals. When used in a conditional phrase, COBOL-85 checks each character in the compared field to determine if it is part of the class. In the following example, if all characters in FIELD-A were A or B or C or Q or Z, the conditional would be true and the MOVE would be executed.

```
COBOL-85
SPECIAL NAMES.
    CLASS A-THRU-C-AND-QZ IS 'A' THRU 'C' 'Q' 'Z'.
```

```
IF FIELD-A IS A-THRU-C-AND-QZ
    MOVE FIELD-1 TO FIELD-2.
```

COBOL-85 allows subscripted and indexed tables to be referenced by an offset to a current subscript or index.

```
COBOL-85
MOVE TABLE-ENTRY (INDICE + 1) TO TABLE-ENTRY (SUB - 3).
```

And, finally, COBOL-85 elimitates the REMARKS section in the IDENTIFICATION division, and the NOTE paragraph in the PROCEDURE division. These are considered replaced by the COBOL-74 '*' (comment) in column 7 construct.

BOMBS AWAY

COBOL has changed greatly since its conception in 1960. And COBOL has grown to be the most widely used business programming language today. Much of the credit for this goes to the original design team, who created a language that was easy to understand and simple to use. But credit must also be given to the American National Standards Institute Technical Committee for their effort in keeping COBOL a living, growing product that is responsive to the needs of current users. The next version of COBOL will probably be available by the middle of this decade. Under discussion are enhancements to provide Object Oriented extentions, network related structures, and asynchronous task support. Copies of the current standard may be obtained from:

American National Standards Committee
1430 Broadway
New York, NY. 10018

Ask for ANSI Standard X3.23-1985. There will be a nominal publication charge. Any comments about COBOL-85, or enhancement suggestions should be addressed to TECHNICAL COMMITTEE X3J4 (COBOL) at the above address.

Paper #3208

Integrating the OMNIDEX IMS Into Your System Applications

Tim Klooster
DYNAMIC INFORMATION SYSTEMS CORPORATION
5733 Central Avenue
Boulder, Colorado 80301
(303) 444-4000

This paper will present examples of applications using the OMNIDEX Information Management System (IMS). These examples represent actual systems or designs that have incorporated features of the OMNIDEX IMS and put into practice the concepts discussed.

The purpose is to present these examples in a way that will help the reader think of ways to incorporate similar implementations into his own applications.

The examples presented illustrate database design with the OMNIDEX IMS, keywording, IMSAM discrete mode, and Document Management.

This paper presumes that the reader has a basic knowledge of the concepts used by the OMNIDEX IMS.

EXAMPLE 1: KEYWORD RETRIEVAL

Problem Tracking System:

This example takes advantage of the power of OMNIDEX keywording. Keywording simply refers to the ability of OMNIDEX to parse or break down a field by its special characters and give retrieval access to the field by any word within the field. The power of this feature becomes especially apparent with a large descriptive or textual field where many keywords exist within the field.

Another feature utilized in this example is data item grouping. Grouping in OMNIDEX is a feature where two or more fields are logically treated as a single entity. This allows multiple fields to be searched simultaneously for a value or values.

This feature is set up during the OMNIDEX installation simply by appending the grouping option to the field name. When searching for a value in the field, OMNIDEX recognizes that it is a grouped field and automatically searches across all fields in the group for the same value.

The requirements for this application included the ability to catalog and retrieve data associated with all customer accounts.

Additionally, storage of all problem situations and the resolutions for each account was required. These situations could be referenced when a similar problem was encountered. The database would serve as a "knowledge base" for solving problems.

Since the customer service department assists customers as they call in, it was necessary for them to be able to access the data while the customer remained on the phone. This required their data processing department to create a system that would give the users a fast and flexible environment for their data retrievals.

The design that was implemented included a customer master dataset with associated problem description and resolution description detail datasets. The customer master provided keyword lookups by customer name, company, title, city, state, zip, and phone. The customer name, title, company name, and company-alias fields were grouped together so that the user could simply enter any of the information at a single prompt and retrieve the master data immediately.

The problem detail dataset provided keyword access on the comments and error description fields. The users can access this information by an error code or by entering any word in the description or comments fields. The resolution detail dataset provided access by any word in the description or comments fields. The customer service department can now provide assistance to their customers while the customer is on the phone.

This system also allowed the company to catalog and isolate problem areas with their service. Daily and monthly summary reports are generated showing areas needing attention. Forecasting of future problems is also possible.

Keywording can be a benefit to any application where flexible retrievals are required on data that is textual in nature.

Other applications that can benefit from keywording include marketing, legal case tracking, and medical chart tracking.

EXAMPLE 2 - DATABASE DESIGN
Sales Order Database

This example uses a sales order database designed in IMAGE to allow multiple path entry into the primary datasets (see Figure 1-1). KSAM files were used to allow partial key lookups by product name and customer name. Automatic master datasets were created for access by sales date and order number.

In the redesigned database model (see Figure 1-2), we see that OMNIDEX and IMSAM have been incorporated to create a much simpler database structure.

The KSAM files were replaced with OMNIDEX keys on product name and customer name. This eliminates the worry of maintaining external KSAM files. If a sort is required on either of these fields, IMSAM could be added.

The automatic master datasets, sales-dates and orders, have been replaced by OMNIDEX keys. The inventory dataset can now become a manual master dataset physically as well as logically with the multiple key capability of OMNIDEX.

We now have restored the natural IMAGE structure of a parent-child relationship (one record to many) between the order header and order lines datasets.

This structure allows much easier retrievals across the datasets because we now have data structures that are correctly represented in IMAGE.

EXAMPLE 3 - DATABASE DESIGN
Franchise Management System

This example illustrates how OMNIDEX domains can allow a hierarchical tiered structure design in an IMAGE database. Hierarchical databases traditionally facilitate a "top-down" style of data retrieval much better than a network database like IMAGE.

Figure 2-1 shows the structures involved in a franchise management system. Each level has a one to many relationship with the level below it. For example, there are multiple divisions within a company and each company has multiple offices.

Network databases such as IMAGE are designed for a two level or single master to many detail dataset relationship. The master dataset represents a single entity with the detail data sets representing multiple entities or events associated with the master dataset. Since IMAGE will not allow levels below a detail dataset, we find our designs going horizontally instead of "top-to-bottom" as they logically are in real life.

Using OMNIDEX search item (SI) domains, we can simulate a hierarchical design inside of IMAGE. A search item domain refers to indexing a detail dataset with its associated master dataset. Figure 2-2 shows the IMAGE design of the structure outlined in Figure 2-1. Automatic masters are used to allow for OMNIDEX SI domains to be placed around each detail dataset. Generally, automatic master indexes are not required in IMAGE when OMNIDEX or IMSAM indexes are present. I use them in this design to create an SI domain for each detail dataset.

The multifind function in OMNIDEX will allow us to easily cross SI domains for our retrievals requiring multiple dataset access. These domains illustrate a "sawtooth" design which generally favors the "top-down" type of retrievals that are difficult in IMAGE.

The retrieval outlined below illustrates how a "top-down" retrieval can be easily performed in IMAGE using OMNIDEX SI domains. If we ask the question, "How many people under the age of 21 are employed in the Colorado region?", we are required to start at the companies level and end up in the people level.

IMAGE retrieval without OMNIDEX:

The retrieval process would begin with a serial read of the companies dataset since state would not be the most likely key. To find the associated divisions for each qualifying company, we would do a chained read into the division detail set. We would continue by finding the chain head for every office within each qualifying division and then read down the chain into the office dataset. We would continue this process until we reached the people dataset, where program logic would be required to select the age group while reading down the chain.

To satisfy this retrieval, we had to perform a serial read of a dataset, build record selection logic into our program, and read all the records in each detail chain, whether we needed them or not.

By retrieving records that we don't need, through a serial or chained read, we incur an increase in the time it will take to execute this retrieval. Building selection logic into a program also requires programming time when developing this application. These requirements often preclude the ability to perform ad-hoc reporting requests against our data.

OMNIDEX retrieval:

To perform this retrieval using OMNIDEX, the multifind feature would be utilized.* The serial read in IMAGE of the company dataset would be replaced with an ODXFIND intrinsic call to qualify only the entries that we require. We then perform our "top-down" retrieval process using multifind to cross the domains. This process requires only a call to ODXFIND to qualify the entries in our target domain. When we reach the people domain, we can qualify the entries that satisfy our age requirement.

An added performance benefit of this design is that multifind takes action only against the OMNIDEX index sets. In most cases, this can result in a much faster qualification of the entries compared to retrieving the records from the IMAGE datasets.

This action lends itself easily to an ad-hoc query environment as long as the number of OMNIDEX IDs that are qualified and used as input to the multifind operation are kept at a reasonable number. DISC recommends that this number be under a thousand.

Alternative design:

An alternative to using automatic master datasets and search item domains in the above design would be to create stand-alone detail datasets for each level of the hierarchy. OMNIDEX detail domains (DR) would be installed under detail datasets. OMNIDEX indexes would be placed on the common fields between the detail datasets.

A new feature was added to OMNIDEX version 2.05/2.06 that allows any specified field to be written to an ASCII file using the ODXTRANSFER intrinsic. The contents of this file is then used as input into the ODXFIND intrinsic against the target dataset. The ODXTRANSFER call uses the new mode +100 and allows you to specify the field you want to transfer in the "options" parameter.

This feature allows multifind to cross detail domains whereas before you were limited to crossing search item domains or into one detail domain.

The advantage of this design is that it's more simplistic and provides improvement in update overhead over the design using automatic master datasets. The enhancement to ODXTRANSFER allows the creation of a "relational-like" environment where linkages between files can be dynamically created as needed instead of being pre-defined at design time. The files can be inside the same database or in different databases. The only requirement is a common data item with OMNIDEX installed on the target dataset.

The disadvantage of using stand-alone details and ODXTRANSFER is a possible increase in processing time due to the disk activity of writing and reading the ASCII file that is used as the link between the two datasets. Careful planning of the relationships between datasets when setting up your design can help insure acceptable retrieval times.

*For a discussion of Multifind, refer to Page 2-78 of the OMNIDEX Administrators Guide

EXAMPLE 4: IMSAM DISCRETE MODE
Statistical Reporting System

Discrete mode refers to taking action against the OMNIDEX and IMSAM indexes only. By comparison, normal mode refers to action taken against both the OMNIDEX and IMSAM indexes along with the IMAGE datasets.

IMSAM discrete mode can greatly benefit data retrieval access times since information can generally be extracted from the IMSAM indexes much faster than the IMAGE datasets. Additionally, IMSAM returns the data sorted to your program.

Discrete mode requires that all required data is stored in the IMSAM index key. This is accomplished using composite keys. Composite keys are a feature of IMSAM where all or parts of multiple fields can be concatenated into one key. When the data is returned, it is sorted in the order of the components of the composite key. Currently, you can have up to seven components in an IMSAM composite key for a total of 128 bytes in length. IMSAM discrete mode is generally 30 - 100 times faster than normal mode.

This application called for the ability to store and access a very large amount of data for a period of two to three years. IMSAM was chosen to provide multiple level sorting on the data.

Discrete mode retrievals were chosen to provide the performance required in the monthly reporting cycle. The application began with a few million records and would grow up to ten million in the next year.

A single stand-alone detail dataset was created for storage. *(Note: A stand-alone b-tree is possible for this application. A stand-alone detail set was chosen because updates were performed on a monthly basis, requiring all the data to be present in either a flat file or data set to load into the IMSAM indexes.)*

IMSAM composite keys were constructed based on the sort requirements and control breaks for the monthly summary reports. All sales amounts and quantities were included at the end of the key.

Reporting the data utilized IMSAM discrete mode by calling DBIGET using mode 1300 to position the pointer at the requested area of the b-tree. Subsequent DBIGETS with a mode 1090 are then used to read down the b-tree.

A sample performance analysis of expected retrieval times is as follows:

1. IMSAM has a maximum tree block size of 4096 bytes.
2. One of the IMSAM composite keys has a length of 64 bytes.
3. Each disk drive access will yield a maximum 64 IMSAM keys.

   (IMSAM block size / IMSAM keysize = number of keys per read)
4. The average number of records per month = 416,667.

   (10 million total records / 24 months = records per month)
5. Most monthly reports will report the previous months sales and a comparison with the same month in the previous year.
6. An average of 833,333 records will be read for each report. (416,667 x 2 = 833,333) - 2 months
7. The disk drive average access time per monthly report would be approximately 6.6 minutes.

   The calculation is as follows:

   - 833,333 / 64 (recs per block) = 13,021 I/O's (rounded)
   - add 8 I/O's to position record pointers in the b-tree
   (4 for previous year + 4 to reset at current year)

   - 13,029 I/O's / 33 = 395 seconds or 6.6 minutes
   (33 = HP benchmark of the number of I/O's per second for an Eagle disk drive)

This performance analysis shows that it is possible to have great performance on a large number of records in your dataset when using IMSAM discrete mode for the retrieval.

When setting up your IMSAM composite keys, use the following guidelines:

1. Your selection fields should be the leftmost fields in the composite key.
2. The order of the components should match the sort order of your report.

In the event that your selection order differs from your sort order, you should maintain the sort order as your most important criteria. The selection fields can be spread out over the composite key in some cases. For example, if you need to select on the first and forth components of your key, you must insure that the second and third components are set to low values so they don't cause unwanted selections. You must also monitor those fields as you read down the b-tree. It may be required that you periodically reposition your record pointer as these values change.

Other good applications for IMSAM discrete mode include any large historical application, an online general ledger, or any ad-hoc requirements where all the required data can be stored in the key.

EXAMPLE 5: DOCUMENT MANAGEMENT

Correspondence Tracking System

Document management in OMNIDEX provides the ability to index any ASCII file. Each word in the document can be indexed allowing for keyword retrieval anywhere in the document. The DATADEX KEYWORD command must be used to load the documents into the index sets. Datadex or any programming language can then be used for retrieving and displaying the documents. The only design requirements for a document management system are an IMAGE master dataset with fields to store an internal number for each document and the name of the file that has been indexed. Retrievals are then performed against the OMNIDEX index sets and file names are returned to the program. The ODXVIEW intrinsic is used to view the files online in your programs. ODXVIEW allows for user input while viewing the document. The user is allowed to scroll forward or backward in the file by lines or pages.

A large corporation needed to catalog internal and external correspondence and track the routing of memos internally. Functionality included the ability to retrieve documents by the sender, sendees, memo subject, and document keywords. The documents were scanned and loaded into ASCII files on the HP. DATADEX was then used to load the keywords into a database.

The database included a master dataset with the following fields:

- document number	(J2) KEY
- filename	(X26)
- subject	(X30)
- sender initials	(X4)
- sendee initials	(X4)
- memo-date	(J2)
- comments	(X30)

A memo routing detail dataset allowed for memos to be passed to other people with the senders comments and tickler and due dates.

The master and detail sets were indexed together using a search item domain installation. Memos could be retrieved by any master field or document keyword. A screen function key was set up to list the corresponding memo routing information. This detail set was linked to the master using the document number search item. Document number was sorted by a send date field so that the most recent routing would be displayed first. Viewing the actual memo is possible using the file name from the master dataset as input into a call to the ODXVIEW intrinsic.

Utilizing a document management system like this allows instant access to any document without having to search for paper copies or through documents archived to microfilm.

Other examples of document management systems include a source code cross-reference, online reference manuals, and legal documents.

SUMMARY

There are many places where an indexing system like the OMNIDEX IMS can be used in your applications. Database design can be heavily influenced when OMNIDEX or IMSAM are used. Remember, you don't have to change your database design to use the OMNIDEX IMS though! If your existing database cannot be altered, you can still install OMNIDEX or IMSAM without affecting your data.

The keywording power of OMNIDEX can be used in practically any application. IMSAM provides a great alternative to sort keys when your chains are long or chain maintenance is high.

Discrete mode retrievals in IMSAM can make complex reporting possible when before it was considered impossible. Larger amounts of data can be stored and accessed online. When deciding whether to use IMSAM discrete mode or not, it is highly beneficial to work through an analysis of your disk overhead in retrievals and updates. This analysis will provide you with a basis to determine keysize, components of the keys, and tree block size.

You will also be able to predict the speed of your retrievals and which reports and queries are feasible.

Document management also illustrates the power of relational keywording. The ability to retrieve and maintain documents online can be of value in many applications.

Hopefully, I have presented an example or two that made you think of a new way that you can put your new indexing system to work in your company. Getting information to your users in a fast and flexible way is becoming more and more important every day.

Good luck with your new applications!

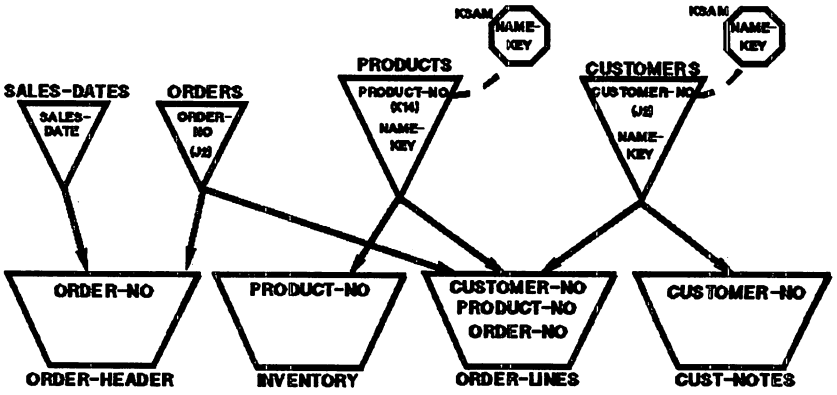

EXISTING DATA BASE
KSAM
NAME-KEY
KSAM
NAME-KEY
PRODUCTS
PRODUCT-NO (X14)
NAME-KEY
CUSTOMERS
CUSTOMER-NO (X5)
NAME-KEY
SALES-DATES
SALES-DATE
ORDERS
ORDER-NO (X5)
ORDER-NO
ORDER-HEADER
PRODUCT-NO
INVENTORY
CUSTOMER-NO
PRODUCT-NO
ORDER-NO
ORDER-LINES
CUSTOMER-NO
CUST-NOTES

Figure 1-1

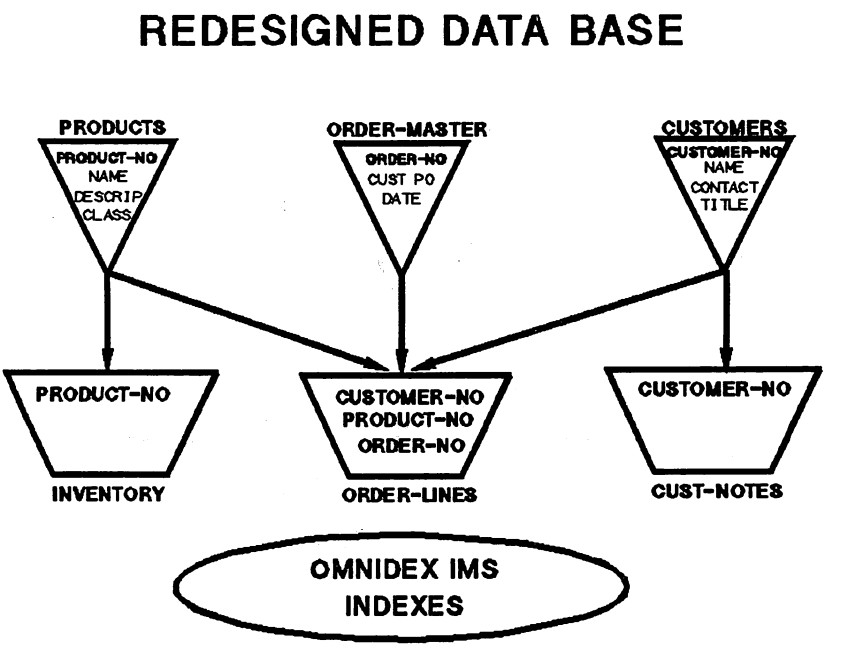

REDESIGNED DATA BASE
PRODUCTS
PRODUCT-NO
NAME
DESCRIP
CLASS
ORDER-MASTER
ORDER-NO
CUST PO
DATE
CUSTOMERS
CUSTOMER-NO
NAME
CONTACT
TITLE
PRODUCT-NO
INVENTORY
CUSTOMER-NO
PRODUCT-NO
ORDER-NO
ORDER-LINES
CUSTOMER-NO
CUST-NOTES
OMNIDEX IMS
INDEXES

Figure 1-2

Database Design

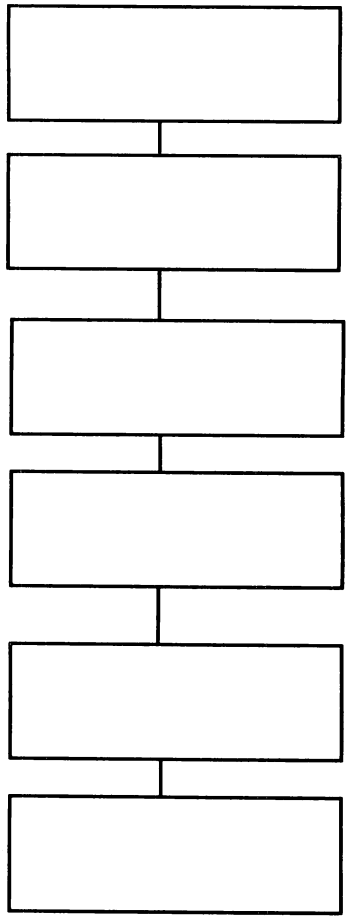
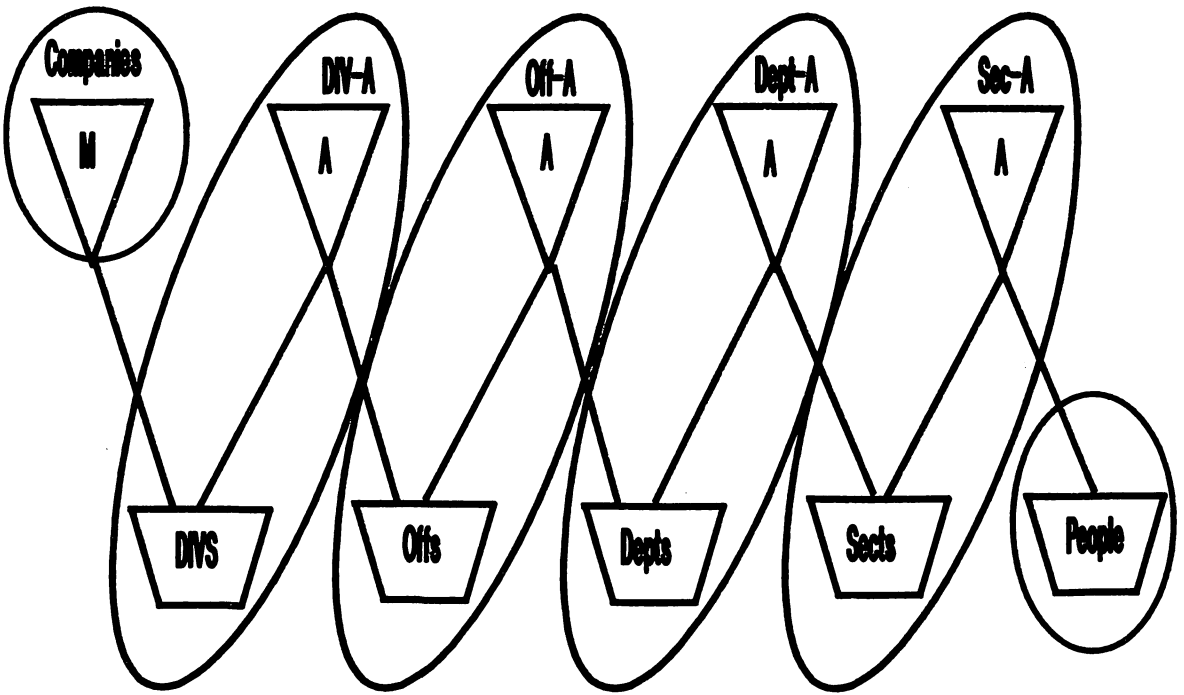


Figure 2-1

Figure 2-2



3208-12

Paper #:3209

The EH Safety Representative Information System on the Safety Performance Measurement System is where you will find... Word Processing and Helps with a V - P L U S!

Patricia Irene Loo
EG&G Idaho, Inc.
P.O. Box 1625
Idaho Falls, Idaho 83415-3405
(208) 526-6063

1.0 Introduction

What are some of the current environmental, safety, and health problems being found at different DOE facilities? What are some of latest software products available for HP-3000 on-line applications? How can I meet my customer's ever-changing requirements? These and many other questions will be focused on within this review of the Environment, Safety, and Health (EH) Safety Representative Information System (SRIS) located on the Safety Performance Measurement System (SPMS). SPMS is a collection of automated environmental, safety, and health information modules for reference by DOE and DOE contractors. SPMS is operated by the Management Information Systems (MIS) Unit of the System Safety Development Center at EG&G Idaho, Inc.

In the following sections an overview of SRIS, an on-line system designed for the HP-3000, will be presented along with an analysis of design methods and software packages used to develop the system.

2.0 What is the EH Safety Representative Information System (SRIS)?

2.1 General Overview of SRIS

If your job requires keeping updated on environmental, safety, or health findings within DOE, SRIS and other modules on SPMS are invaluable tools for analysis. SRIS was developed to disseminate safety representative reports across the DOE community. Currently, safety representatives are located at the following locations:

1. Idaho Falls, Idaho
2. Oak Ridge, Tennessee
3. Richland, Washington
4. Golden, Colorado
5. Aiken, South Carolina

The EH Safety Representatives report directly to DOE Headquarters and perform continual inspections at their respective locations. Following their inspection, the safety representatives document their findings by entering them into the SRIS database. These findings include any environmental, safety, or health problems. When completed, the safety representative submits the report for review by the appropriate operations office. According to DOE guidelines, the operations office is given two full working days following the submittal of the safety representative's report to respond. After two full working days, the report is automatically released to the DOE community for review.

2.2 Capabilities of SRIS

SRIS is a responsive system providing keyword search, report generation, on-line data entry with a word processing environment, and personal computer (PC) data interface capabilities.

Searches can be performed to locate findings and responses on a variety of attributes (e.g., site, organization code, occurrence date of finding, keyword search on narrative text). After retrieving requested items the appropriate report (daily, weekly, monthly, or special) can be generated.

Safety representatives may enter their reports using a PC application or on-line data entry screens. This versatility allows users to select the software environment they feel most comfortable with for data entry. The PC application allows data entry using almost any familiar word processor and provides a window environment with easily accessible help screens. The HP-3000 also offers a word-processing environment and a help facility.

2.3 Access Restrictions

The Department of Energy (DOE) can be divided into three unique entities:

1. DOE contractors (e.g., EG&G Idaho, Inc., Westinghouse Idaho Nuclear Company, Inc.) - Companies who have received contracts from DOE to perform designated tasks such as operate DOE-owned facilities.

2. DOE Operations Offices (e.g., Idaho Operations, Richland Operations) - These regional DOE offices perform contract administration and oversight.

3. DOE Headquarters - Located in Washington D.C., this is the parent office of all DOE facilities.

SRIS is designed to allow only authorized DOE and contractor personnel to read, write, and respond to certain reports.

Users searching for report items are limited by the following "reading restrictions":

- Contractors may read all *released*[1] reports for their own facility.
- Operations office personnel may read all *submitted*[2] reports for their own site and all released reports.
- DOE Headquarters personnel may read all released reports.

EH Safety Representatives (who enter findings) and contractors and operations office personnel (who respond to findings from safety representatives) are limited by the following "writing restrictions":

- Contractors may respond to daily, weekly, or monthly items that are related to their own facility.
- Operations office personnel may respond to daily, weekly, or monthly submitted reports that are related to their operations office.
- Safety representatives may add to or update any of their site's daily, weekly, or special reports until the reports are released.

3.0 System Requirements of SRIS

Safety Representatives are located at various DOE facilities throughout the United States and enter daily, weekly, monthly, and special reports for DOE Headquarters review. This factor accounted for the most comprehensive and difficult requirement of SRIS. That is, provide a centralized, easily accessible, and timely reporting system. Due to the diversity of skills among Safety Representatives, "user-friendliness" was also of prime consideration. To provide an "easy to use" system several modes of data entry were developed. (Initially, time

---

[1] Released reports refer to all SPMS SRIS reports that have been "submitted" by the safety representative for more than two full working days.

[2] Submitted reports refer to all reports that have been completed by the safety representative.

restraints also made it imperative that a system be prepared quickly with available expertise and software. After initial development, enhancements and other modes of data entry were designed). These included data entry screens on the HP-3000, PC data entry screens with an "upload" facility to the HP-3000, and with the use of a "template", a user may create a file using almost any familiar word processor and then load the information to the HP-3000 database. For dissemination of reports, a centralized, easy to use, and fast means to retrieve reports was required. Security restraints were also of primary consideration for both data entry and retrieval.

4.0 Methods used to achieve requirements on the HP-3000

4.1 Data Entry Screens, Help Facility and Word Processing

To provide user-friendly data-entry screens on the HP-3000, HP's VPLUS utility was used for fast screen generation of data entry modules (see Figure 1 on the following page as an example VPLUS screen). To improve "ease of use" in the VPLUS data entry screens, software was reviewed to provide on-line helps. Of primary importance, the help screen software needed to provide an easy method of integration with existing VPLUS screen applications (since the majority of our data entry applications use VPLUS). "AUTO HELP" by PROBUS accomplished this task to best meet our needs. By pressing f6 the user can receive a general help information on the current data entry screen. Also by placing a question mark (?) in the field being questioned and pressing the numeric pad's <ENTER> key, help can be retrieved for the respective field. The help screen might appear as shown in Figure 2 on the following page.

Due to the combined factors that VPLUS does not include any word processing functionality and the extensive amount of narrative text that safety representatives enter, research was also performed to find an efficient means for entering narrative. It was determined that word processors with available "hooks" into an HP-3000 application are extremely rare products! Fortunately, however, one that met the requirements of SRIS was found. Minisoft's MiniWord and Toolkit provided a complete word processor for on-line applications. Commands are all performed through assigned function keys and/or control key sequences. A template of allowed functions was provided to each safety representative as a quick reference guide. Figure 3 shows an example of how the word processor appears to the users.

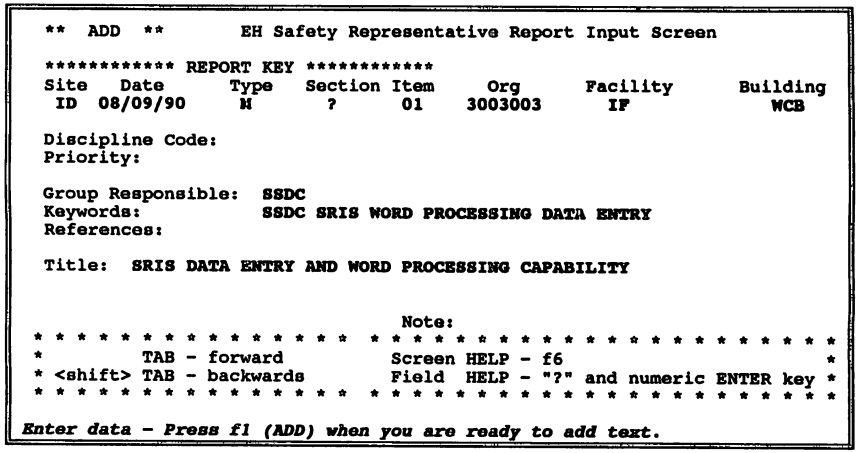


Figure 1. SRIS data entry screen.

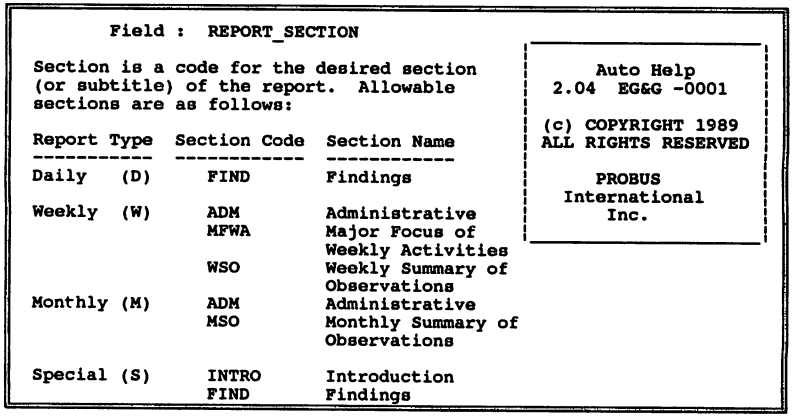


Figure 2. SRIS help screen.

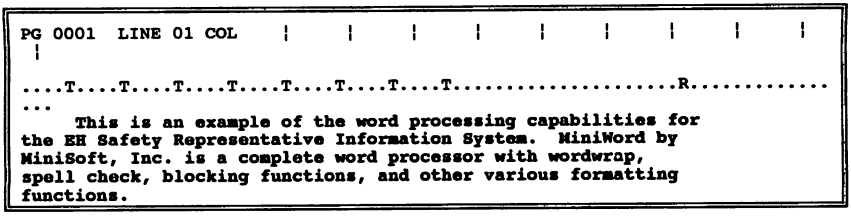


Figure 3. MiniWord word processor linked into VPLUS application.

4.2 Search and Retrieval

To meet the basic search and retrieval requirements for SRIS, in-house software (HP's Image, a database management system, and Omnidex by Dynamic Information Systems Corporation, a high-speed search indexing utility) and previously written search routines were utilized to provide responsive and accurate searches. Figure 4 shows an example of the search and retrieval capabilities within SRIS:

```
*****************************************************************
** Safety Representative Information System Search and Reports **
*****************************************************************

1. Site                    8. Priority (I,II,III)  15. Contractor Response
2. Report Date(yyyymmdd)   9. Group Responsible    16. Verification Narr.
3. Report Type (D,W,M,S)  10. Keywords             17. Add Date (yyyymmdd)
4. Organization Code      11. References           18. Create Initial Subset
5. Facility Acronym       12. Title                19. Reinitialize
6. Building               13. Report Narrative
7. Perf. Objective        14. Field Office Response

Type "HELP" for general info. or "HELP" and an item #, ie. "HELP 3".
Press 'RETURN' key only to end selection or enter field number(s):
1

Now enter your Site
For help on this field, type "HELP"
Press 'RETURN' only for previous prompt
ID

176 cases met the search requirements.


1. Site                    8. Priority (I,II,III)  15. Contractor Response
2. Report Date(yyyymmdd)   9. Group Responsible    16. Verification Narr.
3. Report Type (D,W,M,S)  10. Keywords             17. Add Date (yyyymmdd)
4. Organization Code      11. References           18. Create Initial Subset
5. Facility Acronym       12. Title                19. Reinitialize
6. Building               13. Report Narrative
7. Perf. Objective        14. Field Office Response

Type "HELP" for general info. or "HELP" and an item #, ie. "HELP 3".
Press 'RETURN' key only to end selection or enter field number(s):
13

Now enter your Report Narrative
For help on this field, type "HELP"
Press 'RETURN' only for previous prompt
fire,safety

14 cases met the search requirements.
```

Figure 4. Example of SRIS search and retrieval.

4.3 Report Generation

For report generation, a fourth generation report writer (QUIZ by Cognos) provided a quick, easy means to develop the necessary reports. Figure 5 shows an example of generating a report after performing the search shown in Figure 4.

```
 1. Safety Rep. Daily Finding Report  5. Safety Rep. Special Report
 2. Daily Report with Headings        6. List of Report Titles (by Site)
 3. Safety Rep. Weekly Report         7. List of Report Titles (alphabetic)
 4. Safety Rep. Monthly Report


For general help, type "HELP"
For help on any field, type "HELP", followed by
a number between 1 and  7
Press 'RETURN' key only to end selection criteria
Enter Report field choice number 1


            ****      Print DAILY Findings Report      ****
            *                                            *
            *                                            *
            ***   This is a 80 column portrait report   ***


Do you want your output on your terminal or
on the SSDC printer. (T/P/L) [T]   ?T

Do you desire a hardcopy (Y/N) [N] ?N
Do you want to pause after each screen (Y/N) [Y] ?Y


Printed from SPMS on 08/09/90                                    Page 1


                  EH SAFETY REPRESENTATIVE INFORMATION SYSTEM
                     Daily Report of Findings for 08/20/90

Site: XXX OPERATIONS


        Finding No: 01
          Priority: III

        Title: INSTALLATION OF DRIP TRAYS WHICH DID NOT RECEIVE
               SAFETY REVIEW

          Finding:

            During a tour on 8/17/90, several (8-10) plexiglass drip trays
            were observed in the overhead of the -13 foot elevation of the
            ZZZ facility.  The trays were apparently installed to
            prevent drips from acid and cadmium bearing system valves and
            flanges from falling to the floor.  Above the trays, were
            wet pipe fire protection sprinkler components, including spray
            heads.
```

Figure 5. Example of SRIS report generation.

5.0 Conclusions

SRIS utilizes a variety of software packages to provide a powerful system which can meet the changing requirements of the customer in a timely manner. Too often, system analysts try to meet their customer's requirements by using available in-house software and/or developing the system with the use of only one software development tool. This can create a system which may not fully meet user requirements. The EH Safety Representative Information System has made use of advanced technologies in HP-3000 application software, providing a comprehensive, easy to use, and maintainable system. As requirements and technologies change, so can SRIS!

The Omnidex Handbook:
Tips for Tuning Omnidex Performance

C. Shawn Morris
Dynamic Information Systems Corp.
5733 Central Avenue
Boulder, Colorado, 80301
(303) 444-4000

Introduction

Much has been written about IMAGE performance as it relates to the underlying IMAGE indexing structure. This information has helped users to identify real or potential performance problems, and adjust their data base administration practices to avoid problems and improve performance. However, little information is available to guide the user in diagnosing problems with alternate indexing products such as Omnidex.

This paper will provide the Omnidex user with specifics about the internals of Omnidex keyword indexes. Common problems and their underlying causes will be discussed as well. Along the way, remedies and recommendations will be provided as guidelines to help the Data Base Administrator maximize Omnidex index efficiency and throughput.

Omnidex Indexing Structure

In order to understand the overhead and possible pitfalls of Omnidex keys, one must first gain a basic understanding of the internal structure of Omnidex indexes. Before beginning that discussion, however, a few terms need to be defined.

Terms to Know

An **Omnidex keyword field** is an IMAGE field that has been designated as an Omnidex key at installation time. A **keyword** is a word found somewhere in the Omnidex keyword field. Keywords are delimited by spaces, special characters and field boundaries, and one Omnidex keyword field may contain several keywords.

A **record complex** is defined as a master record and it's associated detail records. An example of a record complex is a customer master record and all order records for that customer, or a batch header record, and all batch detail records entered for that batch. Another way to think of a record complex is a single, variable length record containing all data related to an entity.

The **Omnidex ID** is a double word integer value (e.g. IMAGE type I2) between 1 and 8,388,607 that uniquely identifies a record complex.

In practice, the Omnidex ID is either the IMAGE search item value of a record (if the search item is a double word integer), or a number that uniquely identifies the search item. This allows Omnidex to store double word integer references to search items rather than the search items themselves which can be as large as 64 words.

An IMAGE **domain** is defined by a master data set and it's associated detail sets. All data sets in the domain are related by a common field known as the IMAGE search item. Thus, a data set contains records, and a domain contains record complexes.

An **Omnidex domain** is defined by the data sets of an IMAGE domain that contain Omnidex keyword fields. The common search item for those sets is called the **Omnidex search item.**

To summarize, an Omnidex ID either <u>is</u> a search item, or has a one-to-one relationship with a search item. The search item associated with an Omnidex ID is common to the master and possibly one or more detail records in a record complex. Because Omnidex IDs are used to identify the record complex that a keyword belongs too, it is helpful to examine first how the Omnidex ID is stored and how it is assigned.

The Inverted File Structure

When building the indexes for an Omnidex keyword field, all records in the Omnidex domain are searched, and words in the Omnidex keyword field (delimited by spaces and special characters) are parsed out of the field and copied to a file. With each keyword, an Omnidex ID is recorded which identifies the search item of the record complex from which the word came. In the simplest case, the Omnidex ID is also the IMAGE search item.

After all keywords are extracted, the unload file is sorted by keyword and Omnidex ID. All IDs are then loaded into a master and detail data set, with each unique keyword becoming a search item for a chain of Omnidex IDs. This structure is called an **inverted file index** because the data values are used to retrieve search items, rather than using the search items to retrieve the data.

To illustrate the indexes that result from this process, assume that 3 records exist in a CUSTOMERS master data set, as shown below:

CUSTOMER-NO:	2	7	9
CUSTOMER-NAME:	Joe Smith	Joe Jones	Sloppy Joe

If CUSTOMER-NAME is designated as an Omnidex keyword field, the resulting inverted index would look like this:

JOE	JONES	SMITH	SLOPPY
2	7	2	9
7			
9			

This list allows Omnidex to very quickly determine how many records contain a keyword such as "JOE" and to determine the unique search item for each record.

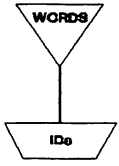
Implementing the Inverted File Index

The Omnidex indexing method is implemented using IMAGE data sets and is maintained using only IMAGE intrinsics. As a result, Omnidex index maintenance participates in all IMAGE activities, including shared locking, set and item security, and logging. An approximation of the actual IMAGE implementation is discussed next. While not exact in every detail, the discussion is adequate to understand the performance issues involved with Omnidex indexing.

As stated earlier, the main Omnidex index sets consist of a master and detail data set. The master record holds the keyword, and the detail set holds the Omnidex IDs of the records that contain the keyword. In graphic form, the structure would look like this:

Master data set with a search item of ODX'WORD.

Detail data set chained by ODX'WORD that holds Omnidex IDs.



For simplicity, assume that each detail record can hold up to three Omnidex IDs. Leaving a space in each record for insertion of new Omnidex IDs, the physical records to track the keyword JOE would look like Figure 1.

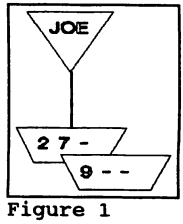


Figure 1

To retrieve all records containing the keyword JOE, Omnidex simply performs an IMAGE DBFIND on ODX'WORD with an argument of "JOE", followed by a chained read of the detail data set. Omnidex then returns the search items to the application program. For each search item returned, IMAGE reads can be executed on the appropriate master or detail data sets to retrieve the corresponding records.

Now, let's take the example a bit further and add a customer to the data set, then consider the IMAGE transactions required to maintain the list for JOE in "real time".[1] Here's the customer record:

CUSTOMER-NO: 10
CUSTOMER-NAME: Joe

Before adding the record, Omnidex parses the keyword out of the field CUSTOMER-NAME, and establishes the head of the chain containing IDs for the keyword JOE. Backward chained reads are then performed until the proper record is found in which to insert the new Omnidex ID. In this case, only one backward read is needed to find where to put an Omnidex ID of 10. Finally, DBUPDATE is called to insert the ID in the ID chain. The resulting Omnidex ID chain looks like Figure 2.

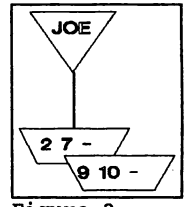


Figure 2

Estimating Omnidex I/O

This example illustrates the most favorable situation for Omnidex indexing;

> **If the Omnidex ID being inserted is always greater than the largest Omnidex ID in the chain, only 1 read to disc and 3 writes to disc are needed to index a keyword.**

Adding 3 initial reads, the estimate for indexing a new Omnidex record is then:

#I/Os = 3 + 4 * #keywords

where #I/Os is the total number of reads or writes required to index the entire record, and #keywords is the number of words (separated by spaces and special characters) in the record to be indexed. If only 1 keyword occurs in each Omnidex keyword field, then the estimate is:

#I/Os = 3 + 4 * #keys

where #keys is the number of Omnidex keyword fields installed on the data set.

This formula should look familiar; it is also the estimate for I/O required to update an IMAGE path. With IMAGE, however, the occurrence of secondaries can greatly increase the amount of work

[1] For the purposes of this paper, the term "real time" is used to indicate that the Omnidex indexes are updated simultaneously with the add, update or delete of a record.

required to maintain the IMAGE path. With Omnidex keys, a similar affliction can occur.

The Perilous Packed Pointer Predicament

Omnidex IDs must be inserted in an ID chain such that they remain in sorted order. This is no problem when an empty slot is available. Yet, when an index record is completely "packed" with IDs, all IDs greater than the one being inserted must be shifted by one along the ID chain. Consequently, the highest ID in the insertion record must in turn be inserted in the next record in the chain. If many consecutive records are packed, then many repetitions of this "ripple effect" can occur. I call this phenomenon the "perilous packed pointer predicament".

To illustrate, let's go back to our original example, and add three records who's Omnidex IDs are not the largest values in the chain. Here are the records:

CUSTOMER-NO:	4	5	3
CUSTOMER-NAME:	Good Joe	Joe Bob Smith	Joe Bob Jones

Conceptually, the new inverted index looks like this:

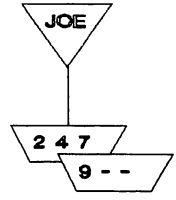
BOB	GOOD	JOE	JONES	SMITH	SLOPPY
3	4	2	3	2	9
5		3	7	5	
		4			
		5			
		7			
		9			

Remember that the Omnidex IDs for the records to which each keyword belongs are kept in sorted order. This allows for easy comparison of lists and is the basis for the ability of Omnidex to perform "AND", "OR" and "NOT" logic. The IMAGE procedure calls required to add references to records 4, 5 and 3 (in that order) for the keyword JOE would proceed as follows:

For record 4:

Perform DBFIND mode 1 on the ODX'WORD search item using "JOE" as the argument. Read the chain backwards until the proper record is located.

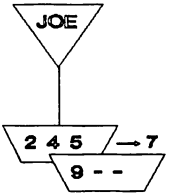
Call DBUPDATE to update the record so that it contains the new Omnidex ID in the proper order.



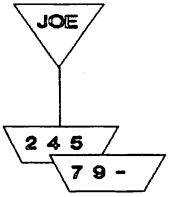
For record 5:

Perform DBFIND mode 1 on the ODX'WORD search item using "JOE" as the argument. Read the chain backwards until the proper record is located.

Call DBUPDATE to update the record so that it contains the new Omnidex ID in the proper order.



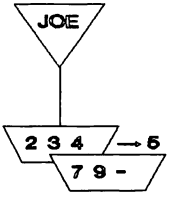
"Push" the ID for record 7 into the next index record using DBGET mode 5, followed by DBUPDATE.



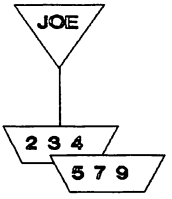
For record 3:

Perform DBFIND mode 1 on the ODX'WORD search item using "JOE" as the argument. Read the chain backwards until the proper record is located.

Call DBUPDATE to update the record so that it contains the new Omnidex ID in the proper order.



Push the Omnidex ID for record 5 into the next record using DBGET mode 5, followed by DBUPDATE.



By now, you should notice the following;

To insert an Omnidex ID into a packed record takes an extra forward chained read and update (2 I/Os) to push an ID to the

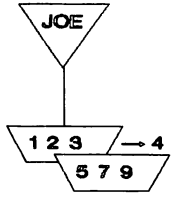
next record.

Just to make sure this concept hits home, let's insert another record for customer number 1.

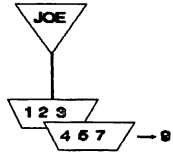
CUSTOMER-NO: 1
CUSTOMER-NAME: Joe Lunch Pail

Perform DBFIND mode 1 on the ODX'WORD search item using "JOE" as the argument. Read the chain backwards until the proper record is located.

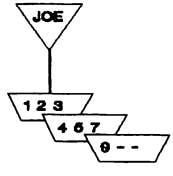
Call DBUPDATE to update the record so that it contains the new Omnidex ID in the proper order.



Push the Omnidex ID for record 4 into the next index record using DBGET mode 5, followed by DBUPDATE.



Push the Omnidex ID for record 9 into a new index record using a call to DBPUT.



To extend the observation from before;

> **When inserting an Omnidex ID into a packed index record, two additional disc I/Os are required for EACH CONSECUTIVE PACKED RECORD.**

That's the perilous packed pointer predicament. It occurs when Omnidex "ripples" all IDs forward in an ID chain until a record is found with an open slot for an Omnidex ID. True Omnidex indexing allows for up to 64 Omnidex IDs per index record and 4 empty slots, making it much less likely that this will occur. Over time on an

active data base, however, the conditions for this phenomenon can develop.

Similar to Migrating Secondaries

This situation is very similar to the problem of migrating secondaries, where IMAGE "bumps" a record with a secondary search item to make room for a primary, and attempts to relocate the record. IMAGE may read through several IMAGE blocks before finding an empty address for the record with the secondary search item.

Omnidex Overhead Summarized

To summarize the previous discussion, the overhead required to index a keyword depends on where in the ID chain that the ID must be inserted. If the ID is not inserted at the end of the ID chain, any extra work required depends on the availability of an empty slot in the index record. If no empty slots exist, the overhead increases by two I/Os for every packed index record after the one in which the Omnidex ID is initially inserted. Therefore, it is helpful to know how Omnidex IDs are assigned to determine if there is potential for performance problems.

Assignment of the Omnidex ID

When a new record is introduced through a DBPUT call, the Omnidex ID that is assigned is usually greater than any previously used. Consequently, maintenance of the chains of Omnidex IDs is done at the end of the index chain, and overhead is minimized.

One instance where insertion of Omnidex IDs occurs somewhere in the middle of an ID chain is after records are deleted. Whenever an Omnidex record complex is removed, the Omnidex ID that was assigned to the record complex is saved for later use on the "free ID list". Omnidex IDs are then re-used on a last in, first out (LIFO) basis. The free ID list is very similar in concept to the "delete chain" that IMAGE uses to track and re-use detail data set addresses from which records are deleted.

The other instance where Omnidex IDs are inserted is when records are updated, or when detail records containing Omnidex keyword fields are added to an existing record complex. In both instances, keywords in Omnidex keyword fields are assigned to an Omnidex ID that was previously allocated to a record complex. The value of the ID, and where it might be inserted in an index chain is often unpredictable.

Finally, a Useful Conclusion!

While little has been put forth to this point as to what actions

can be taken to minimize Omnidex overhead, enough information is available to establish our first guideline:

> **If records are not deleted, or deletes are rare, overhead for adding new records is minimal.**

If a data set is fairly static, at least from the standpoint of deletes and updates, then most of the IMAGE transactions required to maintain Omnidex ID chains occurs at the end of the chains.

This is not uncommon for reference files such as customer masters, catalog masters, vendor masters and history files. In many cases, records in these files are rarely modified after their initial load. As a result, these kinds of data sets are great candidates for Omnidex indexing, because the cost to maintain them is low, while the benefit from increased retrieval capability is high.

Minimizing Omnidex Overhead

In many cases a data set will not be static, receiving frequent add, delete and update transactions. Consequently, Omnidex indexing can add significant overhead to these transactions. Just as one would hesitate to add an IMAGE path to a detail data set unless it served some useful purpose, one must weigh the usefulness of a new Omnidex key against the overhead it adds. Fortunately, Omnidex keys have many more useful purposes than IMAGE keys!

Now that the pertinent features of the Omnidex indexing mechanisms are revealed, it's time to look at ways of minimizing the overhead associated with an Omnidex keyword field. The available methods reflect two strategies:

- Defer indexing to periods of low activity

- Manage indexes for real time throughput

Deferring Index Transactions

Several methods exist to avoid the extra IMAGE transactions required to index keywords in real time. Usually, this involves some method of preventing Omnidex from indexing keywords, followed by a complete or partial rebuilding of indexes at some strategic time.

Disabling Real Time Indexing

The easiest way to avoid the overhead of indexing keywords is to completely bypass the mechanisms that cause indexes to be maintained in real time. If it is not imperative that records be available for retrieval by Omnidex keys immediately after they are added, disabling real time indexing eliminates the overhead

associated with Omnidex indexing.[2]

For example, if users wish to retrieve GL transactions in an accounting data base, it's conceivable that only those transactions for closed months are of value for retrieval. In this case, one may wish to index all records after the period closing, and bypass real time indexing during the day-to-day transaction input. When the new period is closed, all records can be indexed again. The ways to disable real time indexing are straightforward.

If a program calls the Omnidex IMS intrinsics DBIPUT, DBIDELETE and DBIUPDATE to write to a data base, these intrinsics can be called using "IMAGE-only" mode[3]. This causes Omnidex to bypass the indexing of keywords at the time of the transaction.

Another option, for programs that do not use the Omnidex "DBI" calls to update the data base, is to simply decline to implement the call conversion feature of Omnidex. The call conversion feature traps calls to DBPUT, DBDELETE and DBUPDATE, and calls DBIPUT, DBIDELETE or DBIUPDATE on behalf of the program. If this mechanism is not put in place, no indexing of records occurs in real time.

After a period of activity, usually every night, or after a milestone such as the close of an accounting period, a complete reindexing of the Omnidex keys is performed. During this time, no other processes are allowed to have the data base open. After the process completes, all keyword fields are indexed and available for Omnidex retrieval.

The BI Field Option

The BI or "Batch Indexing" option is assigned at the time that a field is designated as an Omnidex keyword field. While disabling real time indexing bypasses indexing for all key fields, the BI option allows the user to choose specific keys for which to disregard indexing.

Assuming that the mechanisms for real time indexing are not disabled, Omnidex keyword fields possessing the BI option are excluded from indexing at the time of a put, delete or update. All keyword fields not installed with this option are indexed in real time. As with the "disabling" alternative described earlier, all Omnidex keyword fields are reindexed in a period of low activity to

---

[2] Note that this alternative is best suited when most of the unindexed activity comes from new records. When IMAGE record deletes or updates occur without real time index maintenance, inconsistencies between the indexes and the data records may result.

[3] If the data base is opened with shared, modify access (DBIOPEN mode 1), then an IMAGE-only mode would be used with DBILOCK as well.

reflect the most recent transactions.

Again using the example of accounting transactions, you may wish to designate several fields as Omnidex keyword fields, and add the BI option to all Omnidex keyword fields except the posting status. If a mechanism for real time indexing is in place, the posting status field is indexed immediately after a record is added, while other Omnidex keyword fields are ignored. This technique permits quick retrievals using the posting status keyword field (perhaps to retrieve unposted records), while other Omnidex retrievals would be permitted only on records for closed periods.

Advantages/Disadvantages

The advantage of the above techniques is that index updates are ignored either completely or for selected keys. Consequently, no I/O is generated to maintain the Omnidex indexes. The drawbacks with these methods are that changes to keyword fields are not immediately reflected in the indexes, and a complete reindexing of all keys, whether they were indexed in real time or not, is required to make the indexes consistent with the data records. A technique called "Deferred Update" can give the same benefits, but requires a less intensive process to bring the indexes up-to-date.

Deferred Update Indexing

The Deferred Update process consists of two steps. In the first step, a flag is set, either at data base open time, or separately with the ODXUTIL utility program, which instructs Omnidex to defer, rather than ignore all indexing activity. For each put, delete or update, all keywords and Omnidex IDs to be added to or deleted from the indexes are written to files. Since the records are small and the blocks large, writes to the files consume very little I/O overhead. As a result, the time required to add 1000 records with indexing disabled is essentially the same as the time required to add 1000 records with indexing deferred.

After all records are added, the ODXUTIL indexing program is used to update the deferred keywords. In this step, the deferred keyword files are first sorted in keyword and ID order. Then the indexed keywords are unloaded from the Omnidex index data sets and sorted in the same order. Finally, all keywords are merged back into the indexes, with common keyword deletes and adds canceling each other out. This process takes only slightly longer than a simple primary path reload of the Omnidex detail index data set.

Advantages/Disadvantages of Deferred Update

The advantage of the Deferred Update process is that the sort/merge procedure takes considerably less time than an exhaustive read and reindex of the entire data set. This follows from the fact that

the index data sets are considerably smaller and more compact than the data files that they reference.

Of course, nothing is free, and there are a couple of drawbacks. First, when a process is performing IMAGE updates in deferred update mode, it must have exclusive write access to the data base (DBOPEN modes 3 or 4). While other processes may have the data base open for reading, no other processes may write to the data base during the first step of the Deferred Update process[4]. During the ODXUTIL update step, exclusive access is required to the data base, and no other processes may have the data base open.

The other drawback to this method is that a Deferred Update process may be performed on only one Omnidex domain at a time. While programs may be run repeatedly in deferred update mode to update the same domain, the deferred activity must be updated (via the second step of Deferred Update) before any other domains may be updated, or before the data base can be opened in shared, read/write mode.

For these reasons, deferred update is best suited for speeding up nightly batch updates that normally have exclusive access to the data base. It will almost always be faster than real time indexing of each transaction (unless a small percentage are updated), and will usually beat a complete reindexing as well (unless a very large percent of records are updated).

Managing Indexes for Real Time Throughput

All of the above described methods use techniques that completely avoid the overhead of real time indexing. They each have the added advantage of leaving the indexes "well organized" after the process is finished, whereas real time indexing may gradually proceed towards packing of index records. However, real time indexing is often desired to get immediate retrieval capability on the Omnidex keyword fields. It is here that some knowledge of the index internals, and a few techniques, can help to keep Omnidex indexing running smoothly.

The Excluded Words List

The Excluded Words List represents the easiest, most effective, yet possibly the most neglected technique for improving real time throughput. It is easily built, easily installed, improves both real time and batch indexing performance, and its concept is simple. In short, the Excluded Words List contains words that should not be added to the Omnidex indexes. Whenever a reindexing is performed, or a record is added or updated, keywords that are

---

[4] Concurrent read access is allowed only with DBOPEN mode 4.

parsed from the Omnidex key fields are checked to see if they exist in the Excluded Words List. If so, they are not indexed, reducing disc accesses and increasing overall throughput.

Typically, the excluded words list contains "noise" words that are of little value for retrieval purposes like "and", "the" and "of". For free form keyword fields, a comprehensive Excluded Words List can save a great deal of the I/O required to index each record. Take for example a field in a customer record that contains customer names. Each name would consist of a first name, last name, middle name or initial, and a salutation such as Mr., Mrs., or Ms., for a total of four keywords per key. By adding the words MR, MRS and MS to the Excluded Words List, only three of the four words in each name field must be indexed - an I/O (and time) savings of 25 percent!

The excluded words list should also contain keywords that occur in a majority of the record complexes in an Omnidex domain. For example, if most of the record complexes in a domain contain a company code of "01", the Omnidex ID chain associated with the "01" keyword may be very long. Consequently, many disc access may be required for searches of the chain, causing new ID insertion or Omnidex retrievals to take a prohibitive amount of time.

It is easy to determine the likely candidates for excluded words. A good starting point is the default excluded words file, XCLUDES.PUB.DISC, provided with the Omnidex software. Then, after indexing all keyword fields (or a representative subset), use the keyword lookup feature of the DATADEX inquiry program to interrogate the indexes for each key[5].

From the list of words that were indexed, identify obvious noise words or words that occur in more than 1/2 of the record complexes in the Omnidex domain. Add these words using the editor of your choice to your file containing excluded words. Load the file using the ODXUTIL "XCLUDE" command, and reindex all keys.

Periodic Reindexing

If the Excluded Words List is the most neglected method of improving throughput, then Periodic Reindexing is the second most neglected, but effective method of improving throughput. As indicated earlier, records in an Omnidex ID chain normally contain 64 slots each for Omnidex IDs. After a complete reindexing, 4 of those slots are empty, reserved as "pad" space for insertion of new IDs. As a result, insertion of new Omnidex IDs can be accomplished without the "ripple effect" that sometimes increases Omnidex

[5] A DATADEX retrieval on an Omnidex keyword field using the argument "!0:2" will list all keywords indexed from an ASCII field. Refer to the DATADEX reference manual for more information on the keyword lookup feature.

overhead.

Another benefit of Periodic Reindexing is that the resulting Omnidex ID chains are sorted in primary path order on the ODX'WORD keyword path. This is advantageous for Omnidex indexing, as it would be for any detail data set that is reloaded on a commonly used IMAGE path. Since all records for a given keyword are physically contiguous in the Omnidex detail index set, several chained reads can be accomplished without generating another read to disc. Since there are 7 records per block in the Omnidex detail index set, Omnidex can scan as many as 448 Omnidex IDs per I/O when searching for the place to insert a new Omnidex ID[6]. For the same reasons, Periodic Reindexing enhances the performance of Omnidex keyword retrievals as well.

The benefits of a reloaded detail index set can be obtained without completely reindexing by simply reloading the Omnidex detail index data set using any popular IMAGE data base utility. A program called ODXMGR is provided with every Omnidex software tape which gives the capability to perform capacity changes and reloads on Omnidex index data sets. If you have a regular schedule for reloading detail data sets, include your Omnidex detail index sets in the rotation. You can identify the sets by first obtaining a list of all data sets in the data base. Every data set beginning with the characters "XODX'" is an Omnidex detail index set.

Increasing Index Pad Space

As stated earlier, 4 slots are left in each of the Omnidex detail index records for insertion of new Omnidex IDs. In a dynamic data base, however, these slots will be systematically taken, creating increasing numbers of packed records. As the occurrence of packed index records increases, forming chains of contiguous packed records, Omnidex overhead problems begin to develop.

While reindexing the Omnidex keyword fields will alleviate the packing problem, and option in ODXUTIL utility program can greatly extend the time it takes for index records to become packed with Omnidex IDs. This in turn will reduce the frequency with which you should periodically reindex.

By using a command called "SET PAD", ODXUTIL permits the user to increase the "pad space" to a maximum of 16 slots for insertion of new IDs. As a result, four times as many IDs can be inserted into a given index record before all slots are used up. The SET PAD option must be used every time you reindex with ODXUTIL if you want more empty slots than the default of 4.

---

[6] A maximum of 64 Omnidex IDs per record, multiplied by 7 records per IMAGE block, yields 448 IDs per disc I/O when the Omnidex detail is reloaded in primary path order.

The tradeoff for increasing the pad space is that it takes more records (and possibly a higher capacity and more disc space) in the Omnidex index detail set to hold the same number of Omnidex IDs. For a pad setting of 16, expect the number of Omnidex detail index records to increase by around 15%.

Conclusions

This paper focused on a conceptual model of the "real time" updating of Omnidex indexes. The model aids in presenting the kinds of performance problems that can occur when maintaining the detail data set that contains the inverted file index.

In general, performance problems begin when the IMAGE detail records that contain the Omnidex ID references become "packed". Consequently, inserting an ID into the chain causes a "ripple effect" that adds 2 I/Os for every consecutive "packed" index record (moving forward from the first record) in the chain.

Performance can best be enhanced by not indexing in real time at all. The indexing can be completed nights or weekends, when there are CPU cycles to spare. This option is best suited for static data bases, because new updates are not reflected in the Omnidex indexes until the next reindex process is performed.

If real time indexing is required, then performance can be greatly improved by using the Excluded Words List. Periodically rebuilding the indexes (weekly if possible, monthly if not) reallocates slots for insertion of new IDs and reloads the Omnidex detail index set. Increasing the number of empty slots using the ODXUTIL "SET PAD" option allocates many more slots for insertion, reducing the need for periodic reindexing.

The suggestions contained herein will give the Omnidex user an assortment of weapons with which to fight the perpetual performance battle. More importantly, however, it is my hope that a glimpse of the Omnidex internals, accompanied with the explanation of when and how each suggestion improves performance, will permit the data base administrator to tailor a mixture of these procedures that best fits his or her unique data base environment.

Tradition vs. Transcendence in Software Engineering

Paper No. 3211

Natalie M. Minenko
Technical Staff

Oracle Corporation

HP Products Division
400 Oracle Parkway
MD: 4OP-11
Redwood Shores, CA 94065
(415) 506-7000

Abstract

When developing a new software product for the marketplace, several questions must be asked early in the design stage, including, "Who are my users?", "How will my software impact my users?", "How easy will it be for them to learn and use this product?", and "How will this product make my users more efficient, more productive, and more profitable?" This paper focuses on the dichotomy of two software engineering principles that are essential to user centered design: tradition versus transcendence.

Tradition is the principle which measures how much of the user's outside knowledge and previous experience can be applied to the new product. Transcendence is the quality which indicates how progressive and innovative a product is, whether it is a bold advance which will revolutionize the way users think and do their work or something that is not so much a break from tradition.

This paper will examine these two principles of software engineering and show how they are essential in answering the above questions and creating a usable and marketable product. Some examples will be drawn from case studies using Hewlett-Packard's V Plus and Oracle Corporation's SQL Forms in the MPE XL environment. These two software packages are both commercially available from Hewlett-Packard and Oracle Corporation, respectively.

If the city of San Diego suddenly passed an ordinance to reverse all of its street lights, such that green meant "stop" and red meant "go," the results would immediately be felt in hospitals and insurance claim offices citywide. Even after local residents got used the new system, accidents would still occur due to the unsuspecting tourist who missed the signs proclaiming, "In this city, green means stop and red means go." Rescinding the law could only make matters worse in the short term--motorists and pedestrians would be so confused that they might start avoiding San Diego altogether rather than take their chances at getting hit.

While this is a rather extreme example, it serves to illustrate the point that successful software products must provide the user with a familiar environment to work in for maximum productivity. If a user can apply previous knowledge gained from the outside world to a product, that guarantees a shorter learning curve than if he or she had to learn all of the product methodology from scratch. As Shneiderman offers in his guidelines for form fill-in design, "If `Address` were replaced by `Domicile`, many users would be uncertain or anxious about what to do." "Tradition" is the principle which measures how much of the user's outside knowledge and previous experience can be applied to the new product.

Providing users with familiar environments is one of the reasons for Oracle Corporation's software engineering strategy. Oracle Corporation's HP Products Division is solely responsible for modifying generic "base code" to run on an HP 3000 MPE XL system. It is necessary that the finished product has a similar look-and-feel to what MPE XL users have experienced in the past, from other companies as well as from previous ORACLE software releases. ORACLE for MPE XL products cannot, for example, allow file name extensions as the DOS and UNIX environments do. They must support the eight softkeys found on HP terminals whenever possible. And even though developers in Oracle's Macintosh group are required to use the desktop metaphor and the Macintosh Graphical User Interface, the HP Products Division cannot expect its customers to point and click in order to use ORACLE until the NewWave interface is supported. These are all examples of the constraints of tradition in software engineering.

Yet without some changes and adaptations, software would not evolve and improve over time. Sometimes, just a small feature is

needed to make a big difference in a product. Function key support that eliminates repetitive keystrokes might fall into this category. At other times, a complete reworking of the design is needed to achieve the desired result. This is often the case when the logical flow of a process must be redefined to maximize the user's productivity. "Transcendence" is the quality that indicates how progressive and innovative a product is, whether it is a bold advance which will revolutionize the way users think and do their work or something that is not so much a break from tradition. It is a quality that must be exploited carefully, so that users can see the connection between their current approach to the task at hand and the methodologies used in your product. Too much innovation, and your users won't be able to see how your 23rd century product will help to solve their problems today.

Having defined tradition and transcendence in the software development context, we proceed to examine user centered design and the questions that it raises. "Who are my users?", "How will my software impact my users?", "How easy will it be for them to learn and use this product?", and "How will this product make my users more efficient, more productive, and more profitable?" are the basic questions that must be asked early in the design phase of a product. When considering each question, the engineer must weigh the influences of tradition and transcendence to determine the overall design.

A common pitfall of designers in all professions is failing to identify the user's needs and the design issues they raise, then designing to meet those needs and solve the user's problem. Too often in their haste to be innovative, engineers can architect a product that either fails to solve the user's problem because (1) the parameters of the problem were misunderstood, or (2) the result embodies a great solution to a problem that the user never had in the first place. A classic example of failure to ask "Who are my users?" comes from architecture and the ill-fated Pruitt-Igoe low income housing complex. [Bannon 28] Built in St. Louis in the early 1950's, it consisted of large apartment complexes surrounded by open spaces while bypassing traditional streets, gardens, and semi-private spaces. Although it won an award from the American Institute of Architects and was notable in that "the intelligent planning of abstract space was to promote healthy behavior" [Jencks 9], the design was totally inappropriate for the occupants. The

architect neglected the fact that most of the occupants did not have prior experience living in a densely packed community, nor did he incorporate many places for traditional social activity in his design. In his effort to provide an innovative housing structure, the architect failed to consider the parameters of the design problem and meet his users' needs. The end result is that the complex became the site of vandalism, drug abuse, and crime, and was demolished within twenty years.

The question "How will my software impact my users?" is overlooked far too often at many development sites. Whether they intend to or not, designers impose their own values and expectations on end users to varying degrees. The Pruitt-Igoe housing project is an example of an intentional attempt at behavior modification that failed miserably. Word processing software provides an example of how computerization has unintentionally modified some people's methodology for document production. In the days when a writer had only a typewriter and a red pen to create his or her works, a common scenario consisted of 3 stages. First the writer would jot down his or her initial ideas in free form on a page. Once these ideas jelled, the writer would then type out one or two drafts of the document in a rough form. During the final part of this process, the writer usually typed very carefully, as each revision at this stage was potentially the final copy. The large amount of retyping required to make any major changes was a big disincentive for the writer to change the structure of the document at this point. Today, word processing software has virtually eliminated the third stage in this process, as well as combined parts of the first and second stages for maximum productivity. The result is that the writer is much more efficient at producing a document and has maximized his or her creative potential. Because of its total flexibility, word processing software has fostered a generation of writers who feel comfortable putting their initial thoughts up on the screen and making changes up through the final printing.

Along the same lines, the prudent software designer will consider the product's impact on the end users in terms of the learning curve for the product. He or she must keep in mind scenarios for the beginning, intermediate, and advanced user, and design accordingly. When making design decisions, the software developer must anticipate where users might be overly frustrated and where they would be bored stiff. It is helpful to ask, "Is the

logical flow of my program clear? Should I shorten the commands needed by the experienced user?" when critiquing the software design.

Often software developers focus on the steps necessary to learn and use a new product, but fail to ask "How easy will it be to learn this product?" and as a corollary, "How can I make the task of learning simpler?" An example of this comes from user interface design. The traditional user interface is a command line with a list of somewhat cryptic commands. Through trial and error, and a host of documentation manuals, users eventually become comfortable with the syntax and commands and learn to use the machine. With the advent of real time graphics came Graphical User Interfaces (GUIs) and a new look for computing. Through the use of metaphors, GUI architects can capitalize on the user's knowledge of the real world. For example, objects in the Macintosh GUI behave in a traditional sense: folders are where one stores files, and the trash can is where one throws them away. Objects in the trash can stay there until the user empties the trash. The transcendence that made this object oriented interface revolutionary was the emphasis on "see and point" to manipulate objects rather than "remember and type." [Apple Computer Corp. 4] With just a few tips to get one started, the new user can become proficient with a GUI very quickly. Very often, as is the case in this example, simplifying the user's task of learning a new operating system required a total rethinking of its design, as opposed to merely eliminating a few keystrokes.

Ease of learning and increased user productivity are two reasons Oracle Corporation strives for a common look and feel among its products while maintaining a native look-and-feel for each individual platform. Current development efforts in the HP Products Division are focusing on programmable softkeys for HP terminals and workstations. Although the varied functionality of ORACLE tools prevents 100% correspondence for each key in every product, similar functionalities will have similar key mappings in each ORACLE tool. Additionally, the mechanism to allow end user customization of the eight softkeys will become standard throughout Oracle's HP MPE XL product line. These two features promote common look-and-feel among products. While our default configuration will ensure a native MPE XL look-and-feel, the fact that each user will be able to customize the softkeys ensures that ORACLE

products will be flexible enough to provide an environment that is familiar to the user.

An additional example of common look-and-feel among MPE XL products is evident in standardized installation procedures. All products are now shipped with two installation scripts. TAPEINST is the script which will restore files from a distribution tape into the ORACLE customer's MPE XL software account. Before completion, TAPEINST instructs the customer to run the product installation script which creates the necessary users and tables in the database account and completes the product installation. Naming convention dictates that this script will begin with alpha characters that identify the product and end in -INST. Since all ORACLE for MPE XL tools follow this convention, even if a product arrives without documentation, the customer will know how to install it on his or her system.

Finally, the most important question to answer is, "How will this product make my users more efficient, more productive, and more profitable?" This is where transcendence becomes a key issue, because any product that does not help end users accomplish their tasks better than they did before does not have much potential in the marketplace. The successful software product will be flexible enough to accommodate users with multiple levels of expertise, provide expanded and enhanced capabilities with minimum of user effort, and remain intuitive enough so that the user knows what to do next. Answers to this question usually become key selling points for the product.

Oracle Corporation resolves this issue by designing a tightly integrated product line and maintaining an architecture that is highly portable across different software platforms. This strategy not only allows maximum efficiency for ORACLE development efforts but minimizes the learning curve and increases productivity for end users. This is not to say that ORACLE software is stagnant, however. New technology is constantly being investigated, developed, and implemented in new product releases and revisions.

Examples of transcendence in software engineering can be seen in the SQL*Forms product. When SQL*Forms was first introduced in the MPE XL marketplace, users immediately realized significant gains in efficiency. With the 4th-generation environment that SQL*Forms provides, users could build a powerful form-based application in a

matter of minutes, because SQL*Forms takes care of the low level details of implementation and access to the form. A comparable application could take days to implement with Hewlett-Packard's V Plus: since this is a 3rd-generation tool, the form structure is defined, but the user must code all routines needed to access the form. Although it took some time for users to switch from a 3rd-generation language to a 4th-generation development environment, the resulting increase in productivity, both in terms of development time and application portability, more than made up for the small amount of time spent learning the new product. SQL*Forms V3.0 incorporated additional innovations into the product. In previous versions, the use of macros left users clamoring for a means of procedural control in their applications. Development responded by introducing triggers which utilized PL/SQL code. Although this meant that many users now needed to learn PL/SQL, the tradeoff for streamlined, easy to create applications was acceptable.

When developing new software for the marketplace, it is important to remember the concept of tradition vs. transcendence and maintain each quality in proper proportion. A user centered approach to design begins by posing relevant questions to identify end users and their needs. In this method, it is essential that the product designer determine what traditional context end users are familiar with before deciding how much innovation or "transcendence" to introduce in a new product. For maximum benefit, these issues must be considered early and often during the software design cycle.

Works Cited

Apple Computer Corp. *Apple Interface Guidelines.* Chapter 1, Philosophy

Bannon, Liam J. "Issues in Design: Some Notes", Chapter 2, *User Centered System Design.* Ed. Donald A. Norman and Stephen W. Draper. New Jersey: Lawrence Erlbaum Associates, Inc. 1986

Jencks, C. *The language of post modern architecture.* New York: Rizoli. 1984

Shneiderman, Ben. *Designing the Human Interface.* Chapter 2, Theories, Principles, and Guidelines. p. 41-80

The author wishes to thank Prof. Terry Winograd, Department of Computer Science, Stanford University, for introducing the concept of Tradition vs. Transcendence.

Tradition

- How much of the user's outside knowledge and previous experience can be applied to the new product?

Transcendence

- How progressive and innovative is the product?
- Bold revolutionary advance?
- Or not?

Four Essential Questions

- Who are my users?
- How will my software impact my users?
- How easy will it be for them to learn and use this product?
- How will this product make my users more efficient, more productive, and more profitable?

Paper #3212:

MPE V/E FORTRAN: The Internals of Alternate Return Paths

Craig Nickerson
United Electric Controls Co.
P.O. Box 9143
Watertown, MA 02172-9143
U.S.A.
Tel. (617) 926-1000

Introduction

As a systems/applications programmer, I have worked extensively with FORTRAN 66 (FORTRAN/3000) under MPE since our company first acquired an HP3000 in 1982. FORTRAN 77 was added to our system when a major upgrade in our ASK manufacturing software was released in that language. We never implemented this upgrade because of the extent of our own modifications and enhancements to the earlier FORTRAN 66 version; however, I found that in many situations, FORTRAN 77 made program coding and structuring a lot easier, it having such features as IF-THEN-ELSE, the ability to call most Compiler Library procedures directly, and the ability to suppress actual, as well as formal, parameter checking.

Over the past several years, in the course of modifying and developing applications software (not to mention a lot of digging through manuals), I have written a large body of general-purpose library procedures, most of them in FORTRAN 66; these include several subroutines using *alternate return paths*. This is one area where the two FORTRANs are mutually incompatible, and in this paper it is my pleasure to share with you how I worked around this obstacle.

Basically, my solution consists of original library procedures called by one FORTRAN to handle operations performed transparently by the other FORTRAN's object code. Unfortunately, I have no SPL source code to show you--we don't have the SPL/3000 compiler, and we've never bothered to get it because I've found ways to work around it (which I touch on briefly in Appendix II); so, my discussion of these procedures will be in terms of their logic. I hope that my descriptions will be clear enough to enable you to write them yourself.

I am assuming that you are a FORTRAN programmer with some experience and understanding of the MPE stack architecture, and have recourse to SPL. The HP3000/MPE shops most likely to have both FORTRAN compilers are those that are running ASK software and have upgraded their manufacturing or other applications from that vendor to the FORTRAN 77 conversions from FORTRAN 66.

After studying this paper, you will have the potential ability to:

- Call an alternate-return subroutine compiled in one FORTRAN from a program compiled in the other.

- Design a FORTRAN alternate-return subroutine to be callable from either 66 or 77 in the conventional manner, through separate entry points.

- Set up, in FORTRAN, a code segment address as an alternate return point.

Where I am describing code syntax or giving examples, I observe the following conventions:

- Optional coding is enclosed in square brackets ([]).

- Braces ({}) indicate a choice that must be made among two or more coding options.

- Generic names of procedures or parameters are given in lower-case.

- Since I am assuming FORTRAN experience, my code samples are skeletal; an ellipse (...) on a line by itself indicates that source code not relating to my point has been left out.

- Where the FORTRAN manuals use the terms "actual argument" and "dummy argument", I adhere to the more general terms "actual parameter" and "formal parameter", respectively.

What Is an Alternate Return Path?

What I am calling the "alternate return path" construct is a means by which a FORTRAN program may specify a statement label where execution may conditionally resume when a called subroutine returns. It is a standard feature of both FORTRANs and well-documented in the manuals, but I review it here for your convenience.

This is the general syntax in FORTRAN 66 of the CALL statement using an alternate return path:

```
CALL subrtn([parm1[,parm2...],]$label1[,$label2...])
```

The calling sequence is identical in FORTRAN 77, except that "*" is used instead of "$".

The called subroutine--which you must write in the same version of FORTRAN (66 or 77) as the caller, if you're programming strictly "by the book"--is designed for alternate returns by the inclusion of the appropriate number of "*"'s in the formal parameter list, according to the number of label identifiers to be passed by the caller:

```
SUBROUTINE subrtn([parm1[,parm2...],]*[,*...])
```

A simple RETURN statement returns control to the caller at the statement following the CALL; an alternate return path is taken by including a "label index" in the RETURN state-

ment. For example: If the formal parameter list contains at least two "*"'s, a RETURN 2 statement will return to the caller via the statement indicated by the *second* label identifier in the CALL.

Only a SUBROUTINE-type procedure may employ alternate return code.

To illustrate:

```
      ...
C OPEN THE MANUFACTURING DATABASE.
      CALL OPENMFGDB(MFGDB,*800,*810)
      PRINT '(" Mfg. Database opened."/)'
      ...
C CAN'T GET IN JUST NOW.
800   CONTINUE
      PRINT '(/" **MFG. DATABASE NOT AVAILABLE**"/)'
      STOP
C SERIOUS PROBLEM!
810   CONTINUE
      PRINT '(/" **CAN''T OPEN MFG. DATABASE**"/)'
      CALL QUIT(1)
      STOP
      END
      ...
      SUBROUTINE OPENMFGDB(IDB,*,*)
      ...
      RETURN
      ...
      RETURN 1
      ...
      RETURN 2
      ...
      END
```

If the Manufacturing Database is opened successfully, subroutine OPENMFGDB executes a simple RETURN statement which returns control to the PRINT statement following the call; if the database is down for maintenance, OPENMFGDB displays an informative message and RETURN 1 selects a return via statement 800; if the database can't be opened for any other reason, RETURN 2 selects a return via statement 810.

What the Object Code Does

This is what generally happens in FORTRAN at object-code level when a normal subroutine or a function is called (assuming that no parameters are passed by value):

1. Any constants passed to the procedure are copied from the code segment to the top-of-stack (TOS). If an actual parameter is an expression, which could involve a nested function call, it is evaluated and the result placed at the TOS.

2. If the procedure is a function, one or more words,

depending upon the function data type, are allocated at the TOS for the returned value.

3. If there is at least one passed parameter, a parameter list is built at the TOS, consisting of one DB-relative word address or byte pointer per parameter (a FORTRAN 77 string descriptor involves an additional word for the byte count); each address points to where a variable has been mapped by the compiler, or to where a constant or expression value has been stacked.

4. When the PCAL instruction (the active ingredient of the CALL statement and function reference) is executed, a 4-word stack marker is placed at the TOS above the parameter list (if present), and the Q-register is set to the resultant S-register (TOS pointer) value. Among the machine register values saved in the stack marker is that of the index (X-) register, accessible to the called procedure (at object code level) at Q-3; this is important to FORTRAN 66, as we'll see further on.

5. When the subroutine or function executes an EXIT instruction (the active ingredient of the RETURN statement), the machine registers are reloaded from the saved values in the stack marker, which is then deleted from the TOS along with the parameter list. Since the P- and status registers are also saved (at Q-2 and Q-1, respectively, from the perspective of the called procedure), this is how the CPU knows at what address in what code segment to resume execution.

6. Any residue at the TOS--function and expression values, constants, etc.--is put away or otherwise deleted by the caller's object code before execution of the next statement so that the stack is "in balance", i.e. the S-register is pointing where it was when the statement calling the procedure began execution.

I have included, as Appendix I to this paper, an excerpt from our Supplemental Procedure Library documentation which describes the stack marker in the context of the entire stack structure.

When an alternate-return subroutine is called in FORTRAN 66, the object code in the caller loads a 0 into the X-register just before executing the PCAL. The X-register save word in the stack marker thus initially contains a 0.

When a FORTRAN 66 subroutine takes an alternate return path, the object code stores the label index, specified in the RETURN statement, to Q-3 just before exiting. The caller's object code then branches to one or another location depending upon the value it finds in the X-register.

In FORTRAN 77, the caller's object code calls the subroutine as though it were a type INTEGER*2 function, i.e. it stacks a 0 and the parameter list, in that order.

When taking an alternate return path, the object code in the

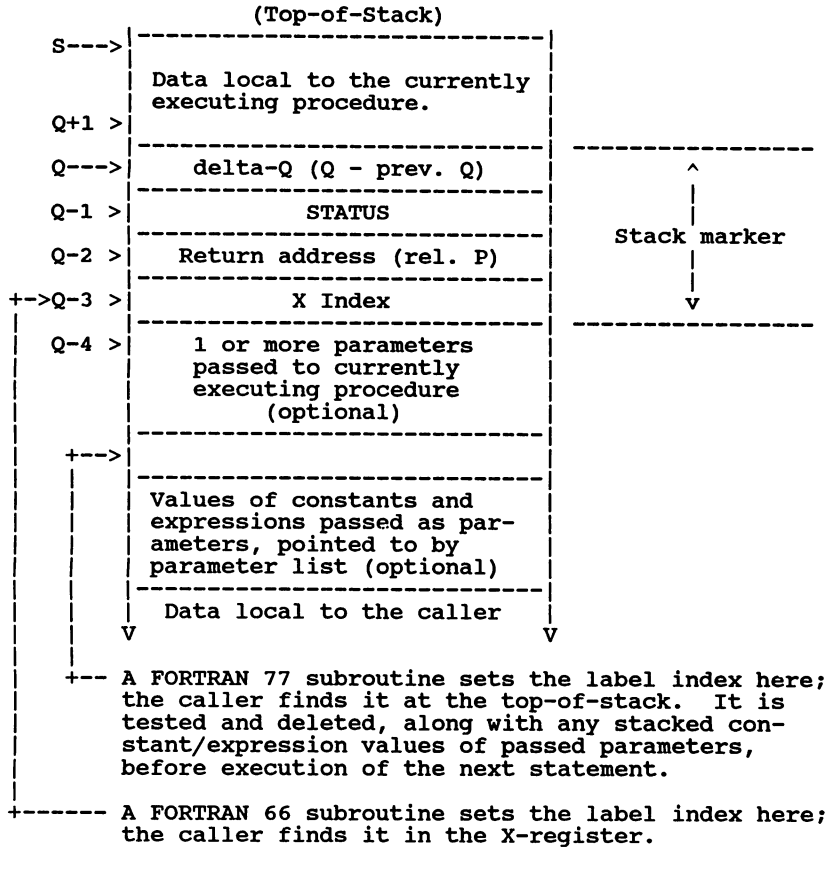
"callee" behaves as though it were returning the label index as a function value. The caller's object code then uses the value it finds at the TOS to determine where to branch.

In both languages, a returned label index value of 0 indicates a normal return via the next statement after the CALL.

At object code level in the above FORTRAN 77 sample, OPENMFGDB finds the address of array MFGDB (formal name IDB) at Q-4 and the label index word, initialized to 0, at Q-5. A simple RETURN statement leaves the index word alone, but RETURN 1 or RETURN 2 stores thereto a 1 or a 2, respectively. All RETURNs generate an EXIT 1 instruction, which deletes only the MFGDB address along with the stack marker, leaving the label index at the TOS for the caller's object code to test and branch to the PRINT statement following the CALL, or to statement 800, or to statement 810, depending upon whether a 0, 1 or 2 is found.

FORTRAN 66 loads and tests the label index in the same manner, except that the called subroutine stores it at Q-3--the X-register save word in the stack marker--so that upon return, the caller finds it in the X-register.

The two methods of setting the label index are illustrated on the next page.


(Top-of-Stack)
S--->
Data local to the currently executing procedure.
Q+1 >
Q--->
delta-Q (Q - prev. Q)
Q-1 >
STATUS
Q-2 >
Return address (rel. P)
+->Q-3 >
X Index
Q-4 >
1 or more parameters passed to currently executing procedure (optional)
Values of constants and expressions passed as parameters, pointed to by parameter list (optional)
Data local to the caller
Stack marker
A FORTRAN 77 subroutine sets the label index here; the caller finds it at the top-of-stack. It is tested and deleted, along with any stacked constant/expression values of passed parameters, before execution of the next statement.
A FORTRAN 66 subroutine sets the label index here; the caller finds it in the X-register.

The Solution for Mutual Callability
Part I: 77 Calling 66

To solve the problem of calling a FORTRAN 66 alternate-return subroutine from FORTRAN 77, I prepared two procedures: subroutine PUTXREG and function GETXREGF.

Internally, PUTXREG copies the formal parameter value to Q-3, whence it is placed in the X-register upon exit; GETXREGF simply returns the value it finds in Q-3.

Implementation of these procedures is as follows:

```
INTEGER[*2] GETXREGF
...
CALL PUTXREG(0)
CALL subrtn[(parm1[,parm2...])]
IDX=GETXREGF()
IF(IDX.GT.0)GOTO {label1
                 {(label1,label2[,label3...]),IDX}
```

Your FORTRAN 77 program must adhere to these design points:

1. Compiler options must include "CHECK ACTUAL PARM 1", which suppresses checking of actual parameter type and plurality for compatibility with the called procedure. The object library structure counts FORTRAN 66 labels as typed parameters in both reference and entry definitions, even though they don't have entries in stacked parameter lists.

2. All parameters passed to **subrtn** must be either local simple variables or constants, since array element addressing involves the X-register, and expression evaluation and references to global data (especially within COMMON blocks) are more than likely to do so. Where **subrtn** is expecting an array name, you must use a local simple variable EQUIVALENCEd to the appropriate element (in which case, the array itself must also be local).

3. Your program size is limited with respect to the amount of local stack. After the first 127 words above Q are exhausted for mapping local and initialized global data, the compiler resorts to mapping arrays for anything that's left and may have to use the X-register even when a local item isn't subscripted. *The X-register must contain a 0 when PCAL is executed.* Use of the "TABLES" compiler list option will show you where and how all of your variables are mapped.

4. The value returned by GETXREGF must be captured in a *separate statement* as shown above, *never* implicitly within an expression.

The Solution for Mutual Callability
Part II: 66 Calling 77

If a FORTRAN 77 alternate-return subroutine was compiled under option "$CHECK_FORMAL_PARM 0" and has at least one data parameter, all you need to do in FORTRAN 66 is to call it as a type INTEGER function:

```
INTEGER subrtn
...
IDX=subrtn(parm1[,parm2...])
IF(IDX.GT.0)GOTO {label1
                 (label1,label2[,label3...]),IDX}
```

Otherwise, two new procedures are called for. Mine are PUSHTOS and POPTOS, which emulate the operations their names imply; the former "pushes" one word of data onto the TOS, the latter "pops" one word of data from the TOS into a variable.

Internally, PUSHTOS overwrites the formal parameter address with the parameter value, then does an EXIT 0 which leaves it at the TOS. POPTOS finds the target value at Q-5, which it copies into the passed variable; an EXIT 2 then deletes the "popped" word from the TOS along with the (one-word) parameter list.

(Happily, PUSHTOS works whether I pass it a variable or a constant. I designed it with a reference parameter so that when one has occasion to use it in FORTRAN 77, one need not suffer the embarrassment of forgetting to employ an $ALIAS directive.)

Implementation is as follows:

```
CALL PUSHTOS(0)
CALL subrtn[(parm1[,parm2...])]
CALL POPTOS(IDX)
IF(IDX.GT.0)GOTO {label1
                 (label1,label2[,label3...]),IDX}
```

Here, the restriction applying to the parameters passed to **subrtn** is that no constants or expressions are allowed; this is because constant, expression and function values have to be stacked *before* the parameter list is built. *The manually-stacked label index word and the parameter list* must *be directly adjacent*.

This time, there is no need to worry about actual parameter checking (which can't be controlled in FORTRAN 66 anyway), since FORTRAN 77 labels are transparent to the object library structure.

Designing a Subroutine for Two-Way Compatibility

An alternate-return subroutine may be designed to be callable from either FORTRAN by using a separate entry point for each language and a utility procedure for passing the

label index in the required manner. My active ingredient for this scheme is (in SPL notation):

```
PROCEDURE SETRTN(ICTL,IPATH);
INTEGER ICTL, IPATH;
```

ICTL is a control word indicating which FORTRAN is anticipating an alternate return path, and where to put the label index; IPATH is the label index itself. After calling SETRTN, a simple RETURN statement is all that's needed to exit via the selected path.

In the case of FORTRAN 77, the label index must be returned to the word just below the parameter list, so SETRTN needs to know the length of the list in words; absent any passed string descriptors, this is simply equal to the number of parameters exclusive of label identifiers. SETRTN may be provided the list length directly through ICTL, but if the subroutine has multiple entry points, SETRTN may alternatively be told through a flag bit to fetch the list length from the subroutine's Q+1, where it has been placed transparently by the initialization code (so that a RETURN statement may be executed at any point in the subprogram unit with the correct stack decrement). The parameter list length is, of course, irrelevant for FORTRAN 66, since the label index is always returned to the X-register via the stack marker.

Internally, SETRTN locates the subroutine's stack marker by using the delta_Q value stored at address Q in its own stack marker; thus, we have Q'=Q-delta_Q. Flag bit ICTL(0:1) indicates whether we're using "mode 66" (off) or "mode 77" (on); if this bit is on, flag bit ICTL(1:1) indicates whether to fetch the subroutine's parameter list length from field ICTL(10:6) (off) or from address Q'+1 (on). The desired label index, supplied by IPATH, is copied to Q'-3 for mode 66, or to Q'-plist_len-4 for mode 77.

I recommend that the two-way subroutine calling SETRTN be written in FORTRAN 77 and structured this way:

```
$CONTROL SHORT,CHECK_FORMAL_PARM 1[,...]
      SUBROUTINE ftn77_entry([parms...,]*[,*...])
      ...
      control=140000B
      GOTO label
C
      ENTRY ftn66entry[(parms...)]
      control=0
C
label CONTINUE
      ...
      CALL SETRTN(control,path)
      ...
      RETURN
      ...
      END
```

The entry point for FORTRAN 77 should be the primary, with

"*"'s provided for the sake of documentation; the name should also include the underscore ("_") character to make it unreferenceable from FORTRAN 66. All formal parameter checking must be suppressed to enable linkage from FORTRAN 66; level 1 checking retains the requirement that both entry points be referenced as subroutines. (By the way, this is what is meant by the phrase "procedure type"; *function* type checking may be suppressed by using any checking level less than 3.)

The label index is specified in the call to SETRTN rather than in the RETURN statement. Since the label index word is always initialized to 0 by the caller's object code, it is not necessary to call SETRTN for a normal exit; nor will a simple RETURN statement clobber a label index you've just set up.

I have successfully used SETRTN in a COBOL II subroutine to set an alternate return path in mode 77; mode 66 is not practicable because COBOL II's formal checking level and the actual checking level in FORTRAN 66 are both fixed at 3. I have not determined precisely how COBOL II handles the parameter list length for multiple entry points, but since it is fixed at object time for each entry point, you can always set up the appropriate value to be passed explicitly to SETRTN.

Another Kind of Alternate Return: The Code Segment Address as Actual Parameter

In designing a COBOL II interface to the FORTRAN 66 Formatter intrinsics, I was forced, for lack of SPL, to write in FORTRAN 77 so that I could "$ALIAS" around the apostrophes in the procedure names. Then, I ran into an interesting problem with the FMTINIT' procedure--the LAST parameter, described in the *Compiler Library Reference Manual* as a "label identifier", is really an address in the calling code segment! Obviously, the "*label" construct is of no use here, for when a error is detected, FMTINIT' takes its alternate return path by copying LAST into Q-2--the return address save word in its stack marker--just before exiting.

The procedures I designed to get around this difficulty are FMTATOP and FMTABOT, as shown in this FORTRAN 77 sample:

```
$CONTROL STANDARD_LEVEL SYSTEM,SHORT
$CHECK_ACTUAL_PARM 2,FTN3000_66 CHARS ON
$ALIAS FMTINIT = "FMTINIT'" (%REF,%VAL,%VAL,%VAL,%VAL)
$ALIAS TFORM = "TFORM'"
      ...
« other $ALIAS's as needed for the list element transfer  »
« routines.                                               »
      ...
      INTEGER GETXREGF
      ...
C VERY FIRST STEP!
      CALL FMTATOP(LAST)
```

```
C TEST CONDITION CODE.
      CALL SAVECCODE
      ICC=GETXREGF()
      IF(ICC)10,10,100
10    CONTINUE              !ALL SET.
      ...
      CALL FMTINIT(FORMAT,UNIT,REC,IOTYPE,LAST)
      ...
« list element procedure calls and other processing.      »
      ...
      CALL TFORM            !NORMAL END OF CALL BLOCK.
100   CALL FMTABOT          !TAIL MARKER.
      CALL SAVECCODE
      ICC=GETXREGF()
      IF(ICC)errlabel,oklabel,eoflabel
      ...
```

FMTATOP copies the return address from Q-2 into LAST, with bit 0 set on (you'll see why presently), sets up CCG in the status save word (Q-1), and does an EXIT 0 which leaves the address of LAST at the TOS. CCG results in a branch taken around everything to the call to FMTABOT.

At this point, LAST is pointing at the instruction following the PCAL to FMTATOP; bit 0 in LAST is on to indicate that this is *not* the "label identifier" we intend to pass to FMTINIT'.

FMTABOT reads LAST by doing a load-indirect from Q-4. Finding bit 0 on, it saves this value internally, reloads LAST from Q-2, reloads Q-2 from the saved previous value of LAST with bit 0 cleared, sets up CCE and exits. The next time FMTABOT is called (upon exit from TFORM'), it finds LAST bit 0 off, in which case it does nothing but exit.

By manipulating the return vector in the stack marker, the initial call to FMTABOT forces a return via the Condition Code test following the call to FMTATOP, which detecting CCE, allows procession to the call to FMTINIT'; at this point, LAST is pointing where it is supposed to--the Condition Code test following the call to FMTABOT; succeeding calls to FMTABOT just drop through, leaving intact the Condition Code from TFORM'.

As you can see, FMTATOP and FMTABOT, as well as the Formatter routines themselves, require some careful program structuring. I deemed it needful to pass LAST to FMTABOT via the TOS, because the object code generated to build a parameter list would put the Condition Code from TFORM' at risk. Because it is, therefore, *absolutely essential* that the address of LAST be at the TOS when FMTABOT is called (an extra word which may be removed at a suitable time by calling POPTOS with a dummy variable), LAST *must* be a simple variable, to insure that the object code will not delete any additional words from the TOS after FMTATOP returns.

I should mention that unless an error occurs and it has to exit through LAST, FMTINIT' creates a temporary global area above the caller's procedure-local data and stores the key

address in DB-2, for use by the transfer routines; this area is cleared away by TFORM'. (The SORT/MERGE intrinsics communicate with each other in a similar fashion.)

Doubtless you're wondering why I'm not using "IF(CCODE())..." to test the Condition Code--somewhere between versions A.00.09 and A.01.00 of FORTRAN 77/V, HP decided to render the CCODE() construct completely useless except for declared system intrinsics, so I developed SAVECCODE as a work-around. All it does is read the saved Condition Code from Q-1, and store to Q-3 a 0, +1 or -1, depending upon whether it finds CCE, CCG or CCL, respectively. I chose to work with the X-register so that with no parameter list to build for SAVECCODE, the status register's precious cargo is out of harm's way until it is saved in the stack marker. Our old friend GETXREGF can then be called to retrieve the representative value in a "plain vanilla" variable--providing for more flexible methods of testing than would be possible with the standard construct. Please note, however, that SAVECCODE does not work (in FORTRAN 77) with any MPE system intrinsics that are explicitly declared as such in your source code.

Conclusion

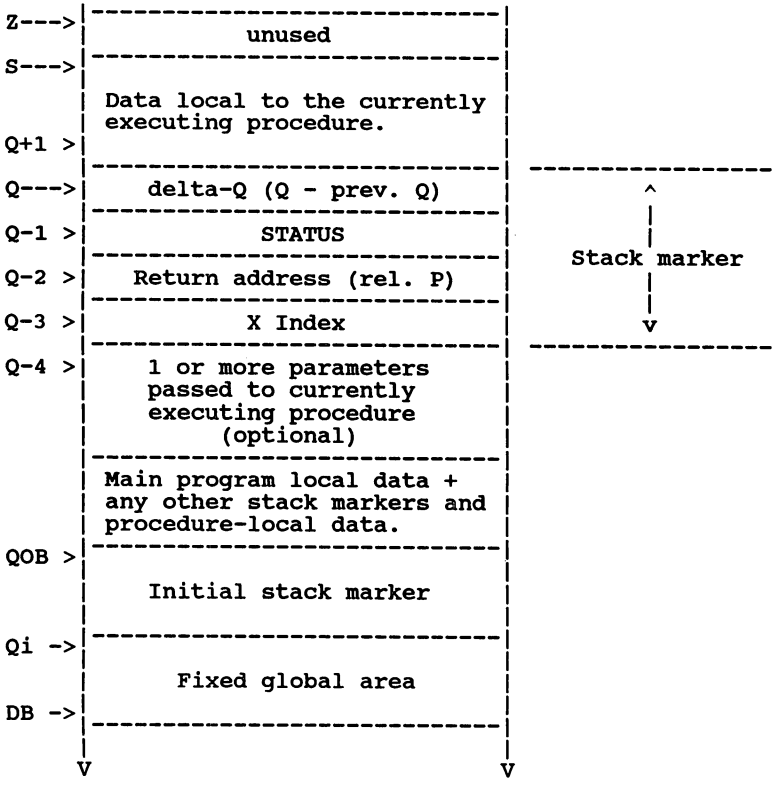
With the utility procedures I've described added to your systems/application programming library, you won't have to rewrite existing FORTRAN 66 alternate-return subroutines (at least, not until you migrate to a different operating system). With the range of mutual callability between the two FORTRANs thus extended, you'll have greater flexibility in choosing a compiler language for a new program. Bear in mind, however, that this interface is best suited for stand-alone utility programs, and interactive applications where the CPU time required for each transaction is not a critical factor.

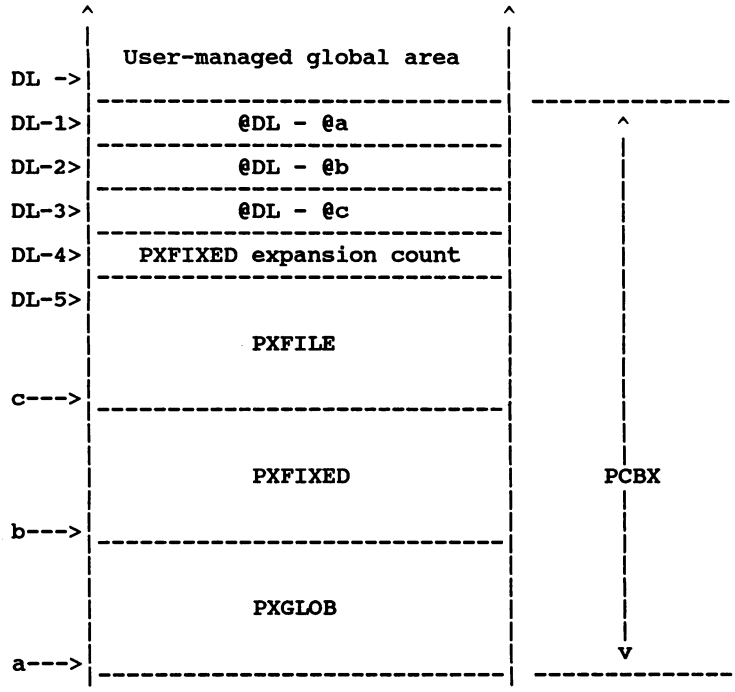
You may find these procedures useful for other things. For example, my universal procedure call interface, used in FORTRAN to call dynamically loaded SL procedures, uses PUSHTOS and POPTOS to stack parameter lists and allocate and retrieve function values.

APPENDIX I

[The following is an excerpt from reference documentation that I wrote for our programming staff. Slight alterations have been made for the purposes of this paper. For more information about the Process Control Block Extension (PCBX), see Eugene Volokh's excellent paper "Secrets of System Tables...Revealed!" (1985 INTEREX Proceedings, Washington, D.C.).]

User Process Stack Structure





DB Data Base. Word address is always 0, by definition. Register value is a segment-relative offset stored at PXGLOB 1.

DL Lower limit of accessability in User Mode. Register value is relative to DB (<=0). Segment-relative offset stored at PXGLOB 0.

Q Base of local data of currently executing subprogram. Register value is relative to DB.

Qi Q-initial; highest DB-relative address of the static global area. Stored at PXFIXED 3. Address Qi contains the PARM value passed through the :RUN command or the process-handling intrinsics; Qi-1 and Qi-2 contain the byte pointer and length, respectively, of the INFO string.

QOB Base of the outer block's (main program's) local data. For a user process, QOB=Qi+4.

S Pointer to the current top-of-stack. Register value is relative to DB. The net effect of calling PUSHTOS or POPTOS is to respectively increment or decrement the S-register value by one.

Z Highest value ever attained by S. Value is stored at

PXFIXED 2. Displacement relative to DB may be dynamically set through the ZSIZE intrinsic; left to its own devices, it never decreases in value.

Stack Markers

The PCAL instruction, generated by CALL statements and function references in FORTRAN, and the CALL verb in COBOL, pushes a four-word stack marker onto the top-of-stack and reloads the Q-register from S. This object contains the following information:

Q-3 Index register contents at the time of the PCAL.

Q-2 Code-segment-relative offset of the instruction immediately following the PCAL in the code segment of origin.

Q-1 Status register contents at the time of the PCAL. The following fields are of significance to the programmer:

(0:1) 0 = User Mode;
1 = Privileged Mode.

(1:1) Set on if external interrupts enabled.

(2:1) Set on if user traps enabled.

(6:2) Condition Code:
0 = CCG;
1 = CCL;
2 = CCE.

(8:8) CST number of the code segment of origin; this and the code offset in Q-2 constitute the return vector.

Q-0 Displacement, in positive words, from the Q-register value just prior to the PCAL; backward link to the previous stack marker.

If the called procedure has one or more parameters, a parameter list is pushed onto the top-of-stack by other compiler-generated instructions prior to executing PCAL.

The EXIT instruction, generated by the RETURN statement in FORTRAN, and the GOBACK verb in COBOL, is the reverse of PCAL; the index and status registers are restored from Q-3 and Q-1, respectively; Q is decremented by the number of words indicated by Q-0; and process execution resumes at the location given by the return vector. The stack marker, and everything on the stack above it, is deleted from the top-of-stack, and S is adjusted to reflect the number of words deleted. The stack decrement (SDEC) field of the EXIT instruction may specify up to 255 additional words to be

deleted...

All stack markers are backward-linked through the 4th word, as far as the initial stack marker...

Any stack marker may be read, modified, or relocated and relinked...; however, the stack locations containing the status save words of stack markers that were "on-line", i.e. in the trace-back chain, as of the last PCAL or EXIT are write-protected in User Mode.

APPENDIX II:
How I Did It Without SPL

As far as the operating system is concerned, a USL that you created is *your* data file to do whatever you want with; so, you are at liberty to alter the machine code that your compiler gave you, and prepare it as changed.

To manage without SPL/3000, I wrote a library procedure for each of several basic stack addressing and register retrieval operations. In each case, I created a program structure in FORTRAN 66, using dummy statements to allocate space for special machine code, and compiled it into a separate USL with all list options. With help from Chapter 9 of the *MPE V Tables Manual*, I then edited the Relocatable Binary Module (RBM) with the DISKED5 utility, replacing object code generated by the dummy statements with the machine instructions I needed. Thus, the operation that "could only be done in SPL" was reduced to a simple library procedure call.

By the time I needed access to the index register for my alternate return path interface, I had enough SPL-type operations encapsulated to write PUTXREG and GETXREG(F) as FORTRAN 66 procedures that work *as written*, without any post-compile editing. Here is the source code that resulted:

```
$CONTROL SEGMENT=UEC'SEG'3,CHECK=0
C**
C** PROGRAM NAME:   PUTXREG/GETXREG
C** SOURCE FILE :   XREGS
C** VERSION      :   STANDARD U.E.
C** PROGRAMMER   :   C.R.N/UE
C** CREATED      :   10/31/89
C
C** UPDATED      :   02/28/91  CRN...SPECIAL PARAMETERLESS
C                              INTEGER FUNCTION ENTRY
C                              "GETXREGF" FOR FORTRAN 77.
C
C  DESCRIPTION   :   PROCEDURES USED BY FORTRAN77 FOR CALLING
C                    FORTRAN66 SUBROUTINES WITH ALTERNATE
C                    RETURNS.  THESE WORK BY MANIPULATING THE
C                    INDEX REGISTER SAVE WORD IN THE STACK
C                    MARKER.
C
      SUBROUTINE PUTXREG(IX)
C
      INTEGER GETQREG,PEEKDB
C
      CALL POKEDB(IX,GETQREG(IDUM)-3)
      RETURN
C
      ENTRY GETXREG(IX)
C
      IX=PEEKDB(GETQREG(IDUM)-3)
      RETURN
C
```

```
C**2/28/91     CRN...FUNCTION-TYPE GETXREG ENTRY FOR FORTRAN
C              77 PROGRAMS WITH VERY LARGE LOCAL STACK.  MUST
C              BE REFERENCED AS A PARAMETERLESS INTEGER
C              FUNCTION.
C
      ENTRY GETXREGF
C
      IT=GETQREG(IDUM)
      CALL POKEDB(PEEKDB(IT-3),IT-4)
      RETURN
      END
```

Integer function GETQREG returns the stack marker location (Q-register value) for the procedure calling it. The location of the *initial* stack marker (QOB) for the calling *process* is returned to the passed variable.

GETQREG resides in a privileged code segment added to our system library; Privileged Mode is required to read the Q-initial value from the PCBX (see Appendix I).

PEEKDB and POKEDB are User Mode procedures used, often in conjunction with GETQREG, to access calculated addresses anywhere on the stack at or above DL. I also use them with the DLSIZE intrinsic, and another procedure of mine called GETDLREG, to access the "heap" (user-managed global area).

* * * * *

Database indexing: The key to performance

F. Alfredo Rego

Adager Lab Manager

Adager
Sun Valley, Idaho
83353-0030
U.S.A.

Telephone +1 (208) 726-9100 Fax +1 (208) 726-8191

Typically, we are interested in accessing a group of entries from a database (for instance, "all the outstanding orders from customer XYZ"). One approach is to scan the database serially, beginning with the first entry and ending with the last entry, "running into" the desired entries along the way. If we have millions of entries, with only a few that meet our selection criteria, we may not be able to afford to use this approach for on-line applications. Another approach is to use indexing methods that allow us to jump directly into the entry or entries which interest us without having to wade through millions of irrelevant entries.

The only purpose of an indexing system is to serve as a performance booster. There are many kinds of indexing methods, with various advantages and disadvantages. In this essay, I focus on the technological challenges posed by the requirement that we should be able to add, maintain and delete indices quickly and conveniently.

Breaking free from indexing traps

There are several types of indexing methods, just as there are many kinds of database management systems. But let's not be confused by this apparent variety. Deep down inside, *all* databases are nothing more, or less, than bunches of bits. All indexing schemes are, by the same token, attempts to shortcut the route that leads us into certain desired bunches of bits within a database.

As long as we keep these fundamental concepts straight, we will be able to take advantage of indices when they exist, without having a nervous collapse when they are gone. Let's take one paragraph from Hewlett-Packard as an exercise in going back to basics. More than 5 years ago, on page 24 of the March 1986 issue of HP's *Information Systems & Manufacturing News*, Terrie Murphy said in an article on ALLBASE:

HPSQL's simple tabular-data structure, with no predefined data-access paths, significantly increases database-administrator (DBA) and programmer productivity. DBAs have great freedom in structuring the database, since it is not necessary to predict all future access paths at design [time]. If the data is available in the database, it is immediately accessible at any

future time. In non-relational models, all access paths need to be known when the database is designed. This adds significantly to overall program-development time. In addition, with no predefined data-access paths, the data structure can be modified in many ways without affecting existing programs; thus greatly simplifying application maintenance.

The issue is "predefined access paths", as viewed from an ALLBASE/SQL perspective. We can easily rewrite the same paragraph from an IMAGE viewpoint:

IMAGE's simple tabular-data structure, with (or without) predefined data-access paths, significantly increases database-administrator (DBA) and programmer productivity. DBAs have great freedom in structuring the database, since it is not necessary to predict all future access paths at design [time]. If the data is available in the database, it is immediately accessible at any future time. In IMAGE, all access paths need not be known when the database is designed. This saves significant overall program-development time. In addition, with (or without) predefined data-access paths, the data structure can be modified in many ways without affecting existing programs; thus greatly simplifying application maintenance.

Without too much effort, we can re-write this paragraph so that pre-defined access paths appear as slaves or liberators from the perspective of *any* database management system. Since most of the HP3000 users share IMAGE as a common bond, and since IMAGE has undeservedly gotten bad press regarding indexing and pre-defined access paths, let's use IMAGE as an example. Even though we will speak in IMAGE terms, let's remember that the same methodology applies to *any* DBMS.

IMAGE allows you independence from predefined access paths (and from many structural modifications), provided you follow some sensible guidelines.

As a prerequisite, you should be aware of several IMAGE design criteria that people tend to ignore:

1. An IMAGE dataset *is* a simple tabular data structure. The widespread belief that IMAGE is a "pointer-based network DBMS" is not true. You can build an IMAGE database that does not have any pointers whatsoever. You can *always* scan a dataset serially, from beginning to end, to select the entries of interest to you, but you might get bored doing this (particularly if you have millions of entries). IMAGE gives you the choice of two kinds of datasets (masters and details), each optimized for a given high-speed access method. You *may*, almost instantly, access specific master entries using hashing, if you wish. But please remember that you don't have to use hashing at all. Likewise, you *may* access, extremely quickly, specific detail entries using an IMAGE-provided combination of hashing and chaining, if you wish. But please keep in mind that you don't have to use chains at all.

2. The IMAGE intrinsics that allow you to add, access and update entries (DBPUT, DBGET, DBUPDATE) have an important parameter: the *list* of those specific fields that interest you.

3. The IMAGE DBINFO intrinsic gives you a wealth of information at run time.

Binding: at compilation time or at run time?

Knowing these (and other) IMAGE design criteria is necessary but not sufficient. As another prerequisite, you should use high programming standards (this, naturally, applies to *any* kind of computer work that you do). A very important programming standard is that you should postpone *binding* as much as possible. This means that you should not burden your programs, at compilation time, with hard-wired stuff. You should wait until run time to adjust to the

prevailing conditions of the day.

In the case of predefined access paths, if any, you should *not* even think about including (or excluding) them in the strategy of your programs. You should find out, at run time, whether a given field in a given dataset is an IMAGE *search field* or not (using DBINFO). If you are *not* dealing with a search field, you might have to do a serial scan of the whole dataset (using DBGETs mode 2 or 3) to find those entries, if any, whose field values you want. (You are certainly free to develop non-IMAGE indexing schemes to avoid such serial scans.) If you *are* dealing with an IMAGE search field, you can be much more efficient. For a master dataset, use hashing (DBGET mode 7). For a detail dataset, use an IMAGE-provided combination of hashing and chaining (an initial DBFIND followed by DBGETs mode 5 or 6).

If you follow these reasonable guidelines, your applications will be totally immune to changes in access paths. You will be able to add or delete paths at will, to suit the performance needs of your users. And, as a fun bonus, since the only difference between masters and details is *access method*, you will also be able to change masters to details or details to masters without impacting any of your application programs.

What do you think now about Hewlett-Packard's assertion that "In non-relational models, all access paths need to be known when the database is designed"? I am sure HP meant to qualify this statement by adding, "if your programming standards are so low that you hard-code everything".

This hard-coding issue has nothing to do with being relational or non-relational. If you hard-code in SQL, nothing will save you from getting into deep trouble. Let's illustrate this observation.

In the case of adding, accessing or updating IMAGE entries, you should *not* even think of using "@" to specify the list of fields that interest you. The "@" list asks IMAGE to deal with *all* the current fields in the dataset. If you add, delete or shuffle the fields of a dataset, you *must* then edit and recompile all the programs that access that dataset. (Absolutely the same is true in SQL if you use SQL's "*" instead of a specific list of columns.)

Since this prospect does not attract me, I strictly follow a methodology with IMAGE field lists. Even though it may take a little more effort up front, I always *build* a list with the names of those specific fields that the program needs to access. The first time I invoke an access intrinsic (DBPUT, DBGET or DBUPDATE), I pass it this list. Afterwards, when I invoke an access intrinsic that depends on the same list, I pass it IMAGE's asterisk list ("*"), which tells IMAGE "don't bother to assemble and check my list; simply use the previous list". (The asterisk "*" means different things to different people: It is important to remember that SQL interprets it to mean "give me everything".)

For more than a decade now, I have been able to add, delete and shuffle fields in my IMAGE datasets. Even though this fact, in itself, is significant, it is even more impressive because I have not been forced to edit or recompile those programs that don't use such fields.

What do you think now about Hewlett-Packard's opinion that "[with SQL] the data structure can be modified in many ways without affecting existing programs"? Of course, HP meant to qualify this opinion by adding, "provided you don't use the SQL asterisk '*' instead of a specific list of columns in your SQL statements".

Indexing and structural freedom

By binding as late as possible, we gain two kinds of freedom: the freedom from pre-defined access paths and the freedom from rigid data structures.

We are able to add, maintain and delete indices quickly and conveniently. We can use the indices that are "bound" with the official DBMS (such as hashing and chaining in IMAGE) and we can use our own (or third-party) indices to complement the official indices.

Since indices are only one aspect of the general database structure, we are also able to add, maintain and delete any other database objects as well.

The fact that, with run-time binding, our indexing schemes are flexible is just one of the consequences of having a flexible over-all approach to database management.

TITLE: Managing A PowerHouse Environment

AUTHOR: David Robinson
PowerSpec International
403 Cross Lake Drive
Fuquay-Varina, NC 27526
919-552-8049

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING.

PAPER NO. 3214

Payment Slip

Please pay the following amount to the address below:

Amount: $1,250.00

Payee: Green Valley Realty

Description: Commission for property sale

Date: March 15, 2024

Signature: _____

USING MPE XL TO YOUR ADVANTAGE - A GUIDE FOR THE APPLICATIONS PROGRAMMER

Pamela Herbert Bristow
A.H. Custom Software, Inc.
El Cerrito, CA 94530
(415) 535-5070

The introduction of the RISC architecture and the MPE XL operating system marked a strong advancement for the HP 3000 line of computers. MPE XL, although similar in commands and syntax to its predecessor MPE V, is a far superior operating system. And now MPE XL 2.x (either 1 or 2) has been delivered and with it comes an even greater functionality than was found in the initial MPE XL offering.

In this paper I will give an overview of the new features in MPE XL for those readers who are new to XL. Then I will discuss a few of the features implemented in version 2.x that are of most interest to me including some aspects of the native mode spooler, 'command input/output redirection' and the FINFO command.

USING VARIABLES

There are 2 types of variables in MPE XL. These are the predefined global variables and the user defined variables. Of the 70 some predefined global variables (as a I counted them in version 2.2) some are read only but many of them can be used to 'customize' the environment for any given user or job by using the :SETVAR command. To see what all of the possibilities are try typing

```
:SHOWVAR @
```

This will give you a listing of all of the system global variables and what their values are. It will also show you any user defined variables that have been set up and what their values are. I will refer to several of these variables in discussing other features of MPE XL that I like to use. A :SHOWVAR without the @ will show you all of the user defined variables.

The way you alter any variable is to use the

```
:SETVAR
```

command.

Variables can have 3 different value types - integer, string and Boolean. The system variables have a preset type and if you try to set one to the wrong type of data the system will return an error. User variable can be set in any of

three ways e.g.

```
:SETVAR X 1 or :SETVAR X 'XYZ' or SETVAR X TRUE
```

Once a variable has been declared the value it contains can be de-referenced to allow variable substitution, and string functions can be applied to extract parts of a variable value. For example the system variable HPDATEF is set to the date in the format

TUE, FEB 27, 1990.

To extract the day of the week enter

```
:SETVAR DAY "![LFT(HPDATEF,3)]"
```

The exclamation point de-references the variable HPDATE meaning that it substitutes the value of the variable in place of the variable itself. The LFT parameter along with the ,3 following the HPDATE variable extracts just the left most 3 characters so that now if you type

```
:SHOWVAR DAY
```

you will see :DAY = TUE

If you then enter

```
:SETVAR TIMEFRAME "!!DAY"
:SHOWVAR TIMEFRAME
```

MPE XL will return :TIMEFRAME = !DAY

This is because the double exclamation points resolve to a single exclamation point giving you back the variable. If you then type

```
:SETVAR THISTIME = "!TIMEFRAME"
:SHOWVAR THISTIME
```

MPE XL will return

:THISTIME = TUE

This de-referencing technique can be used in command files to dynamically set the value of a variable.

A BRIEF WORD ON EXPRESSION EVALUATOR FUNCTIONS

In an example above I use the LFT function to extract the 3 left most bytes of a character string. This is but one of many expression evaluator functions that MPE XL provides. These functions return boolean, string or numeric results and can be used to parse character strings, do numerical manipulations and set up TRUE/FALSE conditions and tests. An explicit discussion of them would require a paper in itself but I mention them here because I use a couple more in examples further on in this paper.

COMMAND LINE HISTORY STACK

The command line history stack was the first MPE XL feature I discovered. In MPE V if you miss-typed a command and then tried to recover with a :REDO only to type :REOD, you would be stuck keying in the entire command again. Not so anymore. MPE XL has a command line history stack that will store a variable number of commands. The exact number of commands is determined by a predefined variable called HPREDOSIZE that has a default value of 20. To see what is in the stack you type

```
:LISTREDO
```

What is returned is a listing of the last 20 commands you have issued along with their absolute number e.g.

```
11) LISTF FILEA,2
12) RENAME FILEA,FLEB
13) RENAME FILE,FILEB
14) QE
15) QUICK
     .
     .
     .
30) LISTF FILEC,2
```

These commands can be referenced in several ways, either by absolute number (i.e. REDO 20) or by relative number (e.g. REDO -2) or by initial characters (e.g. REDO LISTF). This last technique will bring back the most recent command that begins with the characters LISTF.

There is also a :DO command that allows you to re-execute any command in the history stack directly. You can access the command as you do for :REDO (by absolute or relative number or by character string) with the default being the last command executed. The :DO command has an additional feature that allows you to edit the command before executing it. The

syntax is

```
:DO [CMD=cmdid] [EDIT = editstring].
```

For example if you typed in

```
:QUECK
```

when you had intended to key in :QUICK you could edit and execute the command in one step by entering

```
:DO ,"  RI"
```

which would return and execute the string

```
:QUICK
```

The options on the edit string must be in the same position as they would be if you were using the interactive :REDO command. Trying to get the insert, delete and replace commands to come out in the right place strikes me as more difficult than using the interactive :REDO but there are some very handy ways to use :DO. These are:

>	appends characters on to the end of the line.
>R	replaces characters at the end of the line so that the last character in the >R command is at the end of the line.
C	changes all occurrences of one string to another.
>D	deletes from the end of the line moving from right to left. You can enter multiple Ds.

For example, if you typed

```
:LISTF MYPROD,2
```

and what you really meant was :LISTF MYPROG,2 you could re-execute the command quickly by typing

```
:DO ,C/D/G
```

which would immediately re-execute the previous command changing all occurrences of 'D' to 'G'. Similarly if you forgot that you needed to run a program with a LIB=G parameter you could say

```
:DO ,">;LIB=G"  or :DO RUN, ">LIB=G"
```

Note that you must surround the edit string with quotes if there are embedded blanks or special characters such as semi-colons.

GREATER EASE AND FLEXIBILITY IN CREATING AND EXECUTING COMMANDS AND PROGRAMS

There are now more options for implementing and executing command files and programs. In MPE V you could either execute a cataloged command (UDC) or an MPE command or :RUN a program. If you wanted to catalog a new command you had to reset the entire catalog. These bonds have been broken.

The command interpreter (CI) now searches through 3 'areas' each time it tries to resolve a command. The catalog is checked first to see if the command is a UDC. If the command is not found there it is then checked to see if it is a regular MPE command (e.g. LISTF or PURGE). If it is not, the CI then checks for the command in any of the groups or group.accounts specified in the variable HPPATH. The default path is your logon group followed by the PUB group of your logon account, followed by PUB.SYS. The CI also uses an implied RUN so that you can just type the name of a program in and if it is found it will be executed.

This means several things. First of all it means that you never need to create a UDC to execute a program stored in PUB.SYS. If you type in

```
:DBUTIL
```

the CI will find the program DBUTIL.PUB.SYS and execute it. It also means that if you want to alter the way an MPE command functions by using a command file (e.g. having the STREAM command execute STREAMX), just typing the name of the command won't work because the CI will execute the MPE command before it finds the command file. You can get around this by using the :XEQ command which will directly execute the file, bypassing the UDC and MPE regular command checks (but using the HPPATH search).

Given all of these options you now must consider the most advantageous place to store any given command. Just briefly, the major advantages of UDCs are that they execute very quickly, they are shared by all users who have access to them and they cannot be inadvertently purged as easily as command files. The advantage to command files is that they are very easy to change and can be stored and maintained in private groups.

There is another option somewhere in between which is the ability to APPEND and DELETE individual UDCs from the catalog without having to reset the entire thing. This can be very nice if you have a team working on a project where you want to catalog a command that may need to change a bit from time to time or if you need to keep adding UDCs as the

project develops. To do this you type

```
:SETCATALOG UDCB;APPEND
```

If you now need to alter UDCB you can edit it as UDCC and reset the commands in it by typing

```
:SETCATALOG UDCB;DELETE
:SETCATALOG UDCC;APPEND
```

This prevents altering the wrong editor file for all of the other UDCs at the user or account level and ruining everyone's day when you reset the catalog to the incorrect file of commands.

Similarly, you can create individual command files in the group of your choice and change them easily. This is especially useful if you are working on a test version of a program and need to set up a bunch of file equations to run your test version along with a bunch of productions versions of programs. For example, you can create a file called FILEQ.mygroup that looks like this:

```
FILEA = FILEA.TRAINING
FILEB = FILEB.TRAINING
BASEA = BASEA.TRAINING
    .
    .
    .
```

and then set the variable HPPATH to point to your group

```
:SETVAR HPPATH "mygroup,!hpgroup,pub,pub.sys"
```

When you type in the characters

```
:FILEQ
```

all of the file equations in that command file will be created and you never have to touch the catalog. If the file equations need to change you can just edit the file and re-execute it and the new equations will be in place.

SETTING THE HPPATH VARIABLE

The HPPATH variable can be used very effectively to customize the working environment for each programmer. One of the simplest solutions is to have a logon UDC that says

```
SETVAR HPPATH "!HPJOBNAME,!HPGROUP,PUB,PUB.SYS" or
SETVAR HPPATH "!HPJOBNAME,!HPPATH"
```

HPJOBNAME is set to the characters that you type as the first part of your :HELLO command before you type the user name as in

```
:HELLO PHB,MANAGER.TRAINING
```

As long as there is a group called PHB, the HPPATH will be set to check that group for commands first (after checking the UDCs, and MPE commands and not finding the command there) so that if 3 people want to use

```
:QE
```

to call QEDIT but they all want to bring it up with different parameters they can all have their way without having a catalog war.

THE PRINT COMMAND

:PRINT allows you to direct the contents of a file to print wherever you want it to. The major advantage to this is that you don't have to wait for a program to execute to list the output. There are also some options to control the listing. The syntax for the command is

```
:PRINT filename;OUT=outfile;START=m;END=;n;PAGE=p;UNN
```

The :PRINT default is to list the file to your terminal and stop every 23 lines and ask you if you wish to continue. You can use the PAGE parameter to control this by typing

```
:PRINT FILEA;PAGE=0
```

to cause the entire document to print out without a page break. This is useful if you are using a slave printer and want to run with LOG BOTTOM ON to print the file out immediately. If you know which lines of a file you want to look at you can use the START and END parameters. :PRINT FILEA;START=20 will start listing the file at line 20. :PRINT FILEA;START = -20 will list the last 20 records of the file.

CHANGING GROUPS WITHOUT LOGGING ON AGAIN

How many times have you gone to run DBUTIL or KSAMUTIL or some other data base/file management program only to discover that you are not logged on in to the group that the data reside in? Until now you had to log in again, specifying the correct group as your home group. In doing so you would

lose any file equations or variables that you had set up during your session. With MPE XL you can issue the

```
:CHGROUP  groupname
```

command and you will be 'moved' to the group specified. When you are done with the file maintenance you can type :CHGROUP without the group name parameter and you will be back in your logon home group.

COPYING FILES WITH FAR FEWER KEYSTROKES

FCOPY always seemed to me to be a tedious command to use. All of that 'FROM=' and 'TO=' and designating a file's current status (NEW,OLD etc). seemed such a bother to me. With MPE XL comes the :COPY command that eliminates all of that. You can now type

```
:COPY FILEA.MYGROUP,FILEA.PUB;YES
```

and FILEA will be purged from .PUB if necessary and copied in from .MYGROUP. If you leave off the ;YES parameter the system will ask you if you wish to purge the 'TO' file giving you an opportunity to confirm the copy before executing it.

THE FINFO COMMAND

The FINFO command gives you an easy way to get information on a file. It has the same capabilities as the FLABELINFO intrinsic but can be used directly. The syntax for the command is

FINFO(MYFILE,KEY) where MYFILE is the name of the file of interest and KEY can have either a numeric or literal value. Some of the possible values are:

option num	option name/alias	return type and meaning
0	"exists"	boolean TRUE if file exists
1	"full filename"	string, fully qualified name
2	"group name"	string
3	"account name"	string
6	"creation date"	string format DAY, MM DD, YYYY
6	"created"	
-6	"intcreated"	integer format YYYYMMDD
7	"accessed"	string, format DAY, MM DD, YYYY
-7	"intaccessed"	integer format YYYYMMDD
8	"last mod date"	string format DAY, MM DD, YYYY
-8	"intmoddate"	integer format YYYYMMDD
12	"file limit"	integer

```
  13      "formatted foptions"    string
  19      "end of file"           integer
  24      "last mod time"         string HH:MM AM or PM
```

You can use the command to return a boolean value e.g.

```
  IF FINFO("MYFILE","EXISTS") = TRUE
     THEN ...
```

or you can write a little command file to give you information about a file or about a file equation in a format of your own personal design that is most useful to you e.g.

```
PARM FILE
IF FINFO("!FILE","EXISTS") = TRUE   THEN
     SETVAR A FINFO("!FILE","FULL FILENAME")
     SETVAR B FINFO("!FILE","CREATION DATE")
     SETVAR C FINFO("!FILE","FILE LIMIT")
     SETVAR D FINFO("!FILE","END OF FILE")
     SETVAR F FINFO("!FILE","LAST MOD DATE")
     SETVAR E FINFO("!FILE","LAST MOD TIME")
     ECHO        ************************************
     ECHO           file name:  !A
     ECHO           created  :  !B
     ECHO           limit    :  !C
     ECHO           eof      :  !D
     ECHO           last mod :  !F !E
     ECHO        ************************************
     ENDIF
IF FINFO("!FILE","EXISTS") = FALSE THEN
    ECHO  ------!>  The file !FILE does not exist  !<-------

    ENDIF
```

To use this command file you simply enter

```
SF MYFILE
```

at the MPE prompt and, if MYFILE exists you will see

```
   ****************************************
      file name:  MYFILE.MYGROUP.MYACCT
      created  :  MON, MAY 6, 1991
      limit    :  100
      eof      :  25
      last mod :  WED, MAY 8, 1991 12:24 PM
   ****************************************
```

If you have a file equation such as

```
TESTFILE = MYFILE.MYGROUP.MYACCT
```

you can enter

```
SF TESTFILE
```

and the display described above will be returned. This is very handy if you have command files that set a particular file equation to any of many values depending on what you are testing and you can't remember how it is currently set.

If TESTFILE does not exist this command file will return

```
--------> The file  TESTFILE does not exist <----------
```

The reason for the exclamation points is to prevent MPEXL from trying to evaluate the 'less than' and 'greater than' signs as part of the expression.

THE NATIVE MODE SPOOLER

With the release of MPE XL 2.1 came the demise of that ghastly, ghostly utility SPOOK. Spool files are handled quite differently in MPE XL 2.1 and beyond. The output spoolfiles are now written to disk as regular MPE files (with a lot of special characters to control printing) with the name

```
dfid.OUT.HPSOOL
```

where dfid is the file's device file id. Just as with SPOOK you can keep the file on disk by using an OUTCLASS parameter with a low number in your job card. Then, when you raise the outclass number to allow the file to print it will be deleted from disk. However, if you use the ;SPSAVE parameter in your job card, the file will remain on disk even after it has printed. This gives you automatic report backup. To see what output spoolfiles you have you can use the LISTSPF command.

The feature I most like about the native mode spooler is that you can use the PRINT command on the dfid.OUT.HPSPOOL file while the job is executing! This allows you to view the progress the job is making along the way which, for long running jobs, can be very nice.

COMMAND INPUT/OUTPUT REDIRECTION

MPE XL 2.1 and above gives you the ability to 'grab' the output of any command and write it to a disk file , or to 'feed' input to a command from a disk file. This is a very powerful feature because it gives you almost unlimited capabilities for using the operating system to read and write files. It also gives you the ability to manipulate the output from commands and use the result as is, or altered, as input to another command. I have used this feature to create a STREAM command that traps the job number and output spoolfile device id and stores them in variables so that I can easily

device id and stores them in variables so that I can easily manipulate the job. I called my STREAM command STRM so that I didn't have to catalog it to override the MPE STREAM command.
This is the command file STRM:

```
PARM JOB
ERRCLEAR
IF FINFO("!JOB","EXISTS") = TRUE THEN
   CONTINUE
   STREAM !JOB > CMDS
   CONTINUE
   IF HPCIERR <> 0
      PRINT CMDS;START=2
      ENDIF
   RESET CMDS
   INPUT JOB_NUM < CMDS
   ECHO   !JOB_NUM
   SHOWOUT SP;JOB = !JOB_NUM > CMDS
   SETVAR FILE_INFO FINFO("CMDS","EOF")
   IF FINFO("CMDS","EOF") > 2 THEN
     PRINT CMDS;START=2;END=3 > DFIDFILE
     ENDIF
   INPUT DFID_NUM < DFIDFILE
   SETVAR DFID RTRIM(STR(DFID_NUM,11,8))
   ECHO !JOB_NUM !DFID
ELSE
  ECHO
  ECHO
  ECHO THE JOB FILE !JOB DOES NOT EXIST
ECHO
ECHO
ENDIF
```

When this command executes it first checks to see that there is such a file as !JOB that can be submitted. I could simply have returned the HPCIERRMSG but I preferred a move informative and personalized response. In this case, if I try to STRM MYJOB.JOB and no such file exists the command file returns

```
THE JOB FILE MYJOB.JOB DOES NOT EXIST
```

Otherwise, it streams the job and writes the output of the stream command to a temporary file called CMDS. If the streaming was not successful (due to a bad job card or something) then the second line of CMDS is written to the screen. This contains the error message.

A successful stream will cause the #Jxxx that usually shows up on the screen to be written to CMDS. This output is then referred to as INPUT and the value is written to the

variable JOB_NUM which is echoed to the screen so that I can see what it is.

The command file then executes a SHOWOUT command on the job and writes the output to CMDS again so that I can trap the output spoolfile device file id. The actual output from the SHOWOUT command is:

```
DEV/CL  DFID    JOBNUM  FNAME    STATE    FRM  SPACE RANK PRI
#C
LP      #O1234  #J789   $STDLIST OPENED       2048        1
 1
OUTFENCE = 1
OUTFENCE 1 FOR LDEV 6
```

What I want from this output is just the #O1234 which, in the real 80 column world starts in position 11 of the second line and can be up to 8 characters long. Therefore, I print the second line of the CMDS file to a file called DFIDFILE and then use this as INPUT to DFID_NUM which is parsed into a variable called DFID using the expression evaluator functions RTRIM and STR. What I actually see on the screen is

```
  #J789
#J789   O1234
```

and what I have is a variable called JOB_NUM that is set to the job number of the last job I submitted (in this case #J789) and a variable called DFID which is set to the output spoolfile device file id of the last job I streamed.

I now have one additional command file that I use to monitor the progress of my job. I call this one PRT and it performs a PRINT command on the output spoolfile for the last job I submitted:

```
PARM PAGE=0,START=1
PRINT !DFID.OUT.HPSPOOL;PAGE=!PAGE;START=!START
```

The PAGE and START parameters can be altered as time goes on so that I can start further and further in to the spoolfile as it gets longer. For example, when the job has just been executed I can type

```
PRT
```

and see the output from the first line to the end with no stops. If I see 50 lines at that time and want to check back in 5 minutes, I can type

```
PRT 0 50
```

and the output to my screen will begin at line 50 of the spoolfile.

You can use these same techniques to write command files to alter the input priority of a job or abort it or whatever you choose and have the privileges to do.

If you wish to have output from a command appended to the end of an already existing file you can specify that as:

```
SHOWOUT SP;JOB=!JOB >> CMDS
```

SUMMARY

The MPE XL command interpreter has some very powerful features that allow you to customize your working environment and to create simple or complex command procedures to automate routine functions. The end result for me has been an increase in productivity and in my level of satisfaction with working on the 3000. Whether you choose to become proficient at developing complex command files or just use some of the more basic features of this operating system, you are sure to find that the improvements it offers will make your professional life much more efficient and a lot more fun.

REFERENCES

Cooper, Kevin "A Programmer Looks at MPE XL" Interact volume 8, issue 10

Cressler, Scott and Vance, Jeff "The Life of an MPE XL Command"
Interact volume 9, issue 9

Mak, James Tsze-Leung "Customizing MPE XL Commands" Interact volume 9, issue 12

Cressler Scott, and Vance, Jeff "Advanced CI Programming"
BARUG - Proceedings of the 1990 Santa Cruz Conference

Paper Number 3216

MAKING QTP RUN EFFICIENTLY

by
John D. Alleyn-Day
Alleyn-Day International
1721 M. L. K. Way, Suite 3
Berkeley CA 94709-2101
415-486-8202

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 may not be entirely deserved. 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 example using QTP (Powerhouse). The same situation could arise in several other fourth generation languages. Some of what I will present is more complex than would normally be the case, but it contains elements that serve as a general example for use as a cookbook for anyone that wants to follow my technique. Also my remarks apply principally to Classic machines running MPE/V. Spectrum machines have different considerations that will modify what I am considering here.

I have worked at several clients with fourth generation languages and seen various circumstances in which batch programs written in QTP or some similar language were taking excessive times to run. The fourth generation language is usually used to access data from KSAM files and from IMAGE databases. 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!

It became apparent to me 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. The simplicity of the programming methods encourages programmers, myself included, to construct very inefficient programs without realizing the true import of their code. I will illustrate this for you as we go along.

The usual start to a QTP program (or a QUIZ program) is a statement along the following lines:

```
access datafile link to auditfile
```

followed by various selection, sorting and updating criteria. Most other fourth generation languages will have a similar statement that joins two or more files together. The performance problems start right here with this statement. We have to look very carefully at what this statement is doing.

Let us suppose that these files are big files. In this context, by a "big" file, I am going to mean about 500,000 records. With files of this size the program will take many hours to run. If it starts at 5.00 p.m. then it may not be finished when work starts at 8.00 am the next morning!

If we had written a good COBOL program to solve this problem, it would have taken only an hour or two to run. However, this inefficiency is not an inescapable problem associated with the fourth generation language. There is a cardinal rule which must be applied. Know what your fourth generation language is actually doing.

The statement above is asking QTP to read "datafile" sequentially and for each record to read a corresponding record from the "auditfile". The records are linked by a "key" value, implicit in the data structure for our simple situation. Reading the datafile sequentially is usually fast and depends significantly on its blocking factor. If we suppose a blocking factor of 10, then we can estimate that this process will take about 40 minutes (I am using 20 I/O's per second as an average disc access time). If you are set up for multi-record access this process should take only minutes, because multi-record access essentially uses a very high blocking factor.

The part that makes the performance so poor is the random reads implied by the access to "auditfile". In general, each record requires a single disc read, and the random access process will take about 7 hours, and there is nothing multi-record access can do about it. If the linkage is to a detail file rather than a master, then each record will probably require two disc I/Os and will take about 14 hours.

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 QTP, leaving you to make the necessary adjustments to achieve similar results in your own language.

Now that we know why our program takes so long to run, we can set about making it run faster -- much faster. Twenty

or thirty percent improvement in efficiency will not be enough; we need it to run five to ten times faster. For this phase, we adopt 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 our QTP program we 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 we avoid this?

We have to avoid the random reads. In fact, we have to go back a few years to the heyday of mainframes and batch processing and take a look at how we got our COBOL program to do the same process in an hour or two.

How did we write our batch COBOL program without random access? We made extensive use of the sort and merge programs and our COBOL program did a lot of record matching. Specifically, we would take our two files, extract the data we needed and add to each one a "record type" to identify which file the record came from. Then we would merge them and sort them by a composite key, made up of the key that we wanted to match on followed by the record identifier.

We would then have a file in which all the data for a particular key would be grouped together, with the first record of a group coming from the first file and the second record coming from the second file. At this point our COBOL program would merge these two records into a single composite record and write it out to a new file.

This file contains precisely the data that we would have got from our "access" statement. However, except for the record matching, we have used only standard record extraction, sort and merges, all of which have been carefully developed over the years for optimum efficiency. The COBOL record matching program is also reading files sequentially and should therefore be very efficient as well.

How can we do this for our own case, without having to write a COBOL program? The first parts of the processing are fairly elementary and I don't plan to describe them in detail. Use SUPRTOOL or COPYRITE (or write a short program in QTP) to extract the data and create the new "record-type" field. The extracts should have identical formats, with locations for each field that will eventually be needed, and the second file should be appended to the first. Sorting these extracts will give us the file described above with the records grouped by matching key.

From here we have two possibilities. If you have been using SUPRTOOL then you will also have SUPRLINK which will do the record matching for you. If you don't have SUPRTOOL, then we have to use QTP to get the same result, and I will now describe in detail how you do this.

Actually, I am going to describe a QTP program that does rather more than this. However, you can use this as a model and use just the pieces that you need. This program was written for an "audit" process" in which we were concerned with changes that were being made manually to a database and we wanted an independent check on those changes. For this purpose, we wrote a simple extract routine in QTP that sorted and copied the dataset contents to an MPE flat file before any changes were made. After the manual changes were complete, the same extract was run again to produce a second MPE file. We then compared the two files, to determine the records that had been added or deleted.

In this case we were interested not in matching records, but in non-matching records, so the program is more complex than in the simple case above. Here is the QTP program. The file "auditnew" is a combination of the two files sorted as described above.

```
access auditnew
sorted on terminal-key, map-key, record-type

temporary n-data character*26
temporary n-flag integer*1
temporary p-flag integer*1
temporary p-data character*26

item p-flag = n-flag at record-type
item p-data = n-data at record-type
item n-data = map-key + terminal-key
item n-flag = record-type

subfile temp2 keep alias delete-file &
  if (p-flag = 1 and n-flag = 1) &
  or (p-flag = 1 and n-flag = 2 and p-data <> n-data) &
  include p-flag, p-data

subfile temp2 alias add-file &
  if (p-flag = 2 and n-flag = 2) &
  or (p-flag = 1 and n-flag = 2 and p-data <> n-data) &
  include n-flag, n-data

subfile temp2 alias last-record at final &
  if n-flag = 1 &
  include n-flag, n-data
```

```
subfile temp2 alias first-record at initial &
  if record-type = 2 &
  include record-type, map-key, terminal-key
```

What we have done here is to produce the program using QTP that we would previously have written in COBOL. We have set up fields in "working storage" to keep track of data from previous records. Be very careful when using this program. The order of some of the statements, particularly the "item" statements, is crucial for proper operation.

Actually, this program won't quite solve the original problem. It actually finds all the non-matching records, rather than the matching records, a task which is very tricky to do in QTP using the conventional "access" statement, and becomes impossible in this case where we are comparing two MPE files. However, the changes needed to get matching records only are trivial and left as an exercise for the reader.

One final point. A programmer must use judgment in applying the techniques I have illustrated. On small files the increased efficiency possible with these 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 may be easily justified.

Finally, I suggest that the Fourth Generation Language Developers consider this problem. Many vendors claim that their systems run batch programs. This is true -- in a way. Fourth Generation Language programs <u>can</u> 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 example that I used from QTP now imply the use of keyed reads, leading to inefficient batch programs. Why could not a Fourth Generation Language interpret the same "access" statements 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 as I know, the fourth generation language vendors have not seen this as a problem that they need to address. However, as mentioned above, there is one group that has stepped into the breach, namely Robelle. They have a part of SUPRTOOL called SUPRLINK, which carries out the matching

of records in an efficient manner and which appears to have all the capability 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. 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 judgment as to when it is worth the trouble and when it is not.

Paper # 3217
TurboImage Logging

By: Larry Boyd
Bradmark Technologies, Inc.
4265 San Felipe, Suite 800
Houston, TX 77027
(713)621-2808

Introduction

Transaction logging is a feature provided with standard TurboImage that can save you from losing hours, days, or even weeks worth of data. This paper will discuss issues such as who should use transaction logging, what can logging be used for, how do you get started, and some logging tips. But first we need to talk about some of the myths about logging.

Myths about Image Transaction Logging

There are three major myths about Image Transaction Logging. The first revolves around implementation, the second around performance, and the third about usage. All will be briefly discussed now, but covered in greater detail later.

1. "Logging requires source code changes."

   Logging **DOES NOT** require source code changes. Once logging has been enabled on a data base all *Physical* transactions (DBPUT, DBUPDATE, DBDELETE) will be logged.

2. "Logging is a heavy load on the system."

   Although logging does use some resources, most users **WILL NOT** notice any degradation in performance. In most on-line applications logging will have no statistical effect on performance. If this is a question, enable logging for a week and monitor the performance.

3. "Logging is for Recovery."

   Many of those who log **DO NOT** use logging for recovery. There are many additional usages. Logging records ALL adds, deletes, and updates, so the log file can be a valuable resource for resolving specific problems and issues.

Who Should Use Logging?

In some cases logging can be used as an optional aid to an application. In other cases, logging should always be used as protection.

1. Logging should **ALWAYS** be enabled when using the *Output Deferred* option on a database.

   Both TurboImage/V and TurboImage/XL have the capability to enable *Output Deferred*. This can be done using DBCONTROL for the duration of the current DBOPEN or it can be turned on using DBUTIL for all DBOPENs.

   With this option TurboImage/V will not write Image Buffers to disc until they are needed by buffer management. On XL, the MPE Transaction Manager is bypassed and data pages are held in memory until no more memory is available. Also, on either machine, a DBCLOSE will cause the data to be written to disc immediately. Since data is not written to disc and no log of the changes are made by MPE, a system failure is very likely to cause major structural damage to the data base. The ONLY way to repair this damage correctly is to use DBRECOV or to restore the data base from a backup tape.

2. Logging should be used if you have a **High Volume** of transactions processed between your database backups.

   If recovery by hand from a failure would require a large amount of time, then logging should be used. Since there are two ways, Roll-forward and Roll-back, to use DBRECOV, if applications are set up properly, recovery can be done in a matter of minutes instead of hours or even days.

3. Logging should be used if transactions have **No Paper Backup.**

   Often in today's world customers place orders over the telephone and there is no paper backup. Here it would be almost impossible to recreate all transactions if recovery was necessary.

4. Logging should be used if an **Audit Trail** is required by internal or external auditors for changes to a database or dataset.

   With Image Transaction Logging and a log file reporting system, such as **DBAUDIT**, all changes to a database, dataset, or even to a field can be reported. The log file keeps up with the change and the environment from which the change was made (i.e. Program, User, Logical Device, etc.)

5. Logging should be used if there are many **Remote Users** modifying the database.

   Remote users are difficult to communicate with or monitor easily. Since almost all contact is over the telephone you cannot see the input screen or data. This makes it much more difficult to recreate data from failures and give less control of the physical access to data bases, which may increase the need for an audit trail. With logging enabled you can examine the log file to see what was exactly entered when they report a problem and to monitor all accesses to data bases.

6. Logging should be used on databases which require **long backup times.**

   Some databases take quite a while to backup. If logging has been enabled, your data base will be protected if a failure occurs. Therefore, backups can be reduced from daily to every other day, for example, or from daily to weekly.

   Remember to consider the volume of changes (see #2). If the volume is not considered high, meaning recovery would not take too long, you can increase the time between backups.

7. Logging can also be used as a very good **debugging tool.**

   With all the information given in the log file, you can usually find the smallest of problems. This information can be used to verify that the data is in the correct format, or that the IMAGE intrinsics are being called in the correct order.

8. The information can be used for monitoring **performance problems.**

   When are the high I/O periods for updates? If all of the updates are being done during a small period of time, can they be spread out over a longer period to reduce peak time? Which programs do the most updating? Can these programs be reviewed for performance evaluation?

What Can Logging Be Used For?

Recovery

The primary use for logging is **recovery.** There are three different types of recovery. They are Intrinsic Level Recovery (ILR), Roll-Forward recovery, and Roll-Back recovery. Both Roll-Forward and Roll-Back recovery require Image Transaction Logging, whereas ILR does not.

ILR is a process that is used to ensure the **physical** integrity of a database. This keeps broken chains from corrupting the database. Once enabled, the logging and recovery is done automatically.

****NOTE**** Although ILR is available on the XL machines, it is no longer required, or recommended, to protect the structural integrity of the database. This is because TurboImage is integrated with Transaction Manager (XM). There are other changes to ILR that can be found in the TurboImage/XL Database Management System Reference Manual (#30391-90001) appendix H.

Roll-Forward and Roll-Back recovery are used to ensure the *logical* integrity of a database. A *logical* transaction is defined as a sequence of one or more *physical* transactions to a database. To define a *logical* transaction in a program the DBBEGIN and the DBEND (or DBXBEGIN/DBXEND can be used on XL) procedures are used to surround all of the DBPUT, DBDELETE, and DBUPDATE calls (*physical* transactions) that make up the *logical* transaction.

If Image Transaction Logging is enabled on a data base, DBRECOV may be used to recover the data base. In the event of a system or data base failure, when DBBEGIN and DBEND are used, the completed *logical* transactions can be recovered. DBRECOV will not apply the *physical* transactions if a DBBEGIN is found without a DBEND marking the end of the *logical* transaction. When a "soft" failure occurs on the data base, the Roll-Back recovery can be used to "erase" the *physical* transactions that were added before the *logical* transaction failed. When a "hard" failure occurs, Roll-Forward recovery is used to re-apply all transactions since logging was started.

Source code changes are required to convert a sequence of *physical* transactions into a *logical* transaction by surrounding the *physical* transactions with DBBEGIN and DBEND. Without the DBBEGIN and DBEND recovery is still possible, but not at the *logical* transaction level.

A very important development of IMAGE has been the expansion of the *logical* transaction to more than one data base. At one time there was no way to tell IMAGE that a *logical* transaction included *physical* transactions to several data bases. UNTIL NOW! In the current versions of TurboImage, on both MPE/V and MPE/XL, there are new DBBEGIN/DBEND modes. These will allow a *logical* transaction to include more than one data base, and is called **MDBX**, or Multiple Data Base Transaction.

Audit Trail

As stated earlier, there are many additional uses for Image Transaction Logging. It is a very good process for **audit trail** reporting. Not only can you report the actual changes to the data base, but you can also report the specific user, port, or program information. So, you can monitor changes in complete detail.

Debugging

With the user and program information you can use logging to see what is actually happening to a data base. This can be great for **debugging** a program (Did it update all the data sets correctly? Did it update them in the correct order?).

Testing

In a third-party application you can use logging to verify that the product does everything correctly. This can greatly reduce the guess work while **testing** a third-party application, or even your own.

Tuning

For **tuning**, logging can be used to see which program, data base, or data set is used the most. Then you can allocate your time based on where optimizing will help the most.

How Do You Get Started?

There are several steps that must be completed to initiate transaction logging on a data base. I will discuss the basic steps necessary to start logging to disc, but for more detail see the TurboImage manual (for both V and XL, see chapter 7).

1. Give Logging (LG) capability to the account. First, the system manager must give the account LG capability.

   ```
   :altacct account;cap=ia,ba, ..., LG
   ```

2. Give all users who will be using logging Logging (LG) capability. This includes application users and the logging manager.

   ```
   :altuser mgr;cap=ia,ba, ..., LG
   :altuser user1;cap=ia,ba, ..., LG
   ```

3. Build the log disc file. When logging to disc you must build the new file and allocate space for it.

   :build *discfile*;disc=1000,32,1;code=log

   The *discfile* is a standard MPE file name, assigned by you. If the *discfile* is five characters or less, and ends with "001", you can specific AUTO on the GETLOG command. Then logging will open a new log file when the current one is full. Otherwise, logging will fail when the log file fills.

4. Log on as a user with logging or operator capability to obtain a log identifier (*logid*), and link the *logid* to a log file.

```
:Hello mgr.account
:getlog logid;log=discfile,disc
```

The *logid* is a variable that is assigned by you.

5. Run DBUTIL to set the data base to the *logid*.

```
:run dbutil.pub.sys
>>set DBNAME logid=logid

>>enable DBNAME for logging
        or
>>enable DBNAME for Roll-back
```

Enabling a data base for logging sets the Roll-forward switch, and enabling a database for Roll-back sets the Roll-back switch, along with the Roll-forward switch.

```
>>exit
```

The above steps are one-time operations and do not need to be repeated.

Once the data base has been enabled for logging, no one can open the data base until the *logid* has been "activated" by the console operator using the :LOG command (If the :ALLOW command has been executed, the use must have logging or operator capability).

```
:log logid,start
```

There are several concerns that should be considered to protect the data base and log files during backups and recovery. These are described in detail in Chapter 7 of the TurboImage manuals.

To turn off logging, the steps are reversed and the commands are a little different:

```
:log logid,stop

:run dbutil.pub.sys
>>disable DBNAME for Roll-Back
>>disable DBNAME for logging
>>exit

:rellog logid
```

Logging Tips

Since Transaction Logging has very powerful features and many uses, following is a short list of items to consider.

With multiple data bases you should use the new **MDBX** option of DBBEGIN/DBEND. If a failure occurs during the "window" of the DBEND, the *logical* transaction will still be complete.

If you make any structural changes to a data base then a backup should be done and logging must be restarted with a new log file. The log file must match the data base structure (same set and item names and numbers).

On DBUPDATEs, Image does not include the Key fields of detail data sets, only the relative record number.

With DELTA logging, IMAGE logs only the fields from the first item changed to the last item changed. This reduces the record size necessary of the log record, thereby, reducing the amount of storage space.

Logging does NOT automatically stop when the :LOG *logid*,stop command is issued. A flag is set and when the last active user closes the data base, then logging will stop. This means that if someone fails to log off at the end of the day, you will not be able to stop logging and do backup.

Conclusion

We have quickly looked at logging and its many advantages. We have covered when you should log, what you can do with logging, how to get started, and tips on logging.

We did not cover the disadvantages, nor the details on managing the logging process. You should read Chapter 7 of the TurboImage manual before starting. But do not be afraid of logging. It has very good uses and should be used in most of your shops.

PAPER # 3218

"THE DATA WAREHOUSE APPROACH TO DEVELOPING DSS/EIS APPLICATIONS"

KIRK W. BUECHER
HEWLETT-PACKARD GREELEY SITE
700 71ST AVE GREELEY, CO 80634 (303) 350-4291

Intro -

Imagine that you are back in ancient Rome, you are a slave, owned by rich and powerful masters, (that's not too hard to imagine, is it?). You know that someday you may be asked to fight a strong and resourceful Gladiator known as EIS or Executive Information Systems, for the entertainment of your masters. There are several possible outcomes : You may have been lucky up to this point and have not yet been asked to fight. You may have been unlucky and were called into the ring to do battle against the mighty EIS, unfortunately you had little or no training, poor tools or weapons, and limited support and were beaten. Perhaps you were more talented or better equipped and managed to win your first battle against EIS, but are now being asked to go into the ring again to face a new more complex EIS.

Regardless of what category you or your systems group may fall into, the purpose of this paper/presentation is to give you a edge so that your first or current EIS challenge is successful. That edge is "The Data Warehouse Approach to Developing DSS/EIS Applications".

Decision Support Systems and Executive Information Systems have been hot topics in computer trade journals for the last three or four years. The focus of most of these articles is on the tools, the user interface, the many pitfalls, or the amazing advantages to implementing a DSS/EIS application. What is often over looked or discounted is the importance of the data source for these applications.

This paper/presentation will focus on the foundation that often determines the success or failure of these applications. Sometimes known as a "subject database", "summary database", or "data warehouse", it is this structure that is one of the true keys to developing successful DSS/EIS applications.

A Quick Refresher -

What is an EIS? A common definition is that an EIS system is intended to provide easy access, to easy to understand, primarily graphically displayed, individually specified, information that can be interrogated and manipulated by non-technical, often managerial end users.

This type of system on executive's desktops has the potential to allow them to :

o Simplify - sift through immense quantities of data and quickly extract relevant information

o Accelerate - eliminate the constraints of time and distance to the flow of information

o Expand Thinking - widen the horizon of thinking and understanding of the business

o Motivate - affect people's attention and behavior

Index Group consultants active in this area, have seen EIS's proven crucial to executives making major changes in business directions (such as shifting from a product to a market focus), organizational structure (especially flattening the organization and the elimination of staff functions) and organizational communications patterns (as in moving to global product sourcing).

If taken to its full potential, an EIS system can be the informational nerve center, the "vital signs" monitoring post for an organization.

Although the concepts of an EIS have been in existence for over a decade in academic circles, it wasn't until the last year or two, that the tools and technology were widely available and affordable enough for most companies to develop their first application. Five years ago, the only EIS tools a firm could purchase were from Comshare or Pilot Executive Software, ran only on large IBM mainframes and cost as much as $300,000.

Todays tools are often PC based, priced under $1000, are easier to develop in, and in many cases have more features. It is also very likely that you already have some of the pieces needed to build a complete EIS application.

The major components of EIS tools are data retrieval/extraction and data display/manipulation.

Beginning with data extraction, the tools range in ability to access data in a single format on a single platform to many formats on PC's, Mainframes, and Unix machines. Query is one example of the low end of this range and HP's product, Information Access an example of the high end.

With data display/manipulation, there is another range of capabilities. With these tools, the spectrum can be broken down based on flexibility of the display, "drill down" features, the ability of playing "what if" games, the ability to perform more in depth analysis of the data, and ease of use for the user. Some examples of this range would be Gallery Collection or Harvard Graphics at the low end, Lotus 123, Excel, Forest & Trees somewhere in the middle, and PowerPlay, Lightship, and Level V near the high end. All of these packages have deferent strengths and weakness of feature sets that make any sort of comparison rough at best. Newwave and Windows 3.0 can also be powerful supplements to these tools. It is important at this point to note that most display tools have some data storage requirement. These files or databases are simply needed to drive the display tool. They are not the original source of the data or the "data warehouse". They are the holding area for data that the display tools need to provide "drill down" capabilities for example.

The Data Warehouse - Intro

The Data Warehouse is an emerging technique for information resource management. To quote William H. Inmon, a noted database author and lecturer, "If the 1970's were the age of database, and the 1980's were the age of PC's and fourth-generation languages (4GLs), then surely the 1990's will be the age of the data warehouse."

The premise behind the data warehouse is to deposit and maintain, on-line, all pertinent information with full history retention, integrated data along the lines of the major subjects of the corporation. It is important to note that the data warehouse contains fundamentally different data from classical operational databases. Operational databases contain current-value data (data that is accurate at the moment of usage). Inventory levels, account balances, and current addresses are all common forms of current-value data. In contrast, the data warehouse contains data that is accurate at some particular moment in time (that is, data over a time perspective).

For example, imagine an airline's decision process when its current "Summary" style EIS system points out that its on-time record is poor in comparison to the Industry average. What can be done? Management can not simply yell at everyone to "Hurry up!" Someone must look at the actual detailed history and analyze the factors that led to those late departures and arrivals. It must be possible to determine why flight #189 from Denver to San Diego was twenty minutes late on May 6th, 1991.

Using another analogy that points out one of the common short comings of many EIS applications; Imagine that you are trying to ford a wide, muddy river. Are you interested in the average river depth, average current speed, average bottom composition, highest water level this month, and highest level year to date? These may be interesting data points, but if that is all the information you have, you may have a rough crossing. What you really want to know is the shortest, safest place to cross the river.

It is because the data in the warehouse is historical and integrated across application boundaries , that it is the ideal foundation for EIS processing. The data warehouse's main purpose then becomes to serve as the source of reconcilability for the EIS. Data integrity and consistency are key to any data warehouse. The value comes from the explicit agreement, across systems, on the standard data item names, representations and meanings. Data items can be transferred and still have the same meaning regardless of context.

A number of leading companies, including Bell Atlantic, American Airlines, Citibank NA, Bankamerica Corp., AT&T, Liberty Mutual Insurance Co., and K-Mart Corp., have in place or are implementing systems of this type. Even IBM has heralded this technology, due to how well it fits with their "Repository" strategy.

The current EIS headache -

When beginning an EIS project, it quickly becomes apparent that the data Management wants, is often spread across several databases, contained in files, and in a spreadsheet or two. In examining the data sources, it is common to discover that the update timing, item naming conventions, formats, and structures, all differ greatly. This makes data synthesis, analysis, or comparison very difficult. Without major push-ups by the EIS software and the people developing the application, the end users are really limited to using the data in only a slightly different way from the purpose envisioned when the data was first automated.

If your company is lucky or smart enough to have historical data available electronically, users generally cannot pull data from these sources to merge with other sources. This is because once a data item has been removed from its original application and the further from that application it moves, interpreting it correctly is and becomes more and more difficult.

If a data item cannot be correctly understood, it cannot be combined with other information to manufacture a new product. Instead, it is just data pollution. Once this type of data pollution enters your EIS application, it can throw you into another common EIS trap. One of constant justification, reconciliation and research into why your EIS's data just doesn't seem to add up or correlate to some other source that the users think it should.

To add to these problems, as optical data storage comes down in price and gains acceptance, we will see an explosion of new data and applications as increasing amounts of textual, graphical, audio, and video data is automated and put online.

The Data Warehouse is the answer. By making connections between data and its interpretation consistent, a data warehouse can add this value. It makes different application's data like different nation's currency. The currency has value where it originates but must be translated outside its issuing country. The more widely understood the data is as negotiable information, the more valuable it is. The data warehouse acts as a filter so systems groups can accumulate information and develop a way of managing mixed systems that retain the data's content and make it consistently available and valuable to the end users.

Data warehouse components -

Data warehouse consists of three functional components, the data warehouse's database, the warehouse's directory or dictionary, and the warehouse's filtering or conditioning system.

The data warehouse's database contains all the company's data for a given subject area in a relational database management system. (It would be possible to do in Image, but this is not recommended. Image vs. Relational will covered briefly in a later section.) It establishes information authority, superseding that of the production application system's. As it evolves, production systems will be considered more as "data in process" systems, while the data warehouse contains edited and audited data in rationalized and integrated data structures.

The warehouse's directory/dictionary defines the data stored in the data warehouse's database. The directory/dictionary could be implemented as a commercial data dictionary or as extensions to the DBMS catalog. It would contain data item names, edit rules, formats for data, name and structure mapping between various data sources and the data warehouse database's accepted internal format. Additionally, it controls the data conditioning process. The mappings are triggered by the conditioning system when data is entered into the data warehouse's database.

The warehouse's filtering/conditioning system is a collection of procedures and control statements for extracting data from sources outside the data warehouse. It then restructures, renames and reformats that data according to the standards stored in the directory/dictionary. The filtering/conditioning system is implemented using a fourth-generation language or a data manipulation product. (Cognos's Powerhouse and HP's Information Access would be examples of each of those.) Some of the other functions of the filtering/conditioning system are to validate data quality while reporting results back to the originating system and to update the warehouse's directory/dictionary. Once this system is in place, it does not eliminate the need for application (source) systems to edit and audit data. It is impossible to ensure that edits and audits are performed or performed properly at the first processing point. Data, therefore, must be re-examined before being loaded into the data warehouse.

Issues - -

IMAGE vs. Relational -

When building a data warehouse, it is highly recommended that a relational database package be used. Reasons for this recommendation include the outstanding flexibility of relational (which is very critical in the early stages of the data warehouse development), the portability across platforms, and standards that SQL brings with it.

In the non-HP world when a EIS type system is discussed, the use of a relational package is considered to be simply one of the basics needed to be successful.

On the other hand, if your shop does not have relational package in house, do not add to your already large challenge, by trying to learn and implement relational while building a data warehouse and EIS applications. Stay with IMAGE, it is where you already have some level of expertise, it is stable, and there is a wealth of modification/monitoring/performance tools available. In addition, it is very low cost when compared to relational and can out perform relational in many cases. Stay with IMAGE but start making plans to move to relational. It is the better solution.

(Note: There are many more global issues to be considered as well when contrasting IMAGE vs. Relational. Please see the Bibliography and other proceedings for more information sources.)

Selecting items and timing of extracts -

In most cases the data needed to populate a data warehouse will be determined by the wants of the executive first, then by the natural associations of the key data with secondary data needed to add meaning and depth of understanding. It is important that the MIS group ask the right questions of management and that management understands that they have a key role in determining what data on what schedule will be moved into the data warehouse. Once this has been decided, the extraction process is put in place so that the original source's data can be routinely transferred through the filtering/conditioning module and into the data warehouse data base. For the sake of audit balancing, one strategy would be to use a common synchronization point, hourly, daily, weekly or monthly, at which point new data is transferred.

Another problem area arises when a key data item is identified and is found to be in two or more sources. Often the problem is compounded when different formats or values are discovered. There are no secrets to resolving these type of problems, it just takes time. Time is needed to investigate the source systems, interfaces between systems, performance issues, and closeness to the real world "thing" that this data represents. Sometimes even after all the data about the data is gathered it comes down to a best guess tempered by how "pure" the data feels or how global the format appears.

Getting Started -

Companies that implement a data warehouse do so because their production applications cannot support the intensive analyses or top-down management examinations of data that is needed. In the absence of such an intense, management driven need, it is unlikely that a data warehouse will be successful. Without a data warehouse, EIS systems are more difficult to build and implement.

Too much planning will kill a data warehouse. The reasoning here is that building a data warehouse often involves a lot of politics, some up-front technical confusion, a data management learning curve, and a considerable re-education about an organization's information. If an attempt is made to answer every objection before moving toward implementation, the project will be slowed to the point of losing your executive sponsor's support.

Keep in mind that the complete analysis of an organization's data normally takes years and is never complete. Do not allow others to set this as a goal for you or for the data warehouse. If this happens, rest assured that success will be very elusive. Instead, meet the EIS driven need behind the first project. Make that successful, and then build on it.

Pick a project big enough to have a good return on your companies investment, but small enough to be reasonably accomplished within six to nine months. If no concrete results are seen within this time frame, funding and management's commitment to the project could easily be lost.

Use whatever software is already available for your first attempt. Image or relational PC based databases, application generators (4th GL's), report writers, and existing data dictionaries, should all be used and leveraged from.

One area not to cut corners on, is the user interface tools. Here, a small amount of money, less than a $1000, can go a long way. Look at the many PC based EIS tools with the end users. Find a package that meets their needs and matches their skill level. Get the most out of that package by coupling it with Windows 3.0 or NewWave. With a slick front end that makes your users happy, it is then possible to develop your data warehouse and it's infrastructure without heavy time constraints.

Choose current staff members who already have experience with the existing tools and are willing to use then in a new way. Look for people that have a good data management perspective. Anyone with relational experience or eager to learn relational technology, would be a plus.

Summary -

Your time is coming. That time when your rich and powerful masters call your name to come and challenge the mighty Gladiator, EIS. It doesn't matter if this is the first or fiftieth time, consider building a data warehouse.

It has the ability to capture and maintain, on-line, all pertinent information with full history retention. A data warehouse can improve the Executive's access to data for decision making at all levels. Instead of seeing just summary data or averages for a year or a month, analysts and Executives alike can look at the actual data and base their decisions on more solid ground.

With the data warehouse as a powerful weapon in your hand, you will be able to defeat todays challenger and be better prepared to meet the challenges of the future.

Bibliography -

1. Parvin Rahnavard, "Decision Support System", Interex Orlando Proceedings, 1988.

2. Terrence O'Brien, Janet Eden-Harris, "Executive Information Systems", Interact June 1989.

3. Terrence O'Brien, Janet Eden-Harris, "How to Build an Executive Information System Using Today's Technology", Interex San Franciso Proceedings, 1989.

4. Will McClatchy, "EIS Powers Executives", InformationWeek, October 9, 1989.

5. Gary Guiden, Douglas Ewers, "The Keys to Successful Executive Support Systems", Indications 5:5, September/October 1988.

6. Cort Van Rensselaer, "Real-World Data Management", Computer Decisions, 1988.

7. J.A. Zackman, "A Framework for Information Systems Architecture", IBM Systems Journal 26:3 1987.

8. John Bongiovanni, "Solving the EIS Puzzle, The Real Story", Information Center Manager, AT&T Denver, 1990.

9. Mike Phillips, "Future Trends in Data Resource Management", DAMA, Denver, January 11, 1989.

10. Cecilia Bellomo, "To Go Relational or Not? An Introductory Guide", Interact 10:2, February 1990.

11. O.J. Larson, "Strategic Importance of Relational Database Technology", Interex Nashville Proceedings, 1989.

12. William H. Inmon, "The Cabinet Effect", Database Programming & Design 4:5, May 1991.

INTEREX '91 - Paper #3219

Critical Item Update - What Will It Do For Me?

by
Steven M. Cooper
Chairman, SIGIMAGE

Allegro Consultants, Inc.
2101 Woodside Road
Redwood City, CA 94062
UUNET: scooper@allegro.com
Voice: (415) 369-2303
FAX: (415) 369-2304

For over a decade now, one enhancement request has been consistently at or near the top of every list submitted to HP: add the ability to update critical items to **IMAGE**. At the Boston **Interex** conference in 1990, HP committed to the implementation of this long-awaited feature. Now that it's coming, exactly what is coming? What will it do for me? This paper attempts to answer these questions.

First, let's make sure that we understand what the problem is today. **IMAGE** (aka **TurboIMAGE**) manages records for us. We add new records by calling *DBPUT* and delete old ones by calling *DBDELETE*. These records are composed of fields. After we obtain a record by calling *DBGET*, we can update a value in a field by changing the value in our copy of the record and then calling *DBUPDATE*. But not always. If we attempt to change the value of a field that **IMAGE** considers a "special" field, then the *DBUPDATE* will fail, returning the exceptional condition #41, "*Attempt to Update a Critical Item*".

What does **IMAGE** consider a "special" field? There are three kinds of fields that are "special". Master datasets contain records that are retrieved by a key value. All records in a master dataset must have unique key values. The field that is designated as the key in a master dataset is the first of the "special" cases and cannot be changed via a call to *DBUPDATE*.

Detail datasets contain records that do not have to be unique. They may be retrieved sequentially, but are more often retrieved through chained access, that is, records that contain the same value in a search field are chained together by **IMAGE** for rapid retrieval. A detail dataset may have 0 to 16 such search fields, determined by the database administrator (dba). These fields are also "special"; none of them can be changed (yet) via a call to *DBUPDATE*.

And lastly, when accessing these detail dataset records by reading up or down a chain, the dba has a choice of retrieving the records in chronological order (the order in which they were *DBPUT* into the dataset) or in sorted order, sorted according to the value in another field, known as the sort field. Each of the search fields may have a sort field specified. These sort fields are also declared to be "special" inasmuch as **IMAGE** will not allow them to be changed via a call to *DBUPDATE*.

The **IMAGE** manual refers to these "special" fields as critical items. Thus, an attempt to change these fields is commonly known as the critical item update problem. (Yes, there is a difference between "items" and "fields", and we should be calling this "the critical *field* update problem". But as long as it gets solved, I don't much care what the manual calls it.)

The following is my understanding of the enhancements underway by HP, as presented by HP at the March, 1991 Reno **SIGIMAGE** meeting. As this software has not yet been released by HP, we must consider this information preliminary and subject to change prior to its release. Use this information only to start the flow of creative juices; don't start changing procedures until you receive the updated software and carefully review the associated documentation. The plans that HP has shared with us include enhancements to **TurboIMAGE/XL** only, not its MPE V counterpart.

By default, **IMAGE** will continue to function as it always has, rejecting all attempts to *DBUPDATE* a change to a critical item. However, the database administrator will be able to use **DBUTIL** to enable and disable the new critical item update feature on a database-by-database basis. When enabled, you will be able to change two of the three kinds of "special" fields via a call to *DBUPDATE*:

* If you change the value of a search field in a record of a detail dataset, then **IMAGE** will remove the record from the chain that it is currently on (corresponding to its original search field value) and place it on another chain. If the record has other search fields that were not changed, those chains will be unaffected.

* If you change the value of a sort field in a record of a detail dataset, then **IMAGE** will reposition the record in that chain, according to its new sort value. Again, all unchanged chains will remain unaffected.

* **IMAGE** will continue to reject attempts to change the key value in master datasets.

Since it is the search and sort fields and not the key fields that are affected, HP may call this feature *SSUPDATE* (for Search and Sort Update). Incidentally, we will continue to set the MODE parameter of *DBUPDATE* to one, as we always have.

Some programs have probably been written that expect the *DBUPDATE* to fail if someone attempts to update a search or a sort field. Of course, these programs will continue to work as before by default. But, if the database administrator enables the new feature, these programs may begin performing updates that would have otherwise been rejected. For this reason, two other intrinsics have been enhanced: A new mode for *DBINFO* will inform the program if critical item updates have been enabled for this database. A new mode for *DBCONTROL* will allow the program to turn off the feature for itself, even if other accessors of the database may be using it.

Now that we understand what we can't do today and what we will be able to do soon, let's examine what this much-requested feature will do for us. First, consider the unsophisticated user using a tool such as **QUERY**. Our user knows how to FIND records and knows that values that have been found can be updated with the REPLACE command. However, at some point in the past, the user tried to REPLACE the value in a search field only to have it fail with some mysterious error message. When brought to the attention of a data processing person, the advice was to DELETE the record, then ADD it back again, carefully reentering all of the fields. Even if no mistake was made in the reentry, the new record will be placed at the end of all of the unsorted chains to which it belongs, thereby destroying the original chain chronology.

Once the database had been enabled for critical item update, our user would have been able to REPLACE without problem, never having to understand the database's design, **IMAGE** internals, or the DELETE/ADD kludge. Programs such as **QUERY** will automatically, silently, and efficiently take advantage of this new feature.

How about more sophisticated users, perhaps the database administrators themselves? On several occasions, I have considered linking a new automatic master dataset to an existing detail dataset in order to speed up retrieval. Adding the dataset and the linkage is easy with the very powerful third-party database utilities. The tough part is trying to determine how many existing programs, interactive and batch, will now fail attempting to modify the field that has just become a search item. Most of the time, the difficulty in finding and fixing these programs is so overwhelming, that we just give up and live without the path. What a shame.

The same problem applies in an even simpler case: deciding to sort a path that is currently unsorted. Here too, the new sort field would become a critical field, thereby causing unknown numbers of programs to fail due to **IMAGE**'s previous refusal to perform critical item updates.

Of course, once the database is enabled for critical item update, the paths and the sorts can be added with impunity. This gives the database administrator new power to tune the database and keep up with changing business needs without a massive maintenance programming task.

Perhaps an example is in order. Consider an Order Processing system that keeps invoice data in a detail dataset. This dataset might be linked to a Customer master, so that we can obtain all of the invoices for a customer quickly. It might also be linked to a Date master, so that we can obtain all of the invoices produced on a particular date quickly. We might also have a status field that indicates whether the invoice is "PAID", "CURRENT", "30 days past due", "60 days past due", or "over 90 days past due". Now, since the vast majority of invoices in this dataset are (hopefully) paid, to find our delinquent invoices, we will have to sequentially read through the entire dataset. If we decide that we need quicker access to this information, the logical approach would be to add a Status master, linked through the status field. However, now when an invoice changes status, say when it is paid or when it rolls from "current" to "30 days", we will not be able to simply *DBUPDATE* it, but will have to *DBDELETE* it and *DBPUT* it back. The programs that make these *DBUPDATE*s will all have to be identified and changed. The critical item update enhancement eliminates all of these problems; we can add our path without adversely affecting existing programs.

Ironically, it will probably be the report writers and the data extraction programs that will benefit most from this enhancement request, even though these programs do no updating at all! Huge amounts of time are typically spent in these programs, doing sequential reads of datasets. They would obviously benefit greatly from the addition of new paths. Indeed, many report generators will automatically use these new paths once they are added. But they have not been added, due to the risk of adversely affecting other programs with the critical item update problem.

All of the improvements mentioned above come for free: no programming changes are required. But are there other benefits that would come from recoding? The answer is yes. Most programs are written to be cognizant of critical fields. Typically, when a program needs to update a search or sort field, the program will *DBGET* the record, *DBDELETE* it, change the value in its buffer, then *DBPUT* it back again. With critical item update enabled, the program could instead *DBGET*

the record, change the value in its buffer, then do the *DBUPDATE*. Besides the elimination of one intrinsic call, there are other performance savings. **IMAGE** does not have to add the deleted record to the free chain and then instantly remove it again for the newly added record. But if the record has other unchanged search fields, the reduction in overhead can be significant. In the worst case of sixteen paths, only one of which has its search value changed, the new *DBUPDATE* will take around 5% of the CPU time and far fewer disc I/O's than the *DBDELETE/DBPUT* pair would have taken. The improvement could be amazing for programs that do this often. And, as a bonus, the chronology of the other fifteen chains is maintained!

Clever programmers have been anticipating this change for years. In any case, we can take a clue from them and borrow their technique in anticipation of the upcoming change. Whenever they want to *DBUPDATE* a detail dataset record, they first try by calling *DBUPDATE*. If this fails with Exception Condition #41, they then silently do the *DBDELETE* and *DBPUT* automatically. These programs will automatically begin running faster, without modification, once the databases they access have been enabled for critical item update.

HP has breathed new life into **IMAGE** with the critical item update enhancement. Now that we've been given what we've been asking for all of these years, it is up to us to use it to its full advantage. With a little forethought, this feature will not only make it easier to update databases, but can have a significant, positive impact on the overall performance of the system as well.

Paper: 3221

Title: Memory Management

Author: Laurie Facer

Company: FACER System
Performance Division
106 Boldleaf Court,
North Carolina 27513

Phone: 1-800-458-1558

Memory Management On MPEXL

Memory is an intermediary storage area used by active programs (processes) to store code and data.

It is used because of its speed. It is considerably faster than disc access and is more accessable by the CPU. Because of cost, however, its availability is limited. It is, therefore, used as an intermediary storage area and requires the transfer of data and code to and from disc.

A memory management system has been written to ensure the most efficient use of the scarce memory resource. Memory management's function is to ensure that code and data required for CPU processing is available when needed. The more disc I/O that can be eliminated by holding code and data in main memory, the more efficiently will the machine operate.

In doing this, memory manager should function with the least possible amount of overhead on the system.

Memory Management Architecture

To be able to understand how well memory management is operating, we need to understand some basic concepts.

Memory manager works with logical pages. Data and code are stored in pages both

in memory and on disc. A logical page represents 4096 bytes (KB) of storage area. (The physical page size is 2048 bytes. The word pages in this paper refers to the logical page.) A page may be either fully or partially occupied and the data or code may flow over into one or more further pages.

Because of MPEXL's virtual memory addressing system, there is no practical restriction on the number of pages that a process uses. The 64KB limit set by the MPE Classic 16 bit addressing system has been eliminated. MPEXL's 48 bit virtual addressing system allows an addressable memory space of 280 trillion bytes. The 64 bit addressing found on the 980 allows an even larger addressable memory space.

However, as stated earlier, the amount of main memory available is limited. Memory manager manages this limitation by ensuring that as many pages as possible are stored in memory with the least amount of disc I/O.

Intitial Page Allocations

Before a process can obtain access to the CPU it has to have the code and data (pages) that it needs allocated into main memory. This requires disc I/O to transfer pages into memory. As a process performs its processing it will probably require additional pages to be made available in memory.

The additional allocations for a process once it has been initialized will result from the need to load new code segments, when the process's stack is increased or heap expanded beyond the current page, to perform file reads and writes, or when its pages need to be swapped from transient memory.

Code and Data Pages

The allocation of code and data pages occurs most frequently at the initiation of a process or job/session. After that point they should occur infrequently.

Code allocations after initial allocations indicate bad locality of code. That is, code that is continuously being called by another routine resides in a different page. MPEXL does, however, have a complex routine to minimise the probability of both pages not being allocated.

Data pages are allocated after initial allocations due to stack or heap expansion beyond the current page. A program that is increasing its stack size will cause these allocations.

File Pages

For a process to be able to process a file record, that record must reside in main memory. The disc I/O system requests the page containing the record required and memory manager stores it into main memory.

If the page required by the process is not memory resident the process is faulted in CPU and is placed in the dispatcher queue until the page is made available. By avoiding page faults, processes complete faster and overhead on the system is reduced.

One of the main problems with MPE on the classic architecture was the level of disc I/O faults. This was addressed with the introduction of memory disc caching. Caching placed the block containing the required record plus additional blocks of the same file into main memory. There was a strong probability that the next record required by the process would be in the current or one of the additional blocks placed in memory. This reduced the fault rate with subsequent reductions in system overhead.

MPEXL recognises that disc I/O is the slowest part of computer processing and has implemented a prefetch algorithm that reduces disc I/O. This is done, however, by placing additional overhead on memory utilization.

Prefetch Algorithm

MPEXL has a prefetch algorithm that replaces the role played by disc caching on MPE. It is a superior solution to disc caching and has made dramatic improvements to performance.

The prefetch algorithm utilizes the additional memory available on the MPEXL machines by loading into memory not only the currently required pages but additional pages. For serial reads the pages loaded start at the current page followed by up to eight pages following from that page. Random reads cause the current page plus pages either side of the required page to be loaded.

The prefetch algorithm works well on MPEXL for three reasons - a) MPEXL has a lot more memory available than the classic machines and loading additional pages places less strain on resources; b) memory management under MPEXL is more efficient than under MPE; and c) the prefetch algorithm is an integral part of MPEXL and has not been grafted onto the operating system in the same way that disc caching was.

Transient Space

It does not take too many processes to be active before all available memory is utilized. To reduce the restictions of memory size on the ability to create and run processes, an overflow area is set aside on disc. This area is called transient space (virtual memory on MPE).

Memory manager utilizes this area by swapping pages not being referenced by "active" processes from main memory to transient space. It then uses the page areas made available in main memory for pages required by "active" processes.

Managing Allocated Pages

Once pages are loaded into main memory, memory manager then utilises transient space as an overflow area. To do this memory manager has to decide which pages are to be kept in main memory and which pages can be swapped to transient space.

Each page in main memory is flagged as being in one of five states - present, absent, in motion in, being kicked out, and recoverable overlay candidate. The important conditions to understand are present, absent and recoverable overlay candidate (ROC). A present page is one that is being referenced by a currently active process. An absent page is one that is empty, and a ROC is a present page that has not been referenced recently by an active process.

Memory manager utilises both absent and ROC pages when allocating pages into memory. It searches its absent pages first, then searches the pages flagged as ROCs. ROC pages are a second best option as they contain data and may need to be swapped to transient space before another page can be allocated. This swapping process requires disc I/O.

MPEXL tries to maintain a pool of 32 pages for new page allocations. If the pages available falls below this level it scans the present pages looking for ROCs. Each time it scans memory looking for ROCs, it tests to see if a flag is turned "on". If it is, it then resets the flag to "off". If the flag is already set to "off", it sets the page to ROC status. If a page has been set to "off" and it is referenced by a process, the flag is set back to "on".

With more recent releases of MPEXL, memory management has been made more efficient by flagging prefetched pages from sequential reads as ROCs as soon as they are loaded into memory. This has had the result of lowering the priority of prefetched pages staying in main memory. Prefetched pages take a lot of memory space. At the same time, the need to maintain them in memory is less pressing than it is for other objects. By flagging prefetched pages as ROCs immediately, the number of pages available for providing free space is dramatically increased.

MPEXL also maintains a memory pressure flag. This flag is based on the number of times memory manager needed to cycle memory to find ROCs. As this value reaches thresholds, the criteria for marking pages as ROCs becomes more severe.

When there is too little memory for the current level of processing, the search for ROCs becomes urgent and results in system overhead. The system overhead appears in the form of higher memory manager utilisation of CPU, increased process wait times as processes wait for pages to be made present, and increased disc I/O as pages are swapped to and from transient memory.

Memory Pressure

Pressure is placed upon memory for three reasons - a) process initiation is high, b) faults due to absent pages are high, and c) memory available is too small.

Process Initiation

When a process is initiated, a high number of allocations are generated due to the initialisation of code and data pages into memory. Additional page allocations for active processes depend on new code segments being required, expansion of the stack pointer beyond the current page, disc I/O, and transient memory activity.

Absent Page Faults

Absent page faults occur for two main reasons - a) file read or write occurs and page is not in memory, and 2) page has been allocated to memory but subsequently swapped to transient memory. An absent page fault initiates a disc I/O to transfer pages from disc to memory. Memory manager has to find the free pages in main memory into which to place the new pages. For file activity (excluding mapped files) this also means utilising the prefetch algorithm and allocating not only the required page but all associated pages.

High page allocations place a work load on memory manager - even with adequate amounts of memory space.

Inadequate Memory

When memory becomes fully utilised the memory manager has to more frequently perform the function of maintaining the "free" page pool and swapping pages to and from transient space. This activity places an overhead on the system in the form of disc I/O and CPU utilisation. Even if allocation rates are low, an inadequate memory size will see page management generate additional workloads.

Is Memory A Bottleneck?

To determine if memory is under prressure, that is, memory is becoming or has become a bottleneck and to determine what to do about the situation, the following questions must be answered:

1) What indicators show that memory is under pressure?

2) What type of pressure is memory under - lack of memory space or high allocation rates?

3) What processes are causing memory pressure?

CPU Utilization

The first indicator of memory manager being a bottleneck is the amount of CPU time it uses to perform its functions. Fortunately memory manager has a low utilisation of CPU time. However, any CPU time diverted away from user processes needs to be minimized.

The amount of CPU time that can be tolerably diverted away from processes will depend on your processing requirements. I would recommend that - over extended periods - 2 to 5% CPU utilization by memory manager is an indicator of moderate memory pressure and anything above 5% would indicate high memory pressure.

Process Wait Times

Memory is also a bottleneck if it is causing processes to wait. If a process has to wait for memory related activities, this extends its processing time and lengthens its response time. If many processes are continuously waiting for memory, then memory is a botteneck for those processes.

Causes Of Memory Pressure

Lack Of Memory

There are several good indicators of memory manager having trouble maintaining required pages in memory, that is, memory size is too small to maintain the number of pages in main memory to allow the machine to function efficiently.

Symptoms

Memory Cycle Rate

The Memory Cycle Rate is a good indicator of the severity of memory pressure. If this rate is high (more than 25 cycles per hour) then memory manager is continuously looking for ROCs. This is a good indicator of lack of memory space as the memory manager is having trouble keeping a pool of 32 "free" pages.

Swapouts

Another good indicator of lack of memory space is the number of Swapouts that are occuring. It represents the number of times memory manager needed to swap a page from main memory to transient memory due to memory pressure.

Transient Page Faults

Transient page faults occur when a process is blocked in CPU due to the absence of an already allocated page in main memory. The required page has been swapped to transient memory. This is an indicator of memory manager not being able to handle workloads.

Solution

There are only two real ways to solve the problem of too little memory - increase the memory size or reduce the workloads. If memory pressure is accompanied with high allocation rates and file activity (see below), reducing these workloads may solve the problem.

High Page Allocations

A high level of memory allocations can place pressure onto the system. This pressure will in turn effect all of the other memory indicators. It will cause high CPU usage by memory manager and will place pressure on the need to maintain the "free" page pool.

Symptoms

Allocation Rate

Allocation rates show the number of page allocations being made per second This figure needs to be correlated with the other memory manager indicators to determine how the allocations are effecting the system.

Transient Memory Swapouts

If allocations are placing pressure on memory, you will also see high transient memory swapouts. This occurs as the "free" page pool is reduced and existing pages need to be swapped from main memory to transient memory to make room for the new pages.

Process Initiation and Logons

High allocation rates are usually the result of a high level of process initiation and logons as process data and code segments are initially placed into main memory.

Page Faults

Every time a page fault in the CPU occurs, a disc I/O is performed to load pages into memory. Reducing I/O activity reduces allocations.

Solution

An increase in memory size will help if there are also symptoms of memory pressure (see above). If there is no memory pressure, increasing memory size will not help as the problem lies not with finding additional memory space, but with the overhead in allocating many pages in a short interval.

Reduced process initiation and logons will reduce the number of allocations. This can be done through better process scheduling and a good menu system that utilizes process handling.

The only solution to high I/O activity is to reduce the amount of disc I/O that needs to be performed. This can be done through improving I/O related algorithms within programs, better system design, and rescheduling processes that generate a high level of disc I/O activity to run at quieter periods during the day.

Detecting The Cause Of Pressure

Global Activity

By looking at the allocation and page fault rates versus the transient memory activity, it can be determined if memory is under pressure due to lack of memory space or high allocation rates.

Firstly, look at the memory manager CPU activity. If the percentage of CPU utilised by memory manager is above 2%, then some pressure may be occuring. If it is above 5%, then this would indicate that there is definitely memory pressure.

To determine the source of the pressure, look at the allocation and page fault rates in relation to the transient memory activity. If allocations are moderate to low (that is, not much higher than found during less busy periods on the machine) and transient memory activity is high, then the pressure is due to a lack of memory space.

This can be verified by looking at the memory cycle rate. If this indicator is higher than normal, then lack of memory space is causing excessive memory manager activity.

If allocations and page faults are high when transient memory activity is not much higher than normal, then allocations are the source of the pressure. Usually, however, you will find that when allocation rates increase, this is accompanied by higher levels of transient memory activity as memory manager needs to make room for the new pages.

Process Activity

Process Allocations

By looking at process activity we can determine the effects of memory pressure and the possible sources of that pressure.

If the memory allocation rates are high, we need to look at processes to determine if the allocations are due to excessive initiations of processes or processes demanding additional pages during processing.

Memory Waits

The effects of memory management activity can be seen by looking at the memory related wait times that processes are experiencing.

Page Faulting

All file activity results in memory manager having to allocate pages in main memory. Processes that have high I/O activity will generate extra work for memory manager. How well memory manager is serving a process's I/O requests is indicated by the number of page faults that occur.

A page fault occurs when a process is blocked in the CPU due to the absence of a page in memory. The prefetch algorithm endeavors to minimize page faults by preallocating pages that it thinks might be required next. A process that has a high level of page faults is a) not being serviced well by memory manager and the prefetch algorithm, and b) is placing additional overhead on the system.

If many processes are experiencing high page faults, then memory manager may be a bottleneck. If a few processes are incurring page faults, then those processes are placing overhead on the memory manager.

San Diego Interex Paper 3222

The MPE XL System Debugger

Presented by Bob Green
Written by David J. Greer

Robelle Consulting Ltd.
Unit 201, 15399-102A Ave.
Surrey, B.C. Canada V3R 7K1
Phone: (604) 582-1700
Fax: (604) 582-1799

MPE XL comes with a powerful program debugger. For those of us who have struggled with Debug/V, there are many great features to look forward to. But, like all new software, there is a learning curve in understanding the new MPE XL debugger. Attempting to find the dozen or so most useful features in the three-inch stack of paper called the System Debugger Reference Manual is impossible, unless you have three spare months. In this article, I intend to summarize the features I've found most useful.

I am indebted to Stan Sieler of Allegro Consultants who taught me much of what is presented in this article. To obtain the maximum benefit from this article, you should try all of the examples that are presented. Having both a CM- and an NM-program available that calls the FWRITE intrinsic will make following the examples that much easier. If you are going to be a big user of Debug/XL, then you really have to have the reference manual. Order part #32650-90013, System Debug Reference Manual.

Native-Mode or Compatibility-Mode

HP was very kind to write not only a nice debugger for native-mode programs, but include features for debugging compatibility-mode programs too. When I first attempted to debug a CM-program, I got so confused that I returned to my classic HP 3000 where at least I knew all the command names. The next section will show Debug/XL commands for our old favorite Debug/V commands.

Debug/V Versus Debug/XL

Just like Debug/V, you can invoke Debug/XL by including the keyword "Debug" on the :Run Command for your program. Debug/XL responds with its "CM" prompt:

```
:run testprog;debug

CM DEBUG Intrinsic: PROG %6.3542
cmdebug >
```

If you have a Pmap (or Robelle's Qmap), you can set a breakpoint just as you would in Debug/V -- using *segment.offset*:

```
cmdebug >B 0.45
```

Anyone who has struggled with Pmaps knows how convenient it would be if the system debugger took advantage of the FPMAP information stored in program files. With this information it should be possible to set a breakpoint by **procedure name**. Debug/XL lets you either set a symbolic breakpoint at the first logical instruction in your procedure or at the more useful entry-point:

```
cmdebug >B open'input'file              {first instruction}
cmdebug >B ?open'input'file             {? implies entry point}
cmdebug >B open'input'file+255          {octal offset from first}
cmdebug >c                              {Continue = Resume}
```

Breakpoints

In the last ten years, I have probably lost over a month of time from one horrible default in Debug/V. In Debug/V, the Break Command would only break the first time it encountered the breakpoint, unless you added ":@" to the Break Command. In Debug/XL the default is to always break. Occasionally, you only want the breakpoint to be invoked once. Use ",-1" after the break location to have the breakpoint removed after one occurrence:

```
nmdebug >b ?open'input'file,-1 {break once}
```

Clearing Breakpoints

In Debug/V, the Clear Command disables breakpoints that have been set with the Break Command. In Debug/XL, use the BD (Breakpoint Delete) Command to remove breakpoints:

```
nmdebug >bd          {You will be asked for which one}
nmdebug >bd @        {Delete all breakpoints}
```

Setting a "Return" Breakpoint

One of the most useful breakpoints is the one immediately after a procedure call. Suppose that your program calls the procedure `extract_ready`. You want to know the result of `extract_ready`, so you would like a breakpoint in the calling code immediately after the call to `extract_ready`. You do the following:

```
cmdebug >b ?extract_ready
cmdebug >c
.
.
.                                  {break at extract_ready}
cmdebug >lev 1;b p,-1              {-1 means only break once}
cmdebug >c                         {continue execution}
```

The "lev 1" goes back to the previous logical level in the calling sequence (use "tr,d" to see a complete traceback). The "b p" sets a breakpoint at the compatibility-mode program counter. The "lev 1" places the program counter at the instruction *after* the one that called the current procedure. The ",-1" tells Debug/XL to execute the breakpoint once and then throw it away. Note that it's safe to use this breakpoint anywhere in the `extract_ready` procedure -- not just at the beginning.

What if we are in native-mode code (e.g., FWRITE)? Our return breakpoint won't work, since we called FWRITE from compatibility-mode. To set a return breakpoint in this case, first switch into cmdebug:

```
cmdebug >b ?FWRITE
cmdebug >c
.
```

```
.
.                                  {break at NM FWRITE}
nmdebug >cm                        {switch into CM}
cmdebug >lev 1;b p,-1              {set return breakpoint}
cmdebug >c                         {continue execution}
```

Abort Command -- Getting Out of Debug

You can terminate your program with the Abort Command. Use this any time that Debug/XL is prompting for commands. The Debug/XL Abort Command is similar to the Debug/V E@ Command.

Displaying Values

When I first used Debug/XL, I became totally confused about how to display the usual DB-, Q-, and S-relative values. It turns out to be very simple. In Debug/V the Display Command takes the register as a parameter. In Debug/XL there are separate command names for displaying values relative to each register. Here are the Debug/V and Debug/XL Display Commands:

Debug/V	Debug/XL
D DB	DDB
D S	DS
D Q	DQ

The Debug/XL Display Command has a count as its second parameter (just like Debug/V), but the display attribute is different. Here is the comparison:

Debug/V	Debug/XL	Description
,I	,# or D	Decimal
,O	,% or O	Octal (default in CM)
,H	,$ or H	Hexadecimal (default in NM)
,A	,S	Ascii/String

Instead of S, you can also use A for displaying string values. The A-option is closer to the A-option of Debug/V, but we find the S-option more useful.

By default, the S-option displays all characters you request and only displays the virtual address of the string once. If you want to see as many characters per line as possible, with each new line starting with the virtual address of the characters displayed, use this command:

```
cmdebug >dq 104,200,s,e            {"e" shows addresses}
```

Symbolic Machine Code

Our list above doesn't show you how to display the actual run-time machine instructions (commonly called decompiling). That's because Debug/XL has many excellent features to symbolically display code. While you can use the DC (Display Code) Command to show symbolic code, we have found a better method -- windows.

Symbolic Traceback

The Debug/V Trace Command was almost useless. You had to manually work through the segment numbers and offsets to figure out the true procedure names. The Debug/XL Trace Command produces a proper symbolic traceback of procedure names.

You can also use the traceback to observe switches from native-mode to compatibility-mode. For example, if you have SM capability you can set a breakpoint in any system SL or system XL routine. KSAM files come in two flavors: CM and NM. If you access a CM KSAM file from a NM program, MPE XL calls the CM FWRITE intrinsic. You can easily prove this to yourself by setting a breakpoint in the CM FWRITE intrinsic:

```
:run testprog;debug          {NM program to read CM KSAM file}
nmdebug >cmdebug             {switch to CM}
cmdebug >b ?FWRITE           {question-mark for entry point}
cmdebug >c                   {continue execution}
.
.
.                            {note the ",d" on the TR Command}
cmdebug >tr,d                {traceback showing switches}
```

Compatibility-Mode Windows

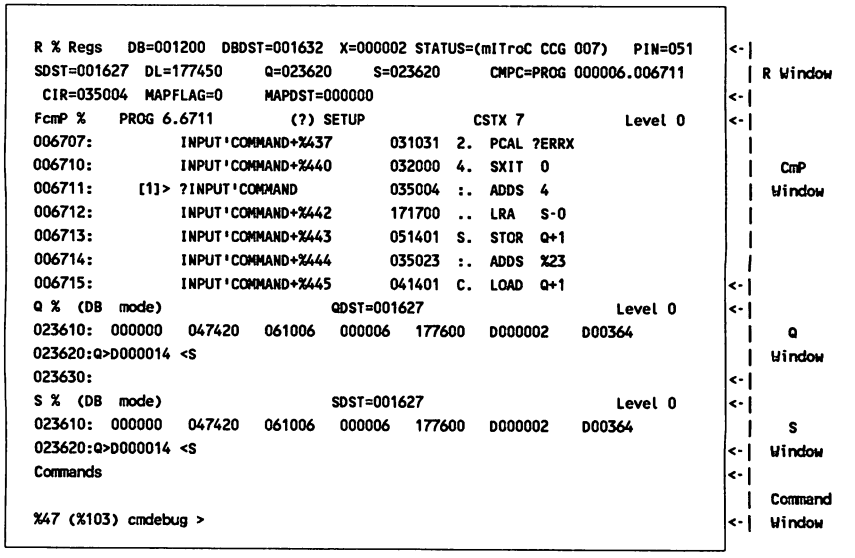
The WON Command is the real power of Debug/XL. WON is short for Windows On. When you turn windows on, the top portion of the screen is reserved for a symbolic display of the currently executing code, another portion displays register and/or stack values, and the bottom of the screen is used to enter commands. This is a very powerful feature.

CM Window Example

The following is an example compatibility-mode window. We first set a breakpoint, continue to that breakpoint, and finally we turn windows on.

```
cmdebug >b ?input'command        {break at the entry point}
cmdebug >c                       {continue execution}
.
.
.
cmdebug >won                     {turn windows on}
```

The top three lines of the display show the register information:



For most of us, only the DL=, Q=, S=, and X= values are interesting. If the DBDST and the SDST (the DB- and S- data segments) are different, you are in split-stack mode. Line four shows that we are currently at location 6.6711 in the program. The PROG would change if the breakpoint was inside an SL. Next we see seven instructions. The "[1]" means breakpoint number 1. The ">" symbol next to "?INPUT'COMMAND" shows the next instruction to be executed. The bottom of the display shows the values around the Q- and S- registers. In our example, the Q and S registers are the same so the Q- and S-displays are identical. Finally, you are prompted for more Debug/XL commands.

Single-Stepping

One other command adds a lot of power to windows: S -- single-stepping. The S Command executes the next instruction, then returns control to Debug/XL. After the execution, register and stack values are updated and any changed values are highlighted. Because the compatibility-mode window shows the top few words of the stack, you can often get an instant picture of what is going on. Here is the first window after executing one single-step:

```
cmdebug >s                    {single-step}
```

```
R % Regs   DB=001200  DBDST=001632  X=000002 STATUS=(mITroC CCG 007)   PIN=1
SDST=001632  DL=177450     Q=023620     S=023624      CMPC=PROG 000006.
 CIR=171700  MAPFLAG=0     MAPDST=000000
cmP %    PROG 6.6712        (?) SETUP            CSTX 7            Level 0
006707:          INPUT'COMMAND+%437       031031  2.  PCAL ?ERRX
006710:          INPUT'COMMAND+%440       032000  4.  SXIT  0
006711:     [1]  ?INPUT'COMMAND           035004  :.  ADDS  4
006712:        > INPUT'COMMAND+%442       171700  ..  LRA   S-0
006713:          INPUT'COMMAND+%443       051401  S.  STOR  Q+1
006714:          INPUT'COMMAND+%444       035023  :.  ADDS  %23
006715:          INPUT'COMMAND+%445       041401  C.  LOAD  Q+1
Q % (DB  mode)                  QDST=001632                       Level 0
023610:  000000   047420   061006   000006   177600   000002   00364
023620:Q>000014   000002   006712   062007
023630:
S % (DB  mode)                  SDST=001632                       Level 0
023610:  000000   047420   061006   000006   177600   000002   00364
023620:Q>000014   000002   006712   062007<S
Commands

%47 (%103) cmdebug >
```

The ">" symbol has moved forward by one instruction. The register values have been updated and the top of stack has changed because we added four to the S-register.

Set CRON

This sounds like the title of a futuristic movie, but when combined with single-stepping it can be very powerful. Once you start using the S (Single-Step) Command, you'll find yourself typing it a lot, especially when debugging NM programs where you have a lot more instructions per source code statement. Fortunately, the Debug/XL designers already thought of this. When you Set CRON, hitting Return tells Debug/XL "execute the last command that I typed". This is most useful when your last command was S, but it applies to **any** command:

```
cmdebug >set cron          {Return = last-command}
cmdebug >s                 {single-step}
cmdebug >                  {another single-step!}
cmdebug >                  {and one more}
cmdebug >                  {and so on}
```

Multiple Steps

While single-stepping is useful, it can be very slow. You can step through a program faster using multiple instructions for every step. The following example shows how to step through every seven executed instructions. Note: you must have a space after the Step Command and before the number of instructions to execute (e.g., "S7" is invalid):

```
cmdebug >set cron          {Return = last-command}
cmdebug >s 7               {execute seven instructions}
cmdebug >                  {another seven!}
cmdebug >                  {and seven more}
cmdebug >                  {and so on}
```

Native-Mode Debugging

Much of what has been discussed applies to native-mode. There are a few minor differences:

1. You don't need to specify Fpmap (or any other magic parameter) on the :Link Command. Procedure name and location information is automatically included in all NM program files.

2. Since the first instruction of a procedure and its entry point are the same, you never need to use a question mark. If you happen to type a question mark, Debug/XL may not print an error. In this case, you will have set a breakpoint in a stub procedure. Since you almost never want to do this, it's important to remember not to type the question mark before the procedure name.

3. In most programming languages, any separators (e.g., apostrophes) used in procedure names will now become underbars.

Here is our previous breakpoint example in native-mode:

```
nmdebug >b open_input_file      {break at procedure entry}
```

Case Sensitivity

It is easy to see that portions of MPE XL were affected by UNIX and the C programming language. In UNIX and C, case is significant (i.e., upper-case and lower-case are not the same). When setting breakpoints in native-mode code, it is important to remember this. Most MPE XL routine names are in upper-case. The most well-known exceptions are all of the IMAGE and VPLUS intrinsics which are in lower-case. The following example results in a Debug/XL error:

```
nmdebug >b fwrite               {not found; lower-case}
```

Switching Modes

Sometimes you want to switch between CM-debug and NM-debug. For example, the NM-FWRITE intrinsic calls the CM-FWRITE intrinsic for certain types of files (e.g., circular). These commands would set breakpoints in both the CM- and NM-FWRITE intrinsics:

```
nmdebug >b FWRITE               {NM-FWRITE breakpoint}
nmdebug >cm                     {switch into cmdebug}
cmdebug >b ?FWRITE              {CM-FWRITE breakpoint}
cmdebug >nm                     {switch back into nmdebug}
nmdebug >c                      {continue execution}
```

Native-Mode Windows

The WON (Windows On) Command is just as powerful in native-mode as in compatibility-mode. The display is different -- instead of the old familiar DB, S, and Q registers, there is a strange group of 32 "general-purpose" registers. The code looks a lot different too -- those famous RISC instructions instead of our old faithful Classic 3000 ones.

NM Window Example

We will show an example native-mode window, by setting a breakpoint for the FWRITE intrinsic:

```
nmdebug >b FWRITE                (requires SM capability)
nmdebug >c                       (continue execution))
.
.
.
nmdebug >won                     (turn windows on)
```

```
GR$   ipsw=0006fe0f=jthlnxbCVmrQPDI  priv=0  pc=0000000a.004a5fc0  pin=0000007a  <-|
r0  00000000 40100e20 004aee30 00000001 r4  c0000000 0000ffff 4033292a 00000000    |   R
r8  00000001 00000009 00000004 4034a880 r12 00000000 00000000 00000000 00000000    |  Window
r16 00000000 00000000 00000000 c0000000 r20 c0000000 00000001 85240000 00000314    |
r24 40332604 000000d0 00000001 c0202008 r28 00000001 ffffffff 4034afd8 004aee30  <-|
nmP$  SYS   a.4a5fb8  NL.PUB.SYS/FSPACE+$5a4                    Level   0,0   <-|
004a5fb8:          FSPACE+$5a4          e840c000  BV      0(2)                  |
004a5fbc:          FSPACE+$5a8          4fc33d31  LDWM    -360(0,30),3          |   NmP
004a5fc0:    [1]> FWRITE               6bc23fd9  STW     2,-20(0,30)           |  Window
004a5fc4:          FWRITE+$4            6fc30340  STWM    3,416(0,30)           |
004a5fc8:          FWRITE+$8            6bc43cc9  STW     4,-412(0,30)          |
004a5fcc:          FWRITE+$c            6bc53cd1  STW     5,-408(0,30)          |
004a5fd0:          FWRITE+$10           6bc63cd9  STW     6,-404(0,30)        <-|
Commands                                                                      <-|  Command
                                                                                |  Window
$7 ($1d) nmdebug >                                                            <-|
```

The first line contains general information about the process (e.g., the pin number). The pc= is the program counter (notice it's a full 64-bit address in *space.offset* format). Lines two through four of the display show all 32 general-purpose registers. The fifth line shows where the first instruction in the window is located (in NL.Pub.Sys @ FSPACE:$5a4). The native-mode instructions are shown, along with the breakpoint number "[1]" and the next instruction to be executed is marked with the ">".

There are two commands that can be a big benefit in examining the code "around" a breakpoint: PB (Program Back) and PF (Program Forward).

Paging

Debug/XL windows have to display all their information in the twenty-four lines on a standard terminal screen. By default, the size of the symbolic instruction list is seven instructions. Especially when you are single-stepping through instructions, it is very useful to see the previous seven instructions or the next seven. The PB (Program Back)

Command displays the previous seven instructions and the PF (Program Forward) Command shows the next seven. While seven instructions is the default, there are commands to change the size of the program window. If you have changed the size, Program Back and Program Forward adjust themselves to the new window size.

```
nmdebug >pf          {program forward}
nmdebug >pb          {program back}
```

PL Command

If you want to change the number of program instructions on the screen, use the PL Command (Program List). The PL Command assumes that the number of lines you want is in the current base. Therefore, PL 10 means 16 instructions in NM Debug and 8 instructions in CM Debug. To get around the problem, we always specify the number of instructions in decimal:

```
nmdebug >pl #10      {show "ten" instructions}
```

Native-Mode Procedure Parameters

Long-time users of Debug/V know how to anticipate where procedure parameters will be located. For example, if we had a procedure with this declaration:

```
integer procedure convint(buf,len);  !result = Q-6
   value   len;
   integer len;                      !len    = Q-4
   byte array buf;                   !@buf   = Q-5
```

In CM-debug, we would look at the parameters as follows:

```
cmdebug >dq -6              {result of Convint procedure}
cmdebug >dq -4              {length of buffer}
cmdebug >dq -5              {address of buffer}
Q-%5     % 000104           {must use this value below}
cmdebug >ddb 104/2,10,s     {print actual buffer contents}
```

Notice that we had to divide the value at Q-5 (i.e., %104) by two, since the buffer was passed as a byte address. In native-mode, this irritation disappears (except for those using SPLash! to emulate Classic byte addressing).

Native-Mode Calling Conventions

With the power to set breakpoints symbolically, by just knowing the name of a procedure, there is even more incentive to be able to guess the location of procedure parameters. NM procedures are allocated registers for the first four parameters, but they are allocated left-to-right -- the opposite of CM procedures. The first parameter is assigned to Register-26, the second to Register-25, the third to Register-24, the fourth to Register-23, and any remaining parameters are stored on the NM stack. The return value is in Register-28 (and Register-29 for 64-bit values). For native-mode, you would think of the declaration for Convint as:

```
integer procedure convint(buf,len); !result = R28
   value   len;
   integer len;                     !len    = R25
   byte array buf;                  !@buf   = R26
```

If you have windows on, the 32 general-purpose registers are always displayed. The only problem area is the buffer parameter:

```
nmdebug >b convint          {note lower-case}
nmdebug >won                {windows on}
nmdebug >c                  {continue execution}
.
.
.                           {debug breaks @ convint}
nmdebug >=r25               {display the length}
nmdebug >dv r26,10,s        {display virtual uses the contents}
                            {of register 26 as an address}
```

Variables

Debug/XL contains a programming language. We won't try and cover all of the features of this language, but variables are so powerful that they are worth knowing about. In our example with the Convint procedure, suppose that the buffer you are passing to Convint is a global variable. Setting the breakpoint at Convint gives you a convenient method to find and save the address of your buffer so that you can use it at any breakpoint.

```
nmdebug >var buf_var=r26   {save address of buffer}
nmdebug >dv buf_var,10,s   {display buffer contents}
nmdebug >c                 {continue execution}
.
.
.                          {sometime much later ...}
nmdebug >dv buf_var,10,s   {display the buffer contents}
```

The final Display Virtual Command displays the contents of the buffer using the address that we saved. When the breakpoint takes place, we may have no convenient way of finding the program variable that has the address of our buffer. Because we have saved the address in the Debug/XL variable "buf_var", we display the buffer contents without knowing where the address is stored.

Virtual Addresses

So far, we have assumed that all addresses are 32-bits. In MPE XL, addresses are actually 64-bits. Debug/XL shows these addresses as *space.offset*. If you are working with mapped files, you will find that the full 64-bit address suddenly becomes important. The following example opens a file with mapped access, saves the virtual address of the file into a variable, and then displays the actual contents of the file.

```
nmdebug >map "file1.suprtest"    {open an mpe file mapped}
nmdebug >var fileaddr = mapva("file1.suprtest")
nmdebug >=fileaddr               {display the virtual address}
```

Debug/XL has a built-in calculator that accepts any Debug/XL expression. You invoke the calculator with an equal sign "=". Debug/XL evaluates the calculator expression and prints the result. The calculator will display the full 64-bit address of "file1.suprtest" as *space.offset*.

You can display the actual contents of the file:

```
nmdebug >dv fileaddr,20,s      {first 20-bytes of file}
```

Warning: Due to a very serious bug in MPE XL, never, never, never do this on the file Catalog.Pub.Sys. If you open Catalog.Pub.Sys with mapped access, you will cause a system failure.

The map command displays the virtual address of a file in *space.offset* format. You can use the DV (Display Virtual) Command to display the file contents or you can use our method. We prefer using a variable and mapva function, since typing in a full 64-bit address correctly is quite difficult.

Cseq.Pub.Nuggets

While it is easy to predict the layout of parameters in our simple example, things can get more complicated in MPE XL. For example, addresses can be passed as 64-bit quantities instead of the default 32-bit values. The best way I've found to determine parameter location is to use the Cseq (calling sequence) utility in the Nuggets collection (available from Software Research Northwest 206-463-3030). Here is the Cseq output for the FWRITE intrinsic:

```
Procedure FWRITE (
        Parm  1:            int16    ;   {R26, bits =  16}
        Parm  2: anyvar     record   ;   {(skip 25) R23, R24}
                                         {bits = 65536}
                                         {Address type = LongAddr}
        Parm  3:            int16    ;   {SP-$0032, bits =  16}
        Parm  4:            UInt16   )   {SP-$0036, bits =  16}
      uncheckable_anyvar
```

Note that the buffer parameter is a "LongAddr" that is passed in both R23 and R24 (the first is the space and the second is the offset). Fortunately, it is still easy to see the contents of the buffer. If we were at a breakpoint at the start of FWRITE, we would display the buffer with:

```
nmdebug >dv r23.r24,20,s        {display buffer contents}
```

Integers: 16-bit versus 32-bit

Cmdebug displays integers in octal as 16-bit quantities. Nmdebug displays integers in hex as 32-bit quantities. In our FWRITE example, it is easy to see the value of the length parameter.

```
nmdebug >dv sp-32,1         {display the length}
$ 00005f00
```

We used the DV (Display Virtual) Command to display the stack contents. The ",1" is not necessary - it's the default, but we have shown it to make the following examples a little clearer. The "dv sp-32" displays the value at sp-32 as a 32-bit quantity, but we know that the actual value of FWRITE's length parameter is a 16-bit quantity. You can display two 16-bit integers using the following:

```
nmdebug >dv sp-32,1,,,2   {display two 16-bit integers}
$ 0000 5f00
nmdebug >dv sp-32,1,#,,2  {display two integers in decimal}
#       0   24320
```

Display Virtual always rounds down to a virtual address that is a multiple of four and then displays one or more 32 bit words.

Miscellaneous Tips

Setting a "Return" Breakpoint

We showed how to set a return breakpoint in compatibility-mode. You use a similar method to set a return breakpoint in native-mode code:

```
nmdebug >b extract_ready
nmdebug >c
.
.
.                                {break at extract_ready}
nmdebug >lev 1;b pc,-1           {-1 means only break once}
nmdebug >c                       {continue execution}
```

The only difference between a CM return breakpoint, and an NM one, is the name of the program counter. In native-mode it's called "pc". This sets a return breakpoint immediately after the code that called `extract_ready`. Note that it's safe to use this breakpoint anywhere in the `extract_ready` procedure -- not just at the beginning.

Debugging Batch Programs

In Debug/V, there was no practical way to debug a program running in batch. In Debug/XL, you can debug a batch program on the console, although it's a bit messy to set up. You have to do these steps:

1. Obtain the pin number of the program you want to debug. You'll need to use a program like Shot.Pub.Nuggets. You can use the Showproc Command, if you have MPE XL version 2.1 or later versions.

2. Go to the console and insure that there will be no output on the console. The easiest way to do this is to initiate a :Restore on the console. This assumes that your tape drive is not configured for auto-reply. Do not reply to the tape request.

   ```
   {On the console ...}
   :hello user.acct
   :restore
   ```

3. On another terminal, log on with SM capability and enter debug. For example,

   ```
   :hello manager.sys
   :debug
   ```

4. Once you are inside Debug, you must set an environment variable and force a breakpoint in the batch program. Our example assumes that the batch program will call the FWRITE intrinsic:

   ```
   nmdebug >env job_debug true     {set special variable}
   nmdebug >b FWRITE:pin#          {don't forget the pin#}
   ```

You don't actually type "b FWRITE:pin#" when setting the breakpoint. You substitute the actual pin# that you obtained in step 1 (e.g., "b FWRITE:*103*").

When the batch program encounters the breakpoint, Debug/XL is invoked and all

Debug/XL input/output is done via the console. On the console you can type any of the usual Debug/XL commands. When you finish your debugging session, you'll need to remember to abort the :Restore that you initiated. You must also return to the Manager.Sys session and disable job debugging:

```
cmdebug >env job_debug false
```

Macros

Debug/XL contains a small programming language that lets you create your own macros. Debug/XL has no command to skip over procedure calls, although almost all PC-based debuggers have this feature. When single-stepping through a program, you rarely want to single-step through external procedures (e.g., the Print intrinsic). Use the *j* macro to jump over the next native-mode BL instruction. Macros use braces for the body of the macro (i.e., as begin/end), so don't interpret the braces as comments. Here is how to declare the macro:

```
nmdebug >mac j {b pc+$8,-1; c}
```

Macros are declared with the Mac Command. The first parameter to the Mac Command is the macro name (in this case it's *j*). The body of the macro follows and is surrounded with braces. Macros can take several lines. The *j* macro sets a breakpoint at the next native-mode instruction after a branch-and-link "pc+$8". The breakpoint is only executed once ",-1". Multiple commands are separated by semi-colons ";". The last step of the macro is to execute the Continue Command "c". Note that the *j* macro is only useful around branch-and-link instructions which is why we jump eight bytes ahead of the program counter instead of four. You execute the macro as if it were a built-in Debug/XL command:

```
nmdebug >j
```

Vfilepages Macro

When doing any performance measurements with disc files, you need to know what portion of the file is in memory. This macro takes advantage of many features of Debug/XL. The macro displays the number of pages of a file that are currently present in virtual memory.

```
/*  Macro:   Vfilepages
/*
/*  Purpose: Display the number of pages (and
/*           sectors) of a file that are in memory.
/*
/*  Warning: Never use this macro on catalog.pub.sys.
/*

mac vfilepages (filename:str) {
   map !filename;
   w !filename " contains ";
   w vainfo(mapva(!filename),"pages_in_mem"):"D";
   w " pages in memory = ";
   w vainfo(mapva(!filename),"pages_in_mem")*#16:"D";
   w " sectors";
   wl;
   unmap(mapindex(!filename));
   }
```

Lines starting with "/*" are treated as comments. The "filename" is a parameter to the macro and it's of string type. To understand the rest of the macro requires looking up the description of the Map, Mapva, W, WL, and Unmap Commands and an understanding of the Vainfo and Mapindex Function. We'll leave that up to you. To invoke this macro, you would do the following (note the quotes around the filename):

```
vfilepages "file50.suprtest"
file50.suprtest contains 8 pages in memory = 128 sectors
```

Warning: Because this macro uses the Debug/XL Map Command, do not use it on the file catalog.pub.sys. If you do, you will cause a system failure.

DBUGINIT File

Once you start writing macros, you will want to have them automatically loaded when you enter Debug/XL. Debug/XL always executes a use-file called DBUGINIT. Debug/XL first looks for this file in the same group and account as the program, then it looks in the logon group and account. Rather than fill our DBUGINIT file with macros, we fill it with Use Commands for different files that contain useful macros: You can use :file commands for the DBUGINIT file, but you must use a fully qualified filename. For example:

```
:hello david.dev,david
:print dbuginit.macro.dev
use splash.macro.splash
use macros.macro.dev
:file dbuginit.david.dev=debuginit.macro.dev
:run testprog;debug        {Debug/XL will use debuginit.macro}
```

:Setdump Command

Classic MPE contains a :Setdump Command, but I believe most of us ignored it because the traceback it printed was not symbolic. If you enable :setdump in MPE XL, you not only get an excellent symbolic traceback, but in native-mode you are placed into Debug/XL (certain exceptions apply to privileged-mode programs).

MPE Commands

You can enter almost any MPE command by preceding it with a colon. This includes UDCs and the :Run Command. Often in the middle of a debugging session, you need to examine your source code. An easier way to do this is to run your editor from within Debug/XL. One word of caution -- Debug/XL, like many HP products, fails to see if a son process has terminated or suspended.

We also find it useful to invoke Cseq.Pub.Nuggets when we are debugging a program. This lets us determine the location of the parameters for any MPE intrinsic:

```
nmdebug >:cseq.pub.nuggets      {obtain parameter addresses}
```

Running Qedit from Debug/XL

If you invoke Qedit from Debug/XL, be sure to run it with Parm=32 (this tells Qedit not to suspend on exit). The most likely reason to invoke Qedit from Debug/XL is to examine your source code. If you do not /Shut your file before running your program, you will get "Error: Busy file" when you try to open your file inside Qedit (inside Debug/XL). To get around the problem, you can either /List your source code or /Text a copy.

Conclusion

If you are going to make heavy use of Debug/XL, I strongly recommend getting the System Debug Reference Manual. While it's not helpful for learning Debug/XL, it's invaluable in looking up specific commands and their syntax. That part number again is #32650-90013.

When I first set out to write this article, I thought that it would only take me a few paragraphs to convey what I'd learned about Debug/XL. If you've got this far, you realize that I underestimated the amount of material -- not surprising given the rich feature set of Debug/XL.

TITLE: Understanding CASE

AUTHOR: Karen Heater

Infocentre Corporation

7420 Airport Road
Suite 201

Mississauga, Ontario L4T 4E5
CANADA

(416) 678-6880

FINAL PAPER NOT AVAILABLE, HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

PAPER NO. 3223

COBOL85 On XL Machines

We've Got A Language!

Rick Hoover

CIV Software
700 Hanover Dr.
Shelby, N.C. 28150

The first question I asked myself was, 'Why a paper on COBOL? Does anybody care anymore? After all, COBOL has been around so long. Would anybody read this treatise?'

That was my thought in 1990 when I presented a paper at the Boston INTEREX about COBOL85 and HiLi. I had completed a consulting job on my first XL machine and spent a lot of time leaning the new capabilities of XL. All of my programs were written using the standard COBOLII commands, creating a USL and prepping the USL into an executable program. Part of the way into the project I found the COBOL compiler commands to compile a COBOLII program into native mode. My first compiles scared the heck out of me. Something had to be wrong! Compiles on the 950 that took 25-40 seconds were now taking 2-3 seconds.Something's not working. Were was the USL file? HELP!!!

My fears were put to rest as I completed the project. Native mode is good. I can compile programs faster and create object files that can be linked to XL files and programs can use multiple XL files and ....

This also led to a day when I sat down with a manual called HiLi. I only knew this term as a game played mainly in Florida. I found out that HiLi was HP's answer to VIEW macros. While thumbing through the rest of the manuals I stumbled across the one manual I would have never thought to look at. But there it was. COBOL. I've been a COBOL programmer now for over a dozen years. What new could I learn from this book. Maybe there was something I didn't know about COBOL on the XL. I began my search.

There were many grey enhanced sections in the manual. What were these? I later found out that COBOL had grown up. It was now COBOL, COBOLII and COBOL85. Back in the early 80's when I moved from COBOL to COBOLII, well let's just say that I wasn't running around yelling from the rooftops, 'WOW LOOK AT THIS...LOOK WHAT I CAN DO NOW!'. But since I found out about COBOL85's enhancements, all I can say is 'WOW LOOK AT THIS...LOOK WHAT I CAN DO NOW!'.

This paper is in no means a complete reference to the enhancements to COBOL. Rather, this paper is meant to show the people out there that are still using COBOLII that there are some really neat things you can do with the COBOL85 compiler. Let me also state that the COBOL85 compiler is (1) on both the Classic and Spectrum machines and (2) you probably already have it and just don't know it. COBOL85 is in the COBOLII.PUB.SYS file. There are entry points to the COBOL85 commands. There should be a file called COBUDC.PUB.SYS that contains the routines to compile using the COBOL85 entry points (this is for Classic users). And, if you own COBOLII, you were automatically upgraded to COBOL85.

I will be discussing each division separately in describing the enhancements available.

Finally, I will be throwing in some of my techniques throughout the paper. These techniques work for me. They may not work for you. So be it. There will also be a few comments made about certain things you can do in COBOL that you may already know. I spent a couple of years doing training classes across the country for a software company. The comments that I will be making in this paper were second nature to me, but I found that there were many programmers that didn't know about the things that I talked about. I will explain later.

IDENTIFICATION DIVISION

This is the shortest section in the whole paper. With the exception of the **PROGRAM-ID**, all paragraphs are obsolete in COBOL85. Well, that's nice but what else is there? Well, there really wasn't that much to begin with. There is one enhancement to the **PROGRAM-ID** paragraph. You can say:

PROGRAM-ID. ***MYPROGRAM*** **IS INITIAL PROGRAM.**

This is the same as saying **$CONTROL DYNAMIC.**

ENVIRONMENT DIVISION

This division is now completely optional. There are a few new items that are well worth mentioning in this division.

SYMBOLIC CHARACTERS ***BELL*** **IS 8.**

DISPLAY "YOU ARE NOT ALLOWED TO DO THAT" BELL

The **SYMBOLIC CHARACTERS** statement allows the programmer to send any kind of special character (such as line feed, carriage return, etc.) to a statement. This can allow for a stronger message or a more impressive formatted message.

CLASS *A-GOOD-GRADE* IS "A", "B", "C"

IF GRADE-ASSIGNED IS A-GOOD-GRADE DISPLAY "GOOD"

There is now a **CLASS** statement that will let the programmer set up specific logic flow in the program. These **CLASS** statements will act in a mode similar to 88 levels except they become program specific. They also compare a variable to a known value.

SELECT OPTIONAL *MYFILE* ...

You can now not only have the **OPTIONAL** keyword but, if the file is not present and opened in I-O or extended mode, a new file will be created.

MEMORY SIZE, SEGMENT-LIMIT and **MULTIPLE FILE** statements are now obsolete.

DATA DIVISION.

This is where we will separate the ultra fancy from the new. I am now hard at work (or is that hardly working) on a paper about local and external fields and programs in COBOL85. I will not be discussing external programs and fields at this time. Let's just say that for those of you who work with languages like PASCAL and C, you will find this to be just to your liking.

When COBOL came out there was a term used in the DATA DIVISION called FILLER. Programmers got sooooo tired of typing in FILLER all the time. COBOLII came along and said "Those of you who are tired of FILLER may now type in F instead. So we went from:

```
05 FILLER    pic x(02).
```

to

```
05 F         pic x(02).
```

Today, we don't even need F:

```
05           pic x(02).
```

There are two new USAGE verbs available. **BINARY** is the same as **COMP**. **PACKED-DECIMAL** is the same as **COMP-3**.

PROCEDURE DIVISION.

Now we come to the fun zone. This is where it's happening. There are so many new concepts in the PROCEDURE DIVISION, it becomes hard to know where to start. Here goes...

ACCEPT *TODAYS-DAY* FROM DAY-OF-WEEK

This statement will set the field TODAYS-DAY to 1 if the day of week is Monday, 2 if Tuesday etc.

DISPLAY "PLEASE TYPE IN YOUR NAME:" NO ADVANCING

This form of the DISPLAY statement will leave the cursor immediately after the : in the DISPLAY statement.

This next statement is one of the more powerful statements for COBOL. We COBOL programmers finally have a CASE statement (and a very powerful one also).

```
EVALUATE MYFIELD-1 ALSO MYFIELD-2
      WHEN "A" ALSO "B" PERFORM PARA-A
      WHEN "C" ALSO "D" PERFORM PARA-B
      WHEN "E" ALSO ANY PERFORM PARA-C
      WHEN OTHER PERFORM PARA-D
END-EVALUATE

EVALUATE CHECK-THIS > 0 ALSO SOMETHING-ELSE
      WHEN TRUE ALSO "A" PERFORM PARA-F
                         PERFORM PARA-G
                         MOVE THIS-FIELD TO THAT--FIELD
      WHEN FALSE ALSO "A" PERFORM PARA-H
      WHEN FALSE ALSO ANY PERFORM PARA-I
END-EVALUATE
```

The EVALUATE statement sets up 1 or 2 fields to be checked. If the field is actually a condition (as in the second example) the programmer can use the terms TRUE and FALSE to control processing. The EVALUATE statement flows through the WHEN statements until the conditions are satisfied. If the conditions are not satisfied and the WHEN OTHER is available, control will pass down to the WHEN OTHER otherwise control will fall out of the EVALUATE. Once a condition is met, the rest of the WHEN statements will be ignored. EVALUATE can work with strings, characters and numerics.

There was one other statement that you may have noticed in the examples. That statement was the END-EVALUATE. Many of the COBOL verbs allow control via an END-verb statement. I will demonstrate:

```
IF A > B
        IF C > D
                IF E > F
                        MOVE X TO Y
                        IF G = H
                                CONTINUE
                        ELSE
                                MOVE Q TO Z
                        END-IF
                END-IF
        MOVE M TO N
        END-IF
END-IF
```

In the above example I have created a nested loop that has 2 new wrinkles that could not have been accomplished before. The first thing you may notice is that there is a new verb CONTINUE hiding in the code. This allows you to make positive statements rather than negative statements about the logic flow. You will not have to say IF A NOT = B to complete a phrase. The other thing you may have noticed is that I can do certain things in the IF statement based on a condition without having to leave the nested IF to PERFORM a paragraph. And I can also do something after the IF's have been resolved.

I have very few IF constructs that were not easily changed over to this new top down format. The END-IF competes the associated IF. Be sure to always have a consistent IF...END-IF link.

INITIALIZE *MYFIELD-1 MYFIELD-2*

This statement will let the programmer set records or fields to an initial state prior to moving data to them. This is a lot better than moving spaces to all alphanumeric fields and zeros to all numeric fields. You can also initialize fields to a set value by saying:

```
INITIALIZE MYFIELD-A
        REPLACING ALPHANUMERIC BY "A"
                  NUMERIC BY "9"
```

In-line PERFORMs are now available. The program does not have to leave the flow to do specific steps. By simply stating:

```
PERFORM VARYING NUMBER-FIELD FROM 1 BY 1
        UNTIL NUMBER-FIELD > 10
        INITIALIZE TABLE-ARRAY(NUMBER-FIELD)
END-PERFORM
```

The above example will initialize an array called TABLE-ARRAY. If you leave off the name of a paragraph, the assumption of an in-line perform will be done. You can also nest in-line performs as in:

```
PERFORM 10 TIMES
        PERFORM UNTIL END-OF-FILE
                ADD 1 TO TOTAL
                ...
        END-PERFORM
END-PERFORM
```

The above code example shows how to construct a nested in-line perform. There is one other touch you can give the in-line perform:

```
PERFORM WITH TEST AFTER UNTIL A > 10
        ...
END-PERFORM
```

The above example has the phrase WITH TEST AFTER which states that the code inside the in-line perform will be run at least once The conditional phrase will be checked after the perform is executed. By default, the PERFORM has an implicit WITH TEST BEFORE.

The last example in the PROCEDURE DIVISION that I will go over is the READ statement since it has a couple of niceties that go over a lot of other verbs:

```
READ MYFILE
     AT END
          code
     NOT AT END
          code
END-READ
```

This makes for logical programming. You can now control most of the verbs with a NOT statement that will control program flow. The above example allows you to place a read statement inside of a performed paragraph that can also contain if statements that can also contain performs that can also contain...

Needless to say the effect is wonderful. The verbs that have this capability are:

WRITE	**READ**	**RETURN**	**DELETE**	**READ**
REWRITE	**START**	**CALL**	**STRING**	**UNSTRING**
ADD	**COMPUTE**	**DIVIDE**	**MULTIPLY**	**SUBTRACT**
ACCEPT				

This allows control over all phases of these verbs. The ability to control a verb if there is an overflow condition has always been available but the ability to say that if an overflow condition occurs, do this otherwise do that and not have to worry about period placement makes the logic flow a lot cleaner.I've found that during the course of developing programs, I can greatly increase the readability of any program by using these techniques.

As I stated in the beginning, this is not meant to be a complete treatise on COBOL85. But for those of you who use the language, try some of these techniques the next time you work on a program. You will find that there is a lot to be gained.

Creating Seamless Packages Through Process Handling

John P. Korb
Paper 3226

Innovative Software Solutions, Inc.
10705 Colton Street, Fairfax, VA 22032
(703) 273-5025

When a company purchases an HP 3000 it receives the FOS (Fundamental Operating System) and a number of utilities. Over time additional software is purchased from HP and other vendors. Often, however, many tasks which could be accomplished by the combined use of several utilities are judged "will require extensive programming" or "not possible with present resources" and so a user requirement goes unfulfilled.

Process handling and the interfacing of utilities to each other to form a problem solution are two often neglected topics. This paper will discuss some of the applications of process handling and the combining and interfacing of utilities in solving problems.

What is Process Handling

Process Handling is one of the MPE "special capabilities". A simple description of Process Handling might be the ability of one program to start up one or more programs during its own execution.

The MPE operating system considers each and every unique execution of a program a "process." When the system operator starts up MPE a process is started which loads MPE into memory and starts up the various processes which MPE needs to function at a minimal level. Thus, MPE itself uses Process Handling to build and control itself. When MPE starts up it builds processes for spooling, memory logging, device recognition, controlling user processes (sessions and jobs), loading programs into memory, and a number of other functions. When a person signs onto the system MPE creates a process which becomes the "main" part of the user's job or session. This "user main" process is the command interpreter which supplies the colon prompt and accepts MPE commands. When the user enters a "RUN" command or invokes a subsystem (EDITOR for example) the "user main" process uses process handling to create a process for the program or subsystem to be run.

Process Relationships

Processes are related to each other and have a sort of genealogy. With the

exception of the first process within MPE, every process has a "father" process which created it. Processes which share a common father process are "brother" processes. When a process creates another process, the process created is a "son" process. Thus, when the user enters a "RUN" command at the colon prompt, the "user main" process creates a "user son of main" process for the program or subsystem to be run.

The "user main" or "command interpreter" process only allows one son process at a time. This is why if the user enters a "RUN" command to run a program, presses "BREAK" to interrupt the program's execution, and enters another "RUN" command without aborting the first program, MPE will respond with "COMMAND NOT ALLOWED IN BREAK". This restriction is a limitation of the command interpreter and the "RUN" command and not a limitation of process handling. In fact, within MPE a number of processes are running simultaneously - the spooler processes and data communication processes for example. A programmer with access to process handling can often have multiple son processes executing concurrently.

Another restriction imposed upon "user main" is that the command interpreter and the son process cannot be executing at the same time. Either the command interpreter is active or the son process but not both. Again, a programmer can set up a program which executes a son process while the father process continues to run. This parallelism adds greatly to the capabilities and usefulness of process handling.

Just as human genealogy is often plotted as a "family tree", process genealogy is often plotted as a "process tree". In a process tree the "user son of main" is the trunk of the tree, its sons are the major limbs, off them sprout their sons, etc. There can be a considerable number of generations of processes in a process tree, but practically there are typically two to five generations.

Processes are dependent upon their fathers for their survival. If a process aborts (or terminates normally), all its son (and grandson, and greatgrandson, etc.) processes terminate.

Creating A Son Process

A programmer starts a process very much as the MPE "RUN" command does - by calling the CREATE or CREATEPROCESS intrinsic. The CREATEPROCESS intrinsic allows the programmer considerable control over the process it creates. In fact, CREATEPROCESS allows more flexibility and more options than the RUN command does. Three of the options of CREATEPROCES are of particular value. These three options (items 3, 8, and 9) allow the programmer to start a program in a mode in which multiple proceses within the process tree are executing concurrently. While other CREATEPROCESS options may be of use later, only

items 3, 8, and 9 are needed for the average application (more about these items later). For those who have used the CREATEPROCESS intrinsic in the past, please note that item 10 is NOT used. Item 10 tells CREATEPROCESS to suspend the calling process and to expect reactivation of the calling process from a son process, father proces or either father or son process. In order to have the father and son processes run concurrently, item 10 is omitted. The side effect of this is that CREATEPROCESS will create the process but will not tell the process to begin executing. If item 10 is provided and has a non-zero value, the father process will be suspeneded and only the son process will executing.

Once CREATEPROCESS has created a son process, it returns a PIN (Process Identification Number). The PIN is like a label and is used in calls to various intrinsics to specify the process upon which the intrinsic is to act.

A process which has been created just sits there until it is allowed to execute. Some applications of CREATEPROCESS automatically start the process (activate it) after creating the son process. When CREATEPROCESS item 10 is omitted (as previously discussed), CREATEPROCESS does not automatically start the process, and the programmer must do so himself.

To tell a process to begin executing, the programmer calls the ACTIVATE intrinsic, passing ACTIVATE the PIN returned by CREATEPROCESS. When both the father and son processes are to execute concurrently, the *susp* (suspend) parameter of ACTIVATE should be zero or omitted. Just as with item 10 of CREATEPROCESS, a non-zero value for *susp* causes the father process to suspend.

The Terminal Interface

When running multiple processes concurrently within a single session, a serious probem becomes apparent - the user cannot determine which process his terminal input will be routed to. This is because all processes within the session normally share the session's one terminal for input and output.

Due to the single-threaded nature of the terminal I/O, only one input or output operation can occur on the terminal at a time. The result is chaos. With multiple processes running concurrently, process "X" may print a prompt on the terminal, but before process "X" reaches the code which accepts the prompt's response, another process (process "Y") may request input from the terminal and any response the user enters for process "X"'s prompt ends up going to process "Y".

As an example, assume that a programmer has written a program which in turn runs both EDITOR and SPOOK5 as concurrently executing son processes. Prompts for both EDITOR and SPOOK5 are displayed on the screen. Since the EDITOR prompt appears first, the user enters an "A" (for ADD) command, thinking that

EDITOR will then prompt for an acccept text lines. Instead, the cursor moves to the next line and the user is perplexed. After sitting and waiting a couple of minutes, the user presses {RETURN} and instead of receiving a line number prompt from EDITOR, receives a prompt from SPOOK5 followed by a line number prompt. Additionally, output from both programs is interleaved. Below is an example (user input is underlined).

```
:run chaos

HP32201A.07.17 EDIT/3000 SUN, SEP 23, 1990,  1:13 PM
(C) HEWLETT-PACKARD CO. 1985
/a
SPOOK5 G.03.05  (C) HEWLETT-PACKARD CO., 1983
> s
    1      this is some test data.
#FILE   #JOB    FNAME    STATE  OWNER
#079    #J33    $STDLIST READY  JOHN.RD
    2      #081    #J34    $STDLIST READY  JOHN.RD
more test data
#080    #J33    T3COMPL  READY  JOHN.RD
#043    #S8     LOGLIST  READY  JOHN.RD
    3      #042    #S8     FANTASIA READY  JOHN.RD
this is line 3.
> //
    4      *ERROR=20 BYTE=0* INVALID COMMAND NAME
//
> ...
1 all
*ERROR=46 BYTE=2* NO TEXT FILE
/> 1 all
_
>     1      this is some test data.
_
>     2      more test data
_
>     3      this is line 3.
_
> /_
_
> e
/_
END OF PROGRAM
/e


END OF PROGRAM
:
```

Utilities and programs interact with the terminal by writing characters to and reading characters from the terminal device. While the terminal is a physical device, a program sees the terminal as one or more files. When a session is created, two files are opened for accessing the terminal - $STDIN and $STDLIST. Input from the terminal is read from $STDIN and output to the terminal is written to

$STDLIST.

Both CREATEPROCESS and RUN allow the diversion of data from its normal terminal-to-program and program-to-terminal paths. The RUN command has ;STDIN= and ;STDLIST= options which allow the user to specify file names to be used to supply program input ($STDIN) and receive program output ($STDLIST). The CREATEPROCESS intrinsic uses options 8 and 9 to specify the ;STDIN= and ;STDLIST= options.

A special type of file is particularly useful when diverting program input and output. The special file type is the "message" file or "IPC" (IPC refering to Inter-Process Communication) file. Message files have the unique characteristic of operating like a queue or pipe. The data written to a mesage file is read from the message file in the same order as it was written (FIFO or First In, First Out). Message files can also buffer program input and output, depending upon how the message files are built. Thus, two or more programs can be linked together with message files providing loose or tight coupling between the programs and data flowing through the message files in real-time.

Some programs can accept input and display meaningful output when their $STDIN and $STDLIST are diverted, and others cannot. Generally, programs which use screen forms (such as V/PLUS applications) cannot easily have their $STDIN and $STDLIST diverted. The reason for this is their block mode input/output as opposed to normal character mode input/output.

In character mode characters are transmitted by the terminal to the HP 3000 as the keys are struck on the keyboard. In block mode the keystrokes cause data to be entered on a form. When the user presses the "ENTER" key, a handshake between the HP 3000 and the terminal begins which ultimately results in entire fields of data being transmitted from the form on the screen to the HP 3000 as one block. Typically, the terminal inserts various special characters between data fields and performs various data editing functions before sending the block of data.

Additionally, most forms management packages evaluate the environment they are running in when initializing themselves. For example, the FRELATE intrinsic is called to see if $STDIN and $STDLIST are related (as they would be with a terminal, but would not be when message files are used). Also, forms management packages evaluate terminal configuration settings, terminal model identification, character echo, and much more. All of this complicates the task of attempting to divert $STDIN and $STDLIST making block mode applications poor choices for package integration.

Programs which use character mode are a different story. The input/output operations performed by character mode programs are less complex than those of block mode programs and it is often rather easy to divert the $STDIN and

$STDLIST of character mode programs.

By using the CREATEPROCESS options 8 and 9, the $STDIN and $STDLIST of programs can be redirected to make use of message files. If those message files are shared with the father process, the father can filter data going to the son process, as well as filter data returned from the son process. Some uses of this capability might be to enhance the functionality of a program, simplify the syntax of its commands, edit or summarize the output of the program, restrict access to certain commands, or perhaps replace the command interface with menu selection.

A Filter Program

A character mode program such as SPOOK5 is a relatively easy program to deal with. SPOOK does not perform fancy checks of the terminal environment and has a single prompt. Output formats are few, and while error messages don't have a lot of detail, they do give the location of the error within the error message, as in "`*ERROR=20 BYTE=0* INVALID COMMAND NAME`".

Consider the following enhancement request. *"Accounting department personnel should NOT be permitted access to SPOOK's Append, Copy, Input, or Output commands. Also, they must be prevented from Texting in spool files with file names other than $STDLIST."* The initial response to this request might be "not possible. SPOOK is an HP utility and we can't modify it." But wait. The accounting department doesn't have direct access to MPE (they have one of those security programs they purchased from a third-party vendor), so they don't have access to the :RUN command. What if the commands going to SPOOK and the output received back from SPOOK were filtered and edited by a program? Couldn't that program impose the requested security?

To satisfy the request the security package will run a "filter" program which accepts commands from the user, checks for restricted commands, passes unrestricted commands on to SPOOK, and displays SPOOK's output on the terminal.

SPOOK permits only one command per line. This simplifies the programming of the "filter" program as commands must begin with the first non-blank character of each input line.

The prompt SPOOK uses is a two character string consisting of a greater-than symbol followed by a blank. Also, prompt string is always written with carriage control 208 (%320) which specifies "no carriage return, no line feed". Therefore, the prompt string can be recognized as an output record containing a character with the decimal value 208 followed by a greater-than character followed by a blank character.

SPOOK reads 72 characters at each command prompt, so the filter program should

also read 72 characters.

So, to perform the function of the first half of the request, the filter program can be fairly simple. To demonstrate the logic required, a form of pseudo-code will be used. Note that "<CR>" indicates a "carriage return" character, "@" as a variable prefix means "use the address of", and "!--" preceeds a comment. The names of MPE intrinsics are in ALL CAPS.

```
Label Start
!--Declarations
Integer Error, ParmNo, ReadLen, CCTL, CommandLen,
   ShowCmdLen, BlankPosition;
DoubleInteger ProcStatus;
LogicalFlag STDLIST'Found;
String To, From, UserCommand, SpookOutput, ShowCommand;

!--Subroutines
Subroutine Display'Thru'Prompt;
   Loop;
      ReadLen:=FREAD (FromSpook, SpookOutput, -132);
      If ConditionCode <> CCE then Return;
      CCTL:=SpookOutput(1,1);
      Print SpookOutput(2,ReadLen) using CCTL;
      If SpookOutput = (208,"> ") then Return;
   EndLoop;
EndSubroutine;

Subroutine Read'Command;
   Loop;
      CommandLen:=FREAD(StdInX, UserCommand, -72);
      Deblank UserCommand;  !--Remove leading blanks
      If UserCommand = {"A","C","I","O"} then
         Print "Restricted Command.";
      Else
         FWRITE (ToSpook, UserCommand, -CommandLen, 0);
      EndIf
   EndLoop;
EndSubroutine;

!--Main Code
COMMAND ["Purge PipeTo"<CR>, Error, ParmNo];
COMMAND ["Purge PipeFrom"<CR>, Error, ParmNo];
COMMAND ["Build PipeTo;Msg;Rec=-72,3,V,ASCII;Disc=1,1,1"<CR>,
   Error, ParmNo];
COMMAND ["Build PipeFrom;Msg;Rec=-132,10,V,ASCII;CCTL;Disc=1,1,1"
   <CR>, Error, ParmNo];
To:="PipeTo"<CR>;
From:="PipeFrom"<CR>;
CREATEPROCESS [Error, PIN, "Spook5.Pub.Sys", Items=(3,8,9,0),
   ItemValues=(1,@To,@From)];
If Error then
   Print "Unable to create SPOOK5 due to CREATEPROCESS error
",Error;
   TERMINATE;
EndIf
```

```
ACTIVATE [PIN];
ToSpook:=FOPEN [To, %30107, %1302];
FromSpook:=FOPEN [From, %30507, %1300];
StdInX:=FOPEN [, %154, %0];
StdList:=FOPEN [, %514, %2];

Loop:
   ProcStatus:=GETPROCINFO [PIN];  !--Make sure SPOOK is still alive
      If ConditionCode <> CCE then TERMINATE;
   Display'Thru'Prompt;
   ProcStatus:=GETPROCINFO [PIN];  !--Make sure SPOOK is still alive
      If ConditionCode <> CCE then TERMINATE;
   Read'Command;
EndLoop;
```

What about the second half of the request (only allow the "texting" of spool files named "$STDLIST")? This part of the request is simply an enhancement of the preceeding example. When the user enters a "T" command, the command is first sent to SPOOK as a "SHOW" command. The output of the "SHOW" command will contain the name of the file. As each line of the "SHOW" command output is received, it is checked for the file name "$STDLIST". Only if "$STDLIST" is found is the "TEXT" command passed through to SPOOK. Below are the changed blocks of pseudo code. Note that the subroutine "Search'For'STDLIST" has been added and the subroutine "Read'Command" has been modified.

```
Subroutine Search'For'STDLIST;
   STDLIST'Found:=False;
   Loop;
      ReadLen:=FREAD (FromSpook, SpookOutput, -132);
      If ConditionCode <> CCE then Return;
      If SpookOutput(18,25) = "$STDLIST" then
         STDLIST'Found:=True;
      EndIf
      If SpookOutput = (208,"> ") then Return;
      CCTL:=SpookOutput(1,1);
      Print SpookOutput(2,ReadLen) using CCTL;
   EndLoop;
EndSubroutine;

Subroutine Read'Command;
   Loop;
      CommandLen:=FREAD(StdInX, UserCommand, -72);
      Deblank UserCommand;  !--Remove leading blanks
      If UserCommand = {"A","C","I","O"} then
         Print "Restricted Command.";
      Else
         If UserCommand = "T" then
            BlankPosition:=Position of " " in UserCommand;
            ShowCommand:=
               "S "+UserCommand(BlankPosition+1,CommandLen);
            ShowCmdLen:=Length(ShowCommand);
            FWRITE [ToSpook, ShowCommand, -ShowCmdLen, 0);
            Search'For'STDLIST;
            If STDLIST'Found then
```

```
                FWRITE (ToSpook, UserCommand, -CommandLen, 0);
             Else
                Print "Restriced Spool File.";
             EndIf;
          Else
             FWRITE (ToSpook, UserCommand, -CommandLen, 0);
          EndIf;
       EndIf;
    EndLoop;
EndSubroutine;
```

The above listings meet the requirements of the program request and can be coded in most programming languages (COBOL, C, PASCAL, SPL, etc.).

Beyond Just Reads And Writes

SPOOK does not perform checks to find out the characteristics of the environment in which it is being run. Many other programs do, however, and as a result they can be more difficult to incorporate into a process handling environment. Below are some questions which help determine the difficulty of the interface to be built and the approach to be taken. If the questions and answers seem to make constructing an interface difficult or impossible, just be patient as there is an approach which handles most if not all of the problems which come up.

- **What does the program's prompt look like?**

In order to determine when to request input from the terminal user, some data from the son program must be used as the trigger which tells the father process "the son program wants input". In the earlier SPOOK example the trigger was the prompt string. Fortunately, there was only one prompt string to search for. Unfortunately, most other programs use multiple prompts.

- **Is the same prompt used throughout the program?**

EDITOR is an example of a program with multiple (and changing) prompts. The normal EDITOR prompt is "/". However, the ADD command prompts with a ten character field containing blanks, a line number, and more blanks. The important thing is to know the types of prompts used and if they are not constant, what pattern they follow.

- **Are all terminal inputs preceeded by a prompt?**

Note that EDITOR's MODIFY command prompts by starting a null length line. The preceeding print line (which is really the prompt) is not distinguishable from any other print line. Thus, what can be done about the MODIFY command?

- **How many characters of input does the program expect?**

Without the program source code it may seem difficult to determine the number of characters the program expects. One easy way is to simply sit down with the program and at its prompt enter as many characters as possible, counting them up

as they are entered. When the number of characters the program expects has been reached, the program ends the input operation and the program continues.

- **Does the number of characters to be read from the terminal** change depending upon previous commands or the type of input being requested?

Once again, entering characters until the program says "no more" may be the easiest way to find out.

- **Is the terminal read supposed to be timed?**

Timed terminal reads are often used when passwords are to be entered or when a database transaction require input while the database is locked. In the case of the database transaction, the timing of the terminal read prevents the user from getting up and walking away from the terminal and causing all users to be locked out of the database for an extended period of time. Some programs display a "timeout" message if there is no response in the designated time period. In such cases, manually keep track of the elapsed time between the prompt and the "timeout" message.

- **Were FCONTROL, FSETMODE, or FDEVICECONTROL options set to change the way terminal input/output takes place (echo off for database passwords, for example)?**

FCONTROL and FDEVICECONTROL each have dozens of options which the application designer can use. Many of these options have significant impact on how the application interacts with the terminal and the user. If the effects of FCONTROL, FSETMODE, and FDEVICECONTROL are ignored, the application may fail to operate. Unfortunately, these intrinsics do not send any information to $STDLIST so there is no way of capturing information about FCONTROL, FSETMODE, or FDEVICECONTROL calls in redirected $STDLISTs.

- **Is FRELATE called to determine if the program's STDIN and STDLIST form an interactive, duplicative pair?**

Programs such as EDITOR use FRELATE to determine if they are being run interactively or in batch. When run from a terminal, FRELATE returns to the program status information which is interpreted as "this program is being run interactively." When run from a job or with $STDIN and/or $STDLIST redirected, the status information returned is different and is interpreted by the program as "this program is being run in batch." This is important as many programs terminate after the first error when run in batch, but continue to accept and process commands when run interactively.

- **Does the program use a CONTROL-Y trap?**

EDITOR and many other programs use the subsystem break (CONTROL-Y) as a way of allowing the user to suspend or terminate command processing. A typical application of CONTROL-Y is to terminate a listing, as when the user enters

"L 1/10000" instead of "L 1/1000". Without the CONTROL-Y trap the user would have to sit and wait while unwanted data is displayed on the terminal screen. With CONTROL-Y the user can stop the display at any time. One side effect of having multiple processes executing concurrently is that only one CONTROL-Y can be active at a time. If two programs which use CONTROL-Y are executing within the same session, only one program (the program which most recently set the CONTROL-Y trap) will have the CONTROL-Y feature. The other program will ignore any presses of CONTROL-Y. What can be done with the other programs which require CONTROL-Y?

Intrinsic Call Interception (hooking)

While it may seem that the above problems are insurmountable, there is a way of obtaining and passing the required information from program to program. The solution involves intercepting the calls a program makes to selected MPE intrinsics. Just as the intrinsics interpret what to do by examining the parameter values they were called with, procedures which intercept the calls can determin what functions the intrinsics called are to perform. The methodology of intercepting calls to intrinsics is not new - it has been around since the late 70's or early 80's. To find out more about intercepting intrinsic calls, please look for articles and papers on "hooking programs." Papers on hooking programs can be found in most recent conference "Proceedings."

Intercepting intrinsic calls is useful as it provides a data collection capability. Once collected, the problem becomes one of how to transmit the data to other processes. Standard program output is sent to $STDLIST. Since the son process has its $STDIN and $STDLIST diverted to message files, with another process (the father process) at the other end of the message files, perhaps the son process's $STDIN and $STDLIST may be used for communicating the information collected from the intrinsic calls.

By using a message file with carriage control (CCTL), not only is it possible to determine the carriage control necessary for the proper display of program output, but it is also possible to transmit intrinsic parameters. The method used involves the use of unused carriage control values. When passing carriage control information through a message file, there are 256 possible carriage control codes (a table of valid carriage control can be found in the "MPE Intrinsics Reference Manual" discussion of the FWRITE intrinsic). With the exception of carriage control values zero and one, most of the low-value codes are not used. Since relatively few intrinsics need be intercepted, each intrinsic can be assigned to an unused carriage control value. The code which intercepts the call to a particular intrinsic communicates the intercepted data by writing a record to the program's $STDLIST specifying the values of the intercepted parameters as the data portion of the record, and a carriage control value specifying the intrinsic call intercepted.

The father process then examines each record it reads from the message file to determine if the record read contains one of the special carriage control values that are being used to transmit intrinsic information. If an "intrinsic data" record is found, the father process can then take whatever action is specified (turn echo off, read 80 characters from the terminal, etc.).

Some of the data obtained from the intrinsic calls may be needed when calls are made to other intrinsics. As an example, the FREAD intrinsic can be used to read from a disc file, a tape file, the terminal, etc. To determine whether the file being read from is supposed to be the terminal (in which case the father process must supply the data) or from some other source (in which the file can be read directly), the parameters passed to FOPEN must be evaluated (FOPTIONS, FILENAME). Two methods are easy to implement - store FOPEN information in the global area of the process's stack or pass the FOPEN information to the father process for safe keeping. The first method requires restructuring the program file but is fast. The second method involves sending data between the two processes for every FOPEN or FREAD and is slow. Again, please see previous conference PROCEEDINGS for any of a number of articles on hooking and making structural changes to program files. In almost all cases it is preferable to save information obtained from FOPEN calls in the global area of the program's stack.

Now that there appears to be a method of passing intrinsic call information between the processes, some of the previously noted interfacing problems have solutions. Below is a second look at the interfacing problems.

- **What does the program's prompt look like?**
- **Is the same prompt used throughout the program?**
- **Are all terminal inputs preceeded by a prompt?**
- **How many characters of input does the program expect?**
- **Does the number of characters to be read from the terminal change depending upon previous commands or the type of input being requested?**

The READ, READX, and FREAD intrinsics can accept input from the terminal. With calls to those intrinsics intercepted and information about the calls (number of characters to read) passed back to the father process through special "print" records containing unusual carriage control values, it no-longer becomes necessary to know what the prompt lines look like, or how many types of prompts there are. If multiple processes are being managed by the father process (SPOOK, EDITOR, and LISTDIR5 for example), the father process can simply save the last print record received from each of the processes as the prompt strings. Thus, if the user is presently interacting with EDITOR and is in ADD mode and decides to issue a SPOOK command, the father process can simply redisplay the last print line received from EDITOR (the line number prompt) to reprompt the user for the next EDITOR line.

- **Is the terminal read supposed to be timed?**

Timed reads are enabled through the FCONTROL intrinsic. If calls to FCONTROL are intercepted and those pertaining to the terminal are passed on to the father process, then the father process can use the passed data to set up the timed read.

- **Were FCONTROL, FSETMODE, or FDEVICECONTROL options set to change the way terminal input/output takes place (echo off for database passwords, for example)?**

Just as in the case of the timed reads, calls to FCONTROL, FSETMODE, and FDEVICECONTROL can be handled by the father process.

- **Is FRELATE called to determine if the program's STDIN and STDLIST form an interactive, duplicative pair?**

Since the object is to make the application think it is running interactively, the calls to FRELATE can be simple diverted with the diverting procedure returning a value which indicates interactive access.

- **Does the program use a CONTROL-Y trap?**

While only one process can make use of CONTROL-Y at a time, CONTROL-Y can effectively be shared if it is re-enabled by the son process after each terminal input completes. For example, if both EDITOR and SPOOK are being run from a common father process, there would normally be a conflict in the use of the CONTROL-Y. However, if one of the processes is waiting on input and the other is running, a solution to the CONTROL-Y conflict can be found by having the procedures which intercept the calls to READ, READX, and FREAD re-enable CONTROL-Y before exiting.

To illustrate what an intrinsic interception procedures might look like, two are printed below as samples.

```
!  Copyright 1987 by Innovative Software Solutions, Inc.
!  Permission to use provided credit is given is hereby granted.
!  All other rights reserved.
!
!  Intrinsic      Replacement      CCTL  Data Values     Global
!  Name           Procedure Name   Code  To Father       Accessed?
!  ----------     --------------   ----  -------------   ---------
!  READ >-------> ISSO ..... 2    Len                   Read
!  READX >------> ISSRO ..... 3    Len                   Read
!  FREAD >------> ISSR1 ..... 4    Len                   Read
!  FOPEN >------> ISSOP ..........................       Read/Write
!  FCONTROL >---> ISSFCTLO ..... 5  CtlCode,Param        Read
!  FSETMODE >---> ISSFSMDO ..... 6  ModeFlags            Read
!  XCONTRAP >---> ISSCTRLY ..... ..................      Read/Write
!  FRELATE >----> ISSFRELT ..... ..................      Read
!
!  Intrinsic Interception Procedure ISSR1
!  (Replacement for the FRead Intrinsic)
```

```
Integer Procedure ISSR1 (FileNum, Target, TCount);
   Value              FileNum, TCount;
   Integer            FileNum, TCount;
   Logical Array      Target;
Begin
   Logical            Status'Rtn=Q-1;
   Byte Pointer       Global'Data;
   Logical            FOpts, SetCtrlY;
   Integer            CtrlYPLabel, OldPLabel;
   Integer            LenRead, Len;
   Logical Array      LBuf(0:19);
   Byte Array         BBuf(*)=LBuf;

   Intrinsic          ASCII  FRead, Print, XConTrap;

   SetCtrlY:=FALSE;

   << Find the FOptions saved by the FOPEN intercept >>
   << procedure. >>
   @Global'Data:=0;
   While Global'Data <> "KJnavEnosillAylimEKC" do
      @Global'Data:=@Global'Data(256);
   @Global'Data:=@Global'Data(22);
   FOpts:=Logical(Global'Data(FileNum*2))*256+
      Logical(Global'Data(FileNum*2+1));

   If FOpts.(10:3) = 4 or
      FOpts.(10:3) = 5 then begin
      << A $STDIN or $STDINX file >>
      << Tell the father process to perform a read of TCount >>
      LBuf:=Logical(TCount);
      << CCTL 4 tells father source is FREAD >>
      Print (LBuf, -2, 4);
      SetCtrlY:=TRUE;
   End;

   << Read the data sent from the father process >>
   LenRead:=FRead (FileNum, Target, TCount);
   Push (Status);
   Status'Rtn.(6:2):=TOS.(6:2);
   ISSR1:=LenRead;

   If SetCtrlY then begin
      << Reset the CONTROL-Y trap (if any) >>
      CtrlYPLabel:=Integer(Logical(Global'Data)*256+
         Logical(Global'Data(1)));
      If CtrlYPLabel <> 0 then begin
         XConTrap (CtrlYPLabel, OldPLabel);
      End;
   End;
End;  << ISSR1 >>

!  Intrinsic Interception Procedure ISSCTRLY
!  (Replacement for the XConTrap Intrinsic)

   Procedure ISSCTRLY (PLabel, OldPLabel);
      Value              PLabel;
```

```
      Integer           PLabel, OldPLabel;
   Begin
      Logical           Status'Rtn=Q-1;
      Byte Pointer      Global'Data;
      Logical           FOpts;
      Logical Array     LBuf(0:19);
      Byte Array        BBuf(*)=LBuf;
      Integer           Len;

      Intrinsic         ASCII, Print, XConTrap;

      << Find the File Information obtained from FOPEN >>
      @Global'Data:=0;
      While Global'Data <> "KJnavEnosillAylimEKC" do
         @Global'Data:=@Global'Data(256);
      @Global'Data:=@Global'Data(22);

      << Store the CONTROL-Y PLABEL in the GLOBAL part of the stack
>>
      Global'Data:=PLabel.(0:8);
      Global'Data(1):=PLabel.(8:8);
      XConTrap (PLabel, OldPLabel);
      Push (Status);
      Status'Rtn.(6:2):=TOS.(6:2);
   End;  << ISSCTRLY >>
```

Handling Multiple Son Processes

Once a single son process has been handled successfully, handling multiple son processes is not terribly difficult. The key to handling multiple processes is to decide how and when each process is to execute and what information about the terminal environment must be saved and restored as the father process switches between son processes.

As an example, assume that a new programming environment is desircd (a programming environment was chosen as it is common to most shops which other applications might not be). At present the programmers waste a lot of time texting and re-texting their source files and entering and exiting from various utilities. The goal is to make the programmers more productive by allowing them to switch back and forth between utilities lightning fast and without having to re-text their source files or spool files. The programmers spend most of their day working with SUPERED (a third-party program editor), SPOOK5, MPE commands, the COBOL compiler, SEGMENTER, and QUERY. The solution is to keep as many of the utilities ready to run as possible. All the utilities are individual programs and can be process handled with the exception of the MPE commands. Assume for a moment that someone has a program which allows most MPE commands (including PREP, RUN, SEGMENTER) which may be incorporated into the package. Thus, the application become one of process handling six different programs, keeping the programs suspended but ready to run when fed some input.

SUPERED, SPOOK5, the MPE command program, SEGDVR (the SEGMENTER program), and QUERY can all be run concurrently. The COBOL compiler can be

invoked when needed. To switch from one utility to another the user enters the name of the utility preceeded by a "$", as in "$SUPERED" or "$SPOOK". SUPERED and QUERY turn character echoing off and change various input options through calls to the FCONTROL and FSETMODE intrinsics. Some of these settings may not be fully compatible with the operation of the other utilities or may confuse the user when used out of context, so it is wise to track which FCONTROL and FSETMODE options have been set within which programs. This can easily be done by keeping a set of variables relating to terminal echo, FSETMODE line feed mode, etc. for each of the programs being process handled. Also, variables will be needed to hold the most recent prompt string (last print record received prior to a request for terminal input), the number of characters requested from the terminal, and any time limit to be placed upon the terminal input. If desired, the father process could even remind the programmer of the utility being switched to when a process switch occurs. Below is an example of how what the terminal screen of such a programming environment might look (user input is underlined).

```
SUPEREDIT>m 1
     1    $Control USLInit, List, Source
  Changes:a, Verbs
     1    $Control USLInit, List, Source, Verbs
  Changes:_
SUPEREDIT>l 7
     7    *    Prep with MAXDATA=8192 or greater *
SUPEREDIT>k
Saved.
SUPEREDIT>$mpe
:COBOL MySource,MyUSL,*MyList
... Cobol compiler messages ...
:$spook
> s
#FILE    #JOB     FNAME     STATE  OWNER
#043     #S8      LOGLIST   READY  JOHN.RD
#045     #S8      MYLIST    READY  JOHN.RD
> t 45
> f "***"
> $segmenter
-usl myusl
-prepare myrun;maxdata=8192;fpmap
 $edit
SUPEREDIT>l *
     7    *    Prep with MAXDATA=8192 or greater *
SUPEREDIT>
```

Again, the advantage of such a system is that all the programs needed by the programmer are instantly available and the programmer can switch between utilities without having to wait through the re-initialization of the utilities or the re-texting of files. When the utility is re-entered, the utility resumes execution right were it left off - in the case of an editor, at the same prompt or line at which it was left.

Remember, handling multiple utilities at once can be difficult and may seem

impossible at first. To simplify the task and make it much easier to tackle, it is best to begin with attempting to handle only one utility. The lessons learned from the experience gained save much time and effort when designing the interfaces for other utilities.

Summary

Process handling allows the execution of one or more programs from within another program and can be used in many ways.

A program with an awkward interface (even if the source code is not available) can receive a face lift through process handling. Prompts or series of prompts can be replaced with new prompts or even menus.

Time and resources can be saved by process handling multiple concurrent utilities or programs. Switching between programs becomes lightning-fast. Also, the CPU time spend entering and exiting from programs and re-texting files or re-opening databases is saved.

New applications can be created by combining readily available utilities. As an example, QUERY, SORT, and EDITOR can be process handled such that the user never sees the underlying QUERY, SORT, and EDITOR - only the merged result of their execution (imagine creating your own user-specific 4GL tools without the 4GL price tag).

As familiarity with process handling and input/output redirection builds, the posibilities of new applications for process handling grow. Try process handling. It isn't as hard as it may seem, and is very powerful.

Technical Note - Calling CREATEPROCESS/ACTIVATE

Call CREATEPROCESS specifying items 3, 8, and 9. Bits 10 and 11 of the word passed for item 3 should be set according to the SL file to be used when the program is loaded by MPE.

Item 3: (15:1):=1 activate father upon termination
(10:2):=x x=0 for LIB=S
x=1 for LIB=P
x=2 for LIB=G

Item 8: a pointer to (the address of) a string containing the name of the file to be used instead of $STDIN.

Item 9: a pointer to (the address of) a string containing the name

of the file to be used instead of $STDLIST.

Remember to call ACTIVATE with the "susp" parameter zero or omitted.

TITLE: Automate Testing To Improve Software Quality

AUTHOR: David R. Mendoza

President

Software Development Resources
845 Berkeley Way

Vista, CA 92084

(619) 726-9753

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING.

PAPER NO. 3227

TITLE: Information Management in the 1990's

AUTHOR: Peter Ney

DCE Information Management Consultancy

Prinsengracht 747 - 751

1017 JX Amsterdam

NETHERLANDS Phone-+31 20-264400

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING.

PAPER NO. 3228

Paper # 3229

Client Server System Design

Steve Palmer
Steve Palmer & Associates
3028 Driscoll Drive
San Diego, CA 92117
619/274-3601

What is Client/Server System Design?

Client/Server is a system design methodology that integrates multi vendor hardware and software solutions. The multi vendor environments are transparent to the user. Suddenly you have a PC that can store 32 GBS of data! Data management and transaction functions are managed independently from user interface and application functions.

A typical Client/Server system design places an HP3000 as the server. This HOST machine has the storage and power to process transactions about data. It can store data in IMAGE data bases, KSAM file structures, MPE files of many kinds i.e., sequential, RIO, message, circular files. It can transfer data to and from disk storage at speeds up to 19,200 bits per second.

The typical Server is a PC. Either Macintosh or IBM platforms work well. They are inexpensive and they have great looking Graphical User Interfaces. A reasonable amount of local data can be maintained for data validation and quick lookup.

Some Examples

This year, we wrote a Sales Contact and Order Entry system that allows sales people in the field to use their laptop PC's on the road. They had access to an abbreviated Customer Master List, a detail Customer Master for customers they will meet with this week, and a Price List (good for the next 30 days).

Their laptops have programs that allow them to lookup data in each file, to print an order, to maintain information about their contacts, and common everyday PC programs like LOTUS/1-2-3 and MicroSoft Word.

At the end of each day, they call into the central HP3000 and synchronize their data files. Only the changed information is transferred.

This year, we wrote a Document Retrieval System. The customer was comfortable with the Graphical User Interface of the PC environment and wanted to use that look and feel for Document Retrieval. Some of their documents are LOTUS/1-2-3 worksheets and WordPerfect 5.0 documents. All the maintenance happens at the PC level; but we centralize data storage, which allows multiple users to access the information simultaneously. Client/Server was the perfect solution.

Also this year, we wrote an Order Entry System for a customer that wanted to off load some work from their HP3000 computer system. The system was too busy handling VIEW/3000 screens. By allowing a PC to process the screen data (screen painting, data validation, and data normalization) we relieved the HP3000 of about half its work. By the way, this user eliminated the need to upgrade to a faster HP3000 and saved $100,000.

Why design for Client/Server?

According to the Business Research Group of Cahner's Publishing Company in Newton, Mass., greater business productivity, not lower cost, is the main reason companies are quickly adopting Client/Server technology.

By taking advantage of a modern Graphical User Interface that has widgets like radio buttons, slider bars, mouse events, color, and sound we make a user more effective. The training issue is reduced, too. Imagine training your users in this familiar PC environment once. You can change the Server to anything you want, and the user part of an application remains the same.

You can attach and detach whenever it is convenient. Your system design must incorporate routines to synchronize the data pools on the Client and Server. We will talk about this later.

Now you can access HP/3000 data with programs like LOTUS/1-2-3, WordPerfect and dBASE. This isn't anything new; we have had this capability for a few years in copying down a file and importing it into PC based programs. But, imagine the ability of viewing HP/3000 data interactively and dynamically. Would that be of interest to you?

Client/Server System Components

The Server hardware component of a Client/Server approach could be an HP3000 computer system (or anything else that supports an interactive link with the Clients). The main task of this system is to maintain data files and to read and write data in files. Therefore, fast disk access time is advantageous. A high speed CPU is less important since most of an application's thinking happens in the Client hardware. You will probably need 1 terminal port for each Client. Clients may be hardwired or connected via modem.

The Client hardware could be a personal computer such as an IBM PC or Macintosh; it could also be a Unix workstation. It is important that this system have graphics capability with a minimum amount of disk (either diskette or hard disk) to store the operating system, the application screen programs and some validation tables in files, if they are needed by the application. It is also important that this system possess communications capability compatable with the Server such as asynchronous on the HP3000. It should have ample main memory to store it's operating system, application program plus the process-to-process linkage software needed at run time.

Server software is a program that can monitor one or more terminal ports. It's main task is to send and receive messages (or transactions) that are esentially requests for data service (read, write, update, delete, find, etc.). This programmable Server is a critical feature that allows data management and transaction functions to be managed separately from user interface and application functions. Stored procedures and triggers allows for unscheduled transactions to be properly processed.

Client software is a user application program. We have found that the best use of Client software is in graphically representing data on a screen and allowing a user to use special devices like a mouse, bar code readers, and hand readers. These programs must send and receive messages that

result in data being read, written, updated, deleted or found on the Server.

Process-to-process linkage software is required to synchronize the Server and Client. It provides the data communications link for interprocess communications. And, it should invisibly handle the transport of data between Client and Server.

In order for Client/Server to function, a session must be active on the Server. This is establish by a conventional :HELLO command. If you are running a UNIX based Client, you must also be logged onto that system. Personal computers generally do not require a log on unless you are running on a network.

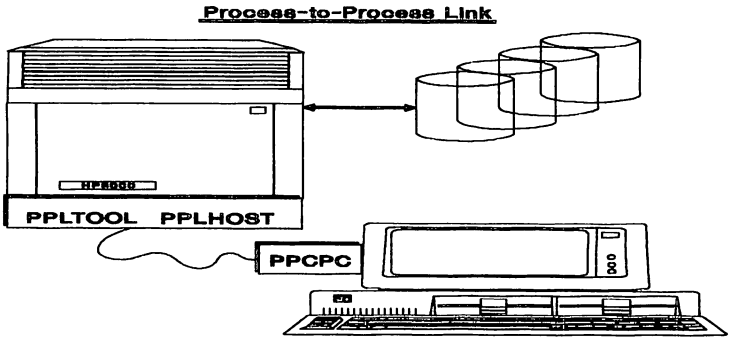
What do we use for software tools?

On the Server, we use PPL (Process-to-Process Link from Walker, Richer & Quinn). This product contains the linkage software described above. It takes the form of 2-programs that reside on the HP3000 (or DEC) Server, PPLTOOL and PPLHOST and 2-programs that reside on the Client (PPLPC.EXE and R1LINK.EXE).

PPLHOST is the communication's server for PPL and PPLTOOL is the application's server. PPLPC.EXE is the application server on the Client system. This is a small (about 4K) terminate and stay resident program that identifies where the communications driver, PPLCOM.SYS, is within the Client's main memory. It must be running whenever a PPL application is running. R1LINK.EXE is a limited version of Reflections I that allows a user to log onto the Server system. Advanced capabilities such as Block Mode screen handling is not present in this program. It is assumed that screens will be managed by the Client.

On the Client, our application development uses the "C" programming language from MicroSoft, a set of "C" language callable routines that manage screens and windows. The average "C" program that contains a few windows and data fields that manipulate 3 or 4 data bases is about 1000 lines of code and is extremely fast and efficient.

We have also been successful in interfacing MicroSoft COBOL (Version 4.0) with PPL. This is not a capability that is built into the WRQ product. We wrote an interface between COBOL and the PC PPL Library of callable routines.



How does PPL work?

There are 2 ways you can implement systems with PPL from Walker, Richer & Quinn. You can use PPLTOOL on the Server to service calls from the Client, which are essentially identical with the calls you would make to the MPE intrinsic library. This is convenient since you don't need to learn anything new and you do not write any application code for the HP3000 environment. The disadvantages include bigger and more complex Client programs, slower execution when handling large amounts of data. These programs are not portable to a DEC VAX environment.

A more efficient approach, but also more difficult, is to make direct PPL calls. Here you write an application program for both the Client and Server. On the HP3000, your program interfaces with PPL via an IPC file. Your program can do preprocessing of the data on the HP3000, then pass down only the results to the Client. The Client portion is portable to a VAX environment since design is not dependent on knowing host database structure.

Here is how the second approach works:

1. The user logs onto the host via Reflection.

2. When PPLPC.EXE is loaded on the Client it writes the address of the application entry point to the device driver PPLCOM.SYS. PPLPC.EXE then terminates and stays resident.

3. When the Client application runs, it opens the device driver PPLCOM.SYS and reads the address of the

entry point for PPLPC.EXE. To communicate with the Server, the Client application calls this address.

4. The host application makes a Version check to verify that PPLPC.EXE is functioning and to get the version number.

5. The Client application makes an Initialized connection call to pass the communication's parameters for the link to PPLPC.EXE. The parameters are passed through a Refleciton configuration file. This starts PPLHOST on the Server.

6. The Client application makes an Open circuit call, passing the name of the host application program to PPLPC.EXE which then sends a message to PPLHOST to activate the host application as a son process. Two files are required: (1) messages from PPLHOST to the application and (2) messages from the application to PPLHOST.

7. Data is exchanged using the PPL error-detecting protocol.

8. When the Client application is ready to send a message to the host application, it calls PPLPC.EXE with a Send a message command (either synchronous or asynchronous is supported).

9. PPLHOST sends an acknowledgement to PPLPC when it receives a message and writes the message to the IPC file called SRV2USER.

10. When the host application is ready to send a message to the Client application, it writes a send a message record to the IPC file called USER2SRV. PPLHOST reads the message and sends it to PPLPC.EXE which then stores the messagein its internal buffer, and sends an acknowledgement to PPLHOST.

11. When PPLHOST receives the ACK from PPLPC, it writes an Acknowledgement record to the IPC file SRV2USER. At this point, PPLHOST and PPLPC.EXE resume the exchange of control packets while they are not doing application work.

Appendix A is a "C" program that illustrates how you might use "PPLTOOL" to call MPE intrinsics or IMAGE data base intrinsics. This is only one of many subroutines in the Document Processing System.

Appendix B is a sample PPL program written in the "C" programming language. It illustrates how direct PPL calls can be coded.

Attach/Detach Feature

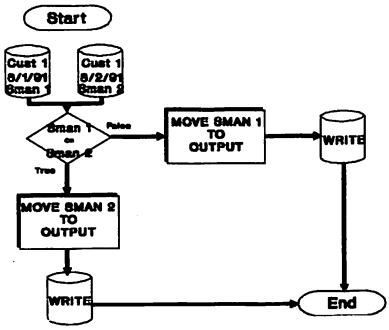
Good Client/Server system design might include an Attach/Detach feature. The classic need is illustrated in our Sales System. Since our sales people are in the field most of the time, it is not possible for them to be connected to the Server all the time. We decided to give them enough information locally stored in the Client to run independently from the Server.

This created one gigantic problem- how to keep the data synchronized. Our initial approach was to copy the data bases down to the Clients at night. This worked fine since there was enough time at night to download.

A better approach is to maintain a change flag in the Server data bases. This extra field in each record is used to indicate a N New, M Modified, D Deleted record. Records are retained for 1 week with a change date. A last download date and time is kept in each Client. When a data download is performed, only those changes that occurred since the last download date and time are transferred from the Server to the Client. After the transfer, the Client download date is updated to current date and time. This was a great improvement over the total transfer method.

When the Client updates a local record, a change date and time is recorded in that record. When transferring data from the Client to the Server, all data that has changed from the last upload date and time is sent to the Server. A batch process on the Server is run once per day. It's job it to resolve conflicts between many Clients that are making changes (even to identical records) in the data base. We established the rule that when 2 Clients want to change the same record on the Server, the Client with the most recent date and time gets recorded and the others get printed on an exception report.

Here is the logic of this rule:



Because we use ISAM or KSAM file structures for a good part of our work, the file structures and retrieval methods are identical on both the Server and Client. Therefore, it is easy to write application Client programs that process data either from the Server or locally in the Client. The application logic doesn't change much when running in a detached mode.

Conclusions about Client/Server

Client/Server is a useful design methodology. It can improve your user's impression of an application by using Graphical User Interfaces. It is easier to train users on a system where they are familiar. It eliminates the need to change your application in the future when you need to move to another Server platform. And it can save you the necessity of upgrading your HP3000 computer system by off loading some application's activities to the Client.

Client/Server is not easy to implement. It requires a good knowledge of both the Server and Client environments. Programming capability is essential for both machines. On the HP3000, you must know which intrinsics to use, and when. On the PC, you must be proffecient in the "C" programming languages and Graphical User Interfaces.

We think Client/Server is a good strategy. Our customers like the speed, look and feel of their Client/Server systems. They like thinking that the server can change and the user interface stays the same. And, they like saving money.

Appendix A

```
/*
 *
 *                      File Handling
 *                      Document Processing System
 *                      Copyright (c) 1991
 *                      Steve Palmer & Associates
 *                      3028 Driscoll Drive
 *                      San Diego, CA 92117
 *
 *
 *                  A L L   R I G H T S   R E S E R V E D
 *
 *                           F I L E S . C
 */

#include <vcstdio.h>
#include "sbp.h"

     /*  File is INDEX   */
COUNT  index_found;
COUNT  index_file_open;
TEXT   index_FILE[17];   /*  file name                    */
TEXT   index_POS_BLK[128];
TEXT   index_key_buf[55];  /* <--- key size here         */
  struct {
      char deleteflag        [   2]; /* rqd by HP3000    */
      char fldid             [  55]; /* key #1           */
      char subfile           [  20]; /* name of org file */
      char zipfilename       [  55]; /* key #2           */
      char mstrdoc           [   2]; /* true/false flag  */
      char available         [   2]; /* checked outodify?*/
     }  index_buf;

/*   index FILE HANDLING FOLLOWS   */

COUNT index_open()
{
    empty(btmsg,  30);
    empty(index_FILE,  16);
    strncpy(index_FILE, "index ", 8);

    empty(index_buf.deleteflag,    2);
    empty(index_buf.fldid,        55);
    empty(index_buf.zipfilename,  55);
    empty(index_buf.subfile,      20);
    empty(index_buf.mstrdoc,       2);
    empty(index_buf.available,     2);
    empty(index_key_buf,          55);


OPENAGAIN:
```

```
     BUF_LEN = sizeof(index_buf);
     BTSTAT= BTRV(B_OPEN, index_POS_BLK, &index_buf,
          &BUF_LEN,index_FILE, 1);
      if (BTSTAT == 12)
           {  index_create();
              goto OPENAGAIN;
           }
      if (BTSTAT != 0)
         (sprintf(buf,"Problem opening %s status = %d",
                                 index_FILE, BTSTAT);
          do_msg();
           }
        else
        {   index_file_open = ISTRUE;
            empty(index_buf.deleteflag,    2);
            empty(index_buf.fldid,        55);
            empty(index_buf.zipfilename,  55);
            empty(index_buf.subfile,      20);
            empty(index_buf.mstrdoc,       2);
            empty(index_buf.available,     2);
            empty(index_key_buf,          55);
         }

      if(useppl)
         {xindex = FOPEN(index_FILE, 7, 229);
          if (cond_code != CCE)
            {sprintf(buf,"Problem opening %s status = %d",
                                 index_FILE, cond_code);
             do_msg();
              }
            }

   return(0);
}     /*   end of procedure   */

COUNT index_get_equal()
{  empty(index_key_buf, 55);
   strncpy(index_key_buf, index_buf.fldid, 55);


   BUF_LEN = sizeof(index_buf);
      BTSTAT=BTRV(B_GET_EQUAL, index_POS_BLK, &index_buf,
                &BUF_LEN,index_key_buf, 0);
   if (BTSTAT != 0)
      { sprintf(buf,"index get equal- %d",BTSTAT);
        index_found = ISFALSE;
      }
     else
      {
       index_found = ISTRUE;
      }
```

```
   if(useppl)
      {BUF_LEN = 0 - BUF_LEN;
       FREADBYKEY(xindex, (char*)&index_buf, BUF_LEN,
                                 index_key_buf, 0);
       if (cond_code != CCE)
          {sprintf(buf,"Problem with Index - get equal\n");
           do_msg();
           index_found = ISFALSE;
             }
         }

  return(0);
}     /*   end of find_equal    */

COUNT index_get_alt_equal()
{  empty(index_key_buf, 50);
   strncpy(index_key_buf, index_buf.zipfilename, 49);

   BUF_LEN = sizeof(index_buf);
   BTSTAT=BTRV(B_GET_EQUAL, index_POS_BLK, &index_buf,
         &BUF_LEN,index_key_buf, 1);
   if (BTSTAT != 0)
      { sprintf(buf,"index get equal- %d",BTSTAT);
      /*  do_msg();  */
        index_found = ISFALSE;
      }
     else
      {
       index_found = ISTRUE;
      }

   if(useppl)
      {BUF_LEN = 0 - BUF_LEN;
       FFINDBYKEY(xindex, index_key_buf,
                    78,trlen(index_key_buf),0);

       if (cond_code != CCE)
          {sprintf(buf,"Problem getting Alt Index");
           do_msg();
           index_found = ISFALSE;
             }
        else
          {index_found = ISTRUE;
            }
         }

  return(0);
}     /*   end of find_equal    */

COUNT index_get_first()
{  empty(index_key_buf, 55);
   strncpy(index_buf.zipfilename, ptr, strlen(ptr));
   BUF_LEN = sizeof(index_buf);
```

```
   BTSTAT=BTRV(B_GET_FIRST, index_POS_BLK, &index_buf,
               &BUF_LEN,index_key_buf, 1);
   if (BTSTAT != 0)
      {
        index_found = ISFALSE;
      }
     else
       {
        index_found = ISTRUE;
        }

    if(useppl)
      {BUF_LEN = 0 - BUF_LEN;
       FFINDBYKEY(xindex, index_key_buf, 78, 55, 0);
       if (cond_code != CCE)
          {index_found = ISFALSE;
           sprintf(buf,"Index not found- FFINDBYKEY");
                 }
         }

  return(0);
}     /*   end of get first   */

COUNT index_get_next()
{  BUF_LEN = sizeof(index_buf);
   BTSTAT=BTRV(B_GET_NEXT, index_POS_BLK, &index_buf,
          &BUF_LEN,index_key_buf, 1);
   if (BTSTAT != 0)
      { index_found = ISFALSE;
      }
     else
      {
       index_found = ISTRUE;
       }

    if(useppl)
      {BUF_LEN = 0 - BUF_LEN;
       FREADC(xindex, index_buf.deleteflag, BUF_LEN);
       if (cond_code != CCE)
          {sprintf(buf,"Problem with Index File- get next");
           do_msg();
           index_found = ISFALSE;
             }
         }

  return(0);
}     /*   end of get next    */

COUNT index_get_prev()
{  BUF_LEN = sizeof(index_buf);
   BTSTAT=BTRV(B_GET_PREV, index_POS_BLK, &index_buf,
          &BUF_LEN,index_key_buf, 0);
   if (BTSTAT != 0)
      { sprintf(buf,"Beginning of Selections");
```

```
        do_msg();
        index_found = ISFALSE;
       }
     else
       {
        index_found = ISTRUE;
       }
    if(useppl)
      {BUF_LEN = 0 - BUF_LEN;
       FPOINT(xindex, -1);
       if (cond_code != CCE)
          {sprintf(buf,"index File- get prior");
           do_msg();
           index_found = ISFALSE;
             }
         }
  return(0);
}     /*   end of get prior    */


COUNT index_upd()
{ BUF_LEN = sizeof(index_buf);
  BTSTAT=BTRV(B_UPDATE, index_POS_BLK, &index_buf, &BUF_LEN,
                                          index_key_buf, 0);
  if (BTSTAT != 0)
     { sprintf(buf,"Update Problem- index File %d",BTSTAT);
       index_found = ISFALSE;
      }
     else
       {
        index_found = ISTRUE;
        }

    if(useppl)
      {BUF_LEN = 0 - BUF_LEN;
       FUPDATE(xindex, index_buf.deleteflag, BUF_LEN);
       if (cond_code != CCE)
          {sprintf(buf,"HP3000 Index File- Update");
           do_msg();
           index_found = ISFALSE;
             }
         }

}    /*    end of update    */

COUNT index_ins()
{  empty(index_key_buf, 55);
   BUF_LEN = sizeof(index_buf);
   strncpy(index_key_buf, index_buf.fldid,  55);

   BTSTAT=BTRV(B_INSERT, index_POS_BLK, &index_buf,
          &BUF_LEN,index_key_buf, 0);
   if (BTSTAT != 0)
     { sprintf(buf,"Insert Problem- index File %d",BTSTAT);
```

```
        do_msg();
        index_found = ISFALSE;
       }
      else
        {
         index_found = ISTRUE;
         }

    if(useppl)
      {BUF_LEN = 0 - BUF_LEN;
       FLOCK(xindex, 1);
       FWRITE(xindex, index_buf.deleteflag, BUF_LEN, 0);
       if (cond_code != CCE)
          {sprintf(buf,"index File- Insert now");
           index_found = ISFALSE;
             }
       FUNLOCK(xindex);
         }

}    /*    end of insert   */

COUNT index_del()
{ empty(index_key_buf, 55);
  strncpy(index_key_buf,  index_buf.zipfilename,  55);
  BUF_LEN = sizeof(index_buf);
  BTSTAT=BTRV(B_DELETE, index_POS_BLK, &index_buf, &BUF_LEN,
                                        index_key_buf, 1);
  if (BTSTAT != 0)
     { sprintf(buf,"Delete Problem- index File %d",BTSTAT);
       do_msg();
       index_found = ISFALSE;
      }
   else
      {index_found = ISTRUE;
       }

    if(useppl)
      {BUF_LEN = 0 - BUF_LEN;
       FLOCK(xindex, 1);
       FREMOVE(xindex);
       if (cond_code != CCE)
          {sprintf(buf,"Problem with Index File- Delete");
           do_msg();
           index_found = ISFALSE;
             }
       FUNLOCK(xindex);
         }

}    /*    end of delete   */


COUNT index_create()
{  empty(btmsg,  30);
   empty(index_FILE,  16);
```

```
   strncpy(index_FILE, "index ", 8);

   FILE_BUF.REC_LEN                          =
                                          sizeof(index_buf);
   FILE_BUF.PAGE_SIZ                        = 1024;
   FILE_BUF.FILE_FLAG                       = 0;
   FILE_BUF.NDX_CNT                         = 2;
 /*  key #1  */
   FILE_BUF.KEY_BUF[0].KEY_POS              = 3;
   FILE_BUF.KEY_BUF[0].KEY_LEN              = 55;
   FILE_BUF.KEY_BUF[0].KEY_FLAG             =
          MOD|DUP|B_STR_TYPE;
   FILE_BUF.KEY_BUF[0].KEY_TYPE             = B_STR_TYPE;

 /*  key #2  */
   FILE_BUF.KEY_BUF[1].KEY_POS              = 78;
   FILE_BUF.KEY_BUF[1].KEY_LEN              = 55;
   FILE_BUF.KEY_BUF[1].KEY_FLAG             =
          MOD|DUP|B_STR_TYPE;
   FILE_BUF.KEY_BUF[1].KEY_TYPE             = B_STR_TYPE;

   BUF_LEN = sizeof(FILE_BUF);
   BTSTAT = BTRV(B_CREATE, index_POS_BLK, &FILE_BUF,
               &BUF_LEN, index_FILE, -1);
    if (BTSTAT != 0)          {
          sprintf(buf,"Problem creating %s status = %d",
                              index_FILE, BTSTAT);
          do_msg();
             }
}
```

Appendix B

```
/*
*   XSEND.C   - Batch file transfer between PC and HP-3000.
*
*   Compiled with Microsoft C v. 5.1 with /AS /Zp switches.
*    Must link in CALLXSM.OBJ.
*
*
*   Copyright 1989 Walker Richer & Quinn.
*
*/

#include <stdlib.h>
#include <stdio.h>

#define XSEND

#define TRUE  1
#define FALSE 0
#define byte  unsigned char

#define SLASH      '\x2F'   /*   /   */
#define SEMICOLON  '\x3B'   /*   ;   */
#define QUIETFLAG  '\x51'   /*   Q   */
#define CR         '\x0D'   /*  CR   */

#define     CONNECT_CMD     0x01
#define     SHUT_XFER       0x01
#define     XFER_ABORT      0x04
#define     USR_ABORT       0x55
#define     NO_XFER         0x41
#define     TOHOST          0x02
#define     FROMHOST        0x03
#define     DISCONN_CMD     0x02
#define     WRITE_CMD       0x03
#define     OPEN_CMD        0x08
#define     INIT_CMD        0x00
#define     RECV_CMD        0x04
#define     MASK            0xFF
#define     IN_PROCESS      0xFD
#define     QUEUED          0xFF
#define     BINARY          0x00
#define     ASCII           0x01
#define     DELETE_FILE     0x04

#define CONFIG_FILE "\\PPLTOOL\\PPL.CFG"

struct ftpb {
  unsigned char PB_REQ; /* put the command here          */
unsigned char PB_ID;    /* returned by init, ids user    */
unsigned char PB_HANDLE;/* this will be returned         */
unsigned PB_PC_SPEC_LEN;/* length of pc file name        */
char far *PB_PC_SPEC;   /* ptr to ASCII pc file name     */
```

```
unsigned  PB_BUF_LEN;   /* buffer length                    */
char far     *PB_BUF;   /* ptr to buffer ( for host filec)*/
unsigned  PB_XFER_TYPE; /* xfer type (currently ignored)  */
unsigned  PB_ATTR_LEN;  /* length of file attribute string*/
char far *PB_ATTRIBUTES;/* MPE file attributes             */
unsigned char PB_FLAGS; /*                                 */
unsigned char PB_OVERWRITE; /* overwrite flag              */
unsigned char PB_COMPRESS;/*                               */
unsigned char PB_STAT_CODE; /* status code returned        */
unsigned      PB_STR_LEN;   /* string length               */
char far     *PB_STRING;    /* ptr to message string       */
long int      PB_UNITS;     /* file size (unsigned chars  */
long int      PB_XFERRED; /* units (chars or records)
                                 xferred  */
} ftparm_block;

struct ftpb  far *ftfar_ptr =&ftparm_block;

int  quiet=FALSE;
char pcfilespec[80], hpfilespec[80], attributes[80];

#if defined(XSEND)
    char unit_message[] = "Characters transferred: ";
    char prog_name[] = "XSEND";
    int  direction = TOHOST;
#else
    char unit_message[] = "Records transferred: ";
    char prog_name[] = "XRECEIVE";
    int  direction = FROMHOST;
#endif

/* function prototypes */

extern byte  GETXFR();
extern byte  CALLXFR(struct ftpb far *);
void evaluate_arguments(int, char *[]);
void initialize_ppl(void);
void send_files(void);
void shutdown(void);

main(int nargs, char *args[])
{
     evaluate_arguments(nargs, args);
     initialize_ppl();
     send_files();
     shutdown();
     exit(0);
}

void evaluate_arguments(int nargs, char *args[])
{

int minargs=2, i;
```

```
      if (args[1][0] == SLASH && toupper(args[1][1]
                     ==QUIETFLAG) {
         minargs = 3;
         quiet   = TRUE; }

      if (nargs < minargs) {
         if (!quiet)
             exit(1); }

      if (quiet) strcpy(pcfilespec, args[2]);
      else       strcpy(pcfilespec, args[1]);

      strcpy(hpfilespec,"\0");
      if (quiet && nargs == 4)  strcpy(hpfilespec,args[3]);
      if (!quiet && nargs == 3) strcpy(hpfilespec,args[2]);
      if ((strncmp(hpfilespec,"\0",1)) == 0)
                     strcpy(hpfilespec,pcfilespec);

      strcpy(attributes, "\0");
      for (i=0;i<strlen(hpfilespec);i++)
          if (hpfilespec[i] == SEMICOLON) {
             strcpy(attributes,&hpfilespec[i+1]);
             strncpy(hpfilespec[i], "\0",1 );
             break; }

}

void initialize_ppl(void)
{
byte cond_code=0;

    if (!quiet) printf("\nInitializing Transfer...\n");

    cond_code = GETXFR();  /* get address from pplcom.sys */
    if (cond_code == 0) {  /* if OK, initialize xfer      */
       ftparm_block.PB_REQ     = INIT_CMD;
       ftparm_block.PB_BUF     = CONFIG_FILE;
       ftparm_block.PB_BUF_LEN = strlen(CONFIG_FILE);
       CALLXFR(ftfar_ptr);
       cond_code=ftparm_block.PB_STAT_CODE;
    }

    if (cond_code != 0) {
       if (!quiet)
           printf("initialize: error %d\n",cond_code);
       exit(2); }

}

void send_files(void)
{
char errmsg[80];
byte cond_code;
int  len, transtype=BINARY;
```

```
long xferred=01;

    if (!quiet) printf("\nStarting file transfer...\n");
    strupr(attributes);
    if (strstr(attributes,"ASCII") != 0) transtype=ASCII;
    ftparm_block.PB_XFER_TYPE    = transtype;

#if defined(XSEND)
    if (len = strlen(attributes)) {
       ftparm_block.PB_ATTR_LEN    = len + 1;
       ftparm_block.PB_ATTRIBUTES  = attributes;
       ftparm_block.PB_XFER_TYPE   =
              ftparm_block.PB_XFER_TYPE|2;
       attributes[len] = '\xFF'; }
#endif
    ftparm_block.PB_REQ          = direction;
    ftparm_block.PB_OVERWRITE    = DELETE_FILE;
    ftparm_block.PB_PC_SPEC      = pcfilespec;
    ftparm_block.PB_PC_SPEC_LEN  = strlen(pcfilespec);
    ftparm_block.PB_BUF          = hpfilespec;
    ftparm_block.PB_BUF_LEN      = strlen(hpfilespec);
    ftparm_block.PB_STRING       = errmsg;
    ftparm_block.PB_STR_LEN      = 80;
    ftparm_block.PB_XFERRED      = 0;
    ftparm_block.PB_UNITS        = 0;
    strcpy(errmsg, "\0");

    CALLXFR(ftfar_ptr);
    while((cond_code=ftparm_block.PB_STAT_CODE) == QUEUED)
                  continue;

    while (TRUE) {
       cond_code = ftparm_block.PB_STAT_CODE;
       if (cond_code != IN_PROCESS) break;

       if (ftparm_block.PB_XFERRED != xferred) {
          xferred = ftparm_block.PB_XFERRED;
          if (!quiet)
             printf("%s%ld%c",unit_message,xferred,CR); }
    }
    if (cond_code == 0 && !quiet) printf("\nComplete.\n");
    if (cond_code != 0) {
       if (!quiet) printf("\nErr transfer: %d\n",cond_code);
       shutdown();
       exit(3); }
}
void shutdown(void)
{
     ftparm_block.PB_REQ = SHUT_XFER;
     CALLXFR(ftfar_ptr);
}
```

Database standards: Rallying points

F. Alfredo Rego

Adager

Sun Valley, Idaho
83353-0030
U.S.A.

Dr. Edgar F. Codd established a sound mathematical foundation for database management with his relational model. Unfortunately, many suppliers of database management systems, claiming a relational pedigree, have twisted the relational ideas to suit (or justify) their implementations and to claim that they "are" standard (or "follow" the standard).

Some people believe that anything that has "SQL" in it is ipso facto "relational". This is a consequence of the widespread belief that SQL is an integral part of the relational model for database management when in fact it is not. Most people are led to believe that there is such a thing as a standard SQL. Do I have a surprise in store for them!

Historical background

In early 1986, I met with several Hewlett-Packard executives in Cupertino who were very excited about the name that they had devised for their new strategic database product, ALLBASE. "The beauty of this concept," they said, "is that it will integrate the best practical aspects of IMAGE with the best theoretical aspects of the relational model for database management." They asked me if I liked the name and the concept behind it. I said that both the name and the concept were wonderful. I still think so now, after more than five years.

A recent conversation I had about the relational model for database management with its creator, Dr. Edgar F. Codd, makes HP's initial conception of ALLBASE even more relevant.

My conversation with Dr. Edgar F. Codd

At the SCRUG meeting in Pasadena, on May 9, 1991, I interviewed Codd in the setting of a public forum. I began my conversation with Codd by explaining that I had selected the session's title, "Understanding Databases," because it was ambiguous. We can choose to interpret the word "understanding" either as a verb or as an adjective. As a verb, "understanding" means that we are taking some action to try to understand what databases are. As an adjective, "understanding" means that we are talking about databases that treat us in a motherly fashion, that are tolerant, compassionate and sympathetic, that never break down, that always perform beyond the call of duty, that don't require expensive maintenance.

I then asked Codd if he knew of any such magically understanding databases. Laughing, he said, "Not at all." Addressing the audience, I asked if anybody else knew of any such magically understanding databases. More laughter. Funny, I thought, after having waded through all the glossy literature. Given this reality, we decided to interpret the word "understanding" as a verb. I invited Codd to help all of us in our efforts to try to understand what databases are.

As the basis for our discussion, I selected some key ideas from Codd's recent book (*The Relational Model for Database Management, Version 2*, Addison-Wesley, 1990).

The evidence would suggest that few people understand the relational model for database management (although many people certainly know SQL very well). I encouraged everyone to read and to study Codd's book because it brings together, under one cover, his fundamental ideas. To illustrate, I quote from the book:

"Four important points concerning relations follow:

1. every relation is a set;

2. not every set is a relation;

3. every relation can be perceived as a table;

4. not every table is a correct perception of a relation.

Designers of the relational DBMS products of many vendors appear to be ignorant of these facts or to have ignored them" (page 27).

"Of course, in many of the relational DBMS products on the market today, support for the integrity features of the relational model is quite weak. This weakness reflects irresponsibility on the part of DBMS vendors" (page 435).

Given these facts, I asked Codd about his feelings whenever people use "SQL" as synonymous with "the relational model for database management." Codd proceeded to clarify the myths surrounding SQL, with particular attention to the overselling of SQL as the standard for relational database management systems. "SQL is just a data sublanguage invented in late 1972 by a group in IBM Research, Yorktown Heights, NY. Although it was claimed that the language was based on several of my early papers on RM, it is quite weak in its fidelity to the model," Codd said.

"How was SQL ever adopted as an ANSI and ISO standard?", I asked. Codd replied, "That's an excellent question; I wish I knew the answer." (For those interested in pursuing this issue, Codd has devoted chapter 23 in his book to discussing the serious flaws in SQL.)

Because Codd had dedicated his book "To fellow pilots and aircrew in the Royal Air Force during World War II," I knew that he would, as a pilot, appreciate the fact that airplanes are amazing things that come in all shapes and prices. There is one airplane that went around the world without refueling; it was very slow, extremely uncomfortable, as fragile as a kiss and, therefore, unable to go through the storms and the turbulence that commercial jets usually encounter. There are huge, slow cargo planes. There is the Concorde. There are business jets. There is Air Force One, with a bed and (we would assume) a shower.

When I asked Codd about his recommendation for the "standard airplane" that everybody must have, he said, "Such a thing does not exist." While on this topic, he referred to page 22 of his book: "I believe that the days of monstrous programming languages are numbered, and that the future lies with specialized sublanguages that can inter-communicate with one another." To me, this sounded very similar to what is expected of open systems, whose existence depends on specialized things that inter-communicate well, as opposed to monstrous things that try to be all things to all people.

Codd shared with us his observations about the main shortcomings of the wishful implementations of the relational model, using IBM's DB2 to illustrate. "DB2, with about 50% compliance with the relational model, is the most faithful implementation," Codd said. "But even DB2 is still a long shot. The main problem is that all so-called relational database management systems do not support the fundamental features of the relational model. The fact that they may (or may not) support some other features does not relieve them of the responsibility of supporting the fundamental features. Without sound fundamentals, any structure is bound to collapse eventually. Everywhere, users are losing their patience. Things are taking too long and are too expensive. There is a deluge of marketing hype," Codd complained.

Regarding my question about any hope for the convergence of these so-called relational database management systems towards the relational model, Codd explained that, "Due to some fundamental decisions that the implementors had made early on in the game, it would be very difficult for them to converge towards the relational model."

I then brought up an issue that is highly relevant to the members of the HP3000 community who have developed high-quality, reliable applications based on IMAGE. Why should these people migrate to a poor-quality, unreliable, expensive and non-compliant so-called relational DBMS? Codd had only a couple of minutes to address this question, as it was the last question before lunch. Codd quickly mentioned that "Any conversion is a very expensive proposition in terms of labor costs, since automatic conversions are ineffective and need a great deal of babysitting." Codd did not foresee labor costs decreasing. "Therefore," he reasoned, "even though it would be an expensive migration, everyone should convert to a so-called relational implementation as soon as possible." En route to lunch, an IMAGE user approached us noting that the word "possible" might best be interpreted to mean "economically feasible."

There is no question in my mind about the ever-rising costs of conversion, but I do not agree with conversion for conversion's sake. I believe that there is only one valid reason for converting: to escape from a poor database management system that is not able to support vital applications. And then, people should only convert to a clearly outstanding database management system. Anything else is an exercise in futility.

What are we to do with SQL?

Codd himself does not think that SQL is a particularly outstanding ambassador of the relational model. On the contrary, he says on page 444 of his book, "Vendors, however, are forging ahead with both 'products on top' and 'distributed RDBMS products', disregarding errors in present relational DBMS products. All the evidence indicates that they will continue to do so. An inevitable result is that existing errors will become more difficult to fix, because more products and more users will be affected. Over time, the defects and deficiencies in the present versions of SQL will become totally embedded in relational DBMS products. It is important to be aware that, first, the language SQL is not part of the relational model. Second, the defects and deficiencies in SQL correspond closely to the various departures of SQL from the relational model."

During lunch, immediately after our public conversation, Dr. Codd and I discussed the SQL issue. As an example of the confusion, he mentioned that, during a visit he paid to one of the various SQL-standardizing committees, the members of the committee agreed on only one thing: they agreed to disagree with Codd.

Regardless of SQL's weaknesses, it is obvious that it is today's lingua franca for databases. The desire for a single, common language is nothing new. For instance, scientists have spoken all kinds of native languages, yet they have felt the strong need to inter-communicate. One solution would have been for each scientist to learn all of the languages spoken by everybody else. Because this would have been unlikely, scientists "agreed" (voluntarily or not) on *some* common language. In this manner, each scientist had to speak at most two languages, the scientist's native language *and* the common language of the day.

There are two important points about common languages:

1. The existence of a common language does *not* preclude the existence of "native" languages.

2. A common language is not forever, as it depends on political factors.

Because political factors are constantly shifting, several common languages have come and gone in the Western scientific community: Greek, Roman, German, French, English. Today, English is the common language for computers. As a highly-structured subset of English, SQL appears to be the emerging common language for database inter-communication.

Given these facts, we might as well learn SQL, even if we don't approve of it. And we might as well teach SQL to our favorite database management system.

Hewlett-Packard's ALLBASE idea to the rescue

The beauty of Hewlett-Packard's ALLBASE concept is its *inclusive* quality. HP is making significant progress toward fulfilling this ALLBASE promise. For instance, ALLBASE/Turbo CONNECT currently allows SQL read-only access to IMAGE databases. Right now, HP is seriously considering the obvious evolution of ALLBASE/Turbo CONNECT: SQL read *and* write access to IMAGE databases.

How serious is HP about implementing SQL read/write access to IMAGE databases? HP is very serious, indeed, but it needs *your* input to help define the future directions for its database programs. On this topic, *The HP Chronicle* (on page 20, May 1991 issue) has an article, "HP seeks customer input on databases", quoting Doug Dedo, HP IMAGE product line manager. Here is a sample of noteworthy items in the article:

Seeking input from customers to help define future directions for its database programs, Hewlett-Packard has released a survey gauging customer needs... "I think that this is a good opportunity to really get into the heart of the TurboIMAGE program and have a voice be heard in a productive, proactive way," said Doug Dedo. "[The survey comes] in a time where it can be incorporated into business planning activities."

The first area [of the survey] involved "Just how do people want TurboIMAGE itself to move into the 21st Century," he said. "We are definitely moving it there... Some people call it mature. Competitors think it is obsolete and yet we've got a phenomenally large customer base and a huge set of applications that are really providing valid business solutions today."

In the second area, HP officials sought input about ALLBASE/Turbo CONNECT write

[access], the bridge between the relational and the TurboIMAGE world. "Our ALLBASE SQL product has an ALLBASE/Turbo CONNECT that links it to TurboIMAGE so that you can do an SQL query and be able to pull information simultaneously out of the relational database as well as out of a TurboIMAGE database," Dedo explained.

Within the last six months, customers have expressed a desire to be able to write back into the TurboIMAGE database. "So the second piece in the questionnaire was to start getting more detailed data on what [customers] see and how they would use the write capability."

"The goal of the survey is to compile information on user needs so that these needs can be met by HP in future product releases," Dedo said.

Breaking free from IMAGE's physical limitations

IMAGE's physical limitations have to do with the way HP has chosen to implement (or *not* to implement) its various design criteria. The April 1991 *Interact* includes an article by Wirt Atmar (of QueryCalc fame) called "The future of IMAGE on the HP3000 is SQL." Atmar quotes a senior Hewlett-Packard executive:

"The problem is that TurboIMAGE has been tuned for over 15 years and there are not many ways we can improve it anymore." Wim Roelandts, HP vice-president and general manager of the Computer Systems Group, made this statement last year.

But this is not true. A number of rather simple enhancements to IMAGE would make dramatic differences in its use, in its performance, and in the minds of its users. Now that HP has graciously agreed to implement the critical item update enhancement, the foundation has been laid for a number of truly significant enhancements to IMAGE...

These few enhancements, which would not only revolutionize the use of IMAGE but also ensure its future competitiveness, are among the most commonly touted advantages of SQL databases. But they have nothing to do with SQL per se. They should be part and parcel of any competitive database structure. IMAGE is particularly amenable to these modifications. And none of them are difficult to accomplish. HP already has all of the code in hand to implement each enhancement.

Atmar's article points to the problem *and* to the solution, the implementation of the original ALLBASE idea. Dedo's survey is a step in the right direction. I applaud HP's willingness to give IMAGE the ability to inter-communicate.

Giving IMAGE the ability to inter-communicate

The bottom line is: Because IMAGE needs to inter-communicate with other database management systems, the issue boils down to providing a read/write SQL interface for IMAGE today (whether we like SQL as a lingua franca or not). This will be a significant step in fulfilling the original Hewlett-Packard promise for ALLBASE: Standardized access to IMAGE databases *and* to relational databases. This will provide Hewlett-Packard users and applications developers with the best of both worlds. What a wonderful idea.

Paper Number : 3232

Relational Database Design

Jo-ning Ta

Oracle Coroption

400 Oracle Parkway

Redwood Shores, CA 94065

(415) 506 - 2974

1 Introduction

This paper presents basic techniques for designing a relational database with emphasis on data integrity by using the result from an entity relationship modelling. Traditionally, data integrity is specified and enforced within the database applications. Whereas today, the data integrity has been added to the SQL language by the ANSI 89 standard(ANSI X3.135-1989). It means the relational systems start to embed more data integrity into the database kernel. The relational systems now can guarantee better integrity and consistency. The application development productivity can be improved because less coding has to be done on data rule checking. Better performance can be expected because data integrity can be implemented and optimized more efficiently in the database kernel.

This paper is mainly emphasizing on the logical database design. The physical database design is not going to be addressed. The audience is assumed to have some basic knowledge of SQL and relational database.

2 Entity Relationship Modelling

Entity Relationship Modelling is a means of defining and controlling the definition of the information needs. The definition can be used as the framework in database design. Briefly, Entity Relationship Modelling involves identifying

- the things of importance in an organization(Entities),
- the properties of those things(Attributes),
- and how they are related to one another(Relationships)

The output of the modelling, Entity Relationship Diagram, can be used for both relational(e.g. Oracle, Allbase/SQL) and non-relational system (e.g. TurboIMAGE).

The following elements are important in Entity Relationship Modelling:

- Entity

  A thing or object of significance, whether real or imagined, about which information needs to be known or held.

- Relationship

  A named, significant association between two entities. There are three types of relationship, which are One to Many(1:M), Many to One(M:1), Many to Many(M:M). Many to Many relationships are common during early strategy or analysis periods. By the end of the analysis stage, they shall all be resolved. Resolution is achieved by means of inserting a new intersection entity between the two ends.

- Attribute: Any description of an entity

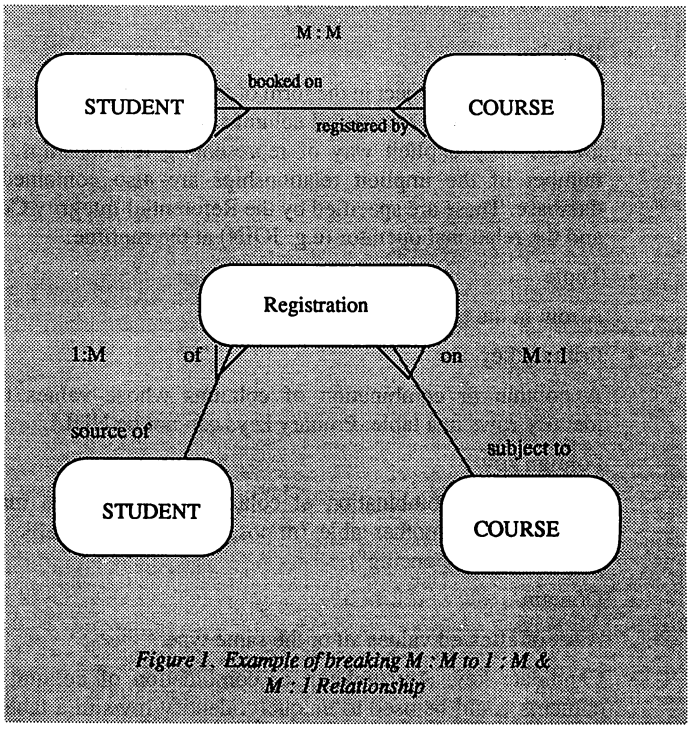


Figure 1. Example of breaking M : M to 1 : M & M : 1 Relationship

3 Relational Terminology

- Relation

  A mathematical object in the form of a table, with distinct unordered rows, and atomic, unordered columns. In a relational system, table are the only explicit way of representing relationships. A large number of the implicit relationships are also contained in the database. These are specified by the Referential Integrity Constraint and the relational operator (e.g. JOIN) at the runtime.

- Tuple

  A row in the table.

- Primary key

  A column or combination of columns whose values uniquely identify rows in a table. Primary keys can not be NULL.

- Foreign key

  A column or combination of columns whose values match the primary key of another table. (or possible of the same table which is called "self-referencing")

- Domain

  A set of allowed values all of the same type.

- Candidate key: Any column or combination of columns whose contents could be used to uniquely identify rows in a table. Every relation has at least one candidate key. When multiple candidate keys exist, the designer chooses one to become the primary key. Then the remaining candidate keys(if any) are alternate keys. It may be used as a secondary access to the data.

- Composite key: A combination of columns as a key.

4 Mapping from entity relationship to relational database

1. Each entity is mapped into a table. A useful standard is to use the plural from of the entity for the table name.
2. Each attribute is mapped into a column of the same name with the proper types in the table that the entity has just been mapped into. Optional attributes become NULL columns. Mandatory attributes should be non-NULL columns.
3. The components of the unique identifier of the entity become the primary key of the table. Remember also that an entity may be uniquely identified by a combination of attributes and/or relationships. When relationships are used follow along the relationship and bring down as columns a copy of the unique identification components of the entity at the far end of the relationships as part of the primary key.(This may be recursive until attributes are eventually found.)
4. *Many to one*(and *one to one*) relationship become foreign key. That is, bring down a copy of the unique identifier of each referenced entity from the *one* end and use as columns of foreign key.Optional relationships create NULL columns.Mandatory relationships create not NULL columns.

Following is an example of mapping:

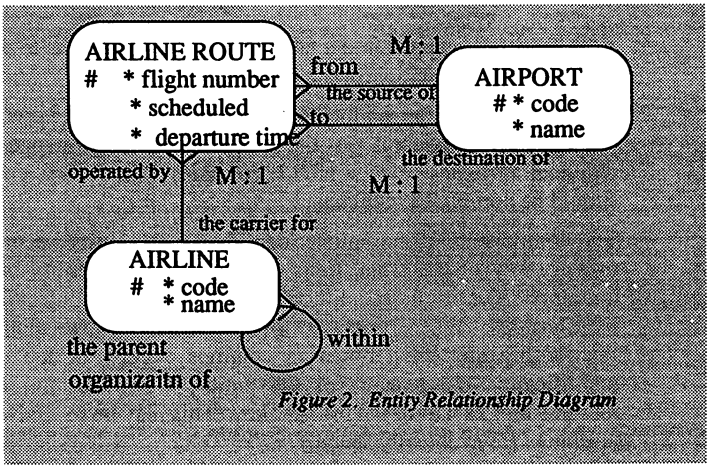


Figure 2. Entity Relationship Diagram

	STEP
Create table airports(	1
code char (4) PRIMARY KEY,	2/3
name char (40) not NULL,	2
)	
Create table airlines(	1
code char(4) PRIMARY KEY,	2/3
name char (40) not NULL,	2
Parent_airline_code char (4) NULL,	4
reference airlines(code),	
)	
Create table airline_routes(	1
flight_number number (4),	2
airline_code char (4),	3
from_airport_code char (4) not NULL,	4
to_ariport_code char(4) not NULL,	4

```
scheduled date  NULL,                                    2
departure_time  date NULL,                               2
constraint  PRIMARY KEY                                  3
    (flight_number, airline_code),
constraint FOREIGN KEY(airline_code)                     4
    references airlines(code),
constraint FOREIGN KEY                                   4
   (from_airport_code) references airports(code),
constraint FOREIGN KEY                                   4
   (to_airport_code) references airports(code),
)
```

5 Data Integrity

Data integrity guarantees that data in a database adheres to a predefined set of constraints. It ensures that users only perform operations which leave the database in a consistent state.

There are two types of implementations for enforcing data integrity.

- Declarative

  Declarative constraints and actions can be specified in the CREATE TABLE statement. At the end of each multiple step update(after all actual updates done in a UPDATE statement), the constraints are enforced and actions are taken.

- Procedural

  Triggers and application code can both specify the conditions and actions. Trigger is a user-defined SQL block associated with a specific table, and implicitly fired(executed) when a triggering statement is issued against the table. The procedural implementation is not covered in this paper.

6 Types of data integrity

- Entity Integrity

  It governs the data content of a single row. It serves as an intra-table constraint.

  not NULL: columns must have values.

  default: default value for columns not specified on INSERT.

  unique: no two rows/fields have the same value.

  check: must be true of all inserts and updates on the table.

```
create table emp(
  empno       number(4),  /* PRIMARY KEY */
  ename       char(20),   not NULL,
  deptno      number(4),  /* FOREIGN KEY */
  mgrno       number(4),  /* FOREIGN KEY */
  salary      number(6)
    constraint sal_range check(salary < 9000),
  hiredate    date        default SYSDATE,
)
create table dept(
  deptno      number(4)   /* PRIMARY KEY */
  dname       varchar2(20) unique,
  deptmgr     number(4),  /* FOREIGN KEY */
)
```

- Referential Integrity

  It enforces a master/detail relationship between tables based on primary/foreign keys. It ensures that any detail record must always have a corresponding master when insert, update, and delete statements are issued against the table.

For TurboIMAGE, the master/detail relationship is modelled by the manual master and detail set.

When there is a delete or update on the primary/unique key that the foreign key is referencing to, there are two actions which can be specified. One is DELETE CASCADE. What it means is if a master row is deleted, foreign key rows referencing the deleted row's key are deleted automatically. The other is UPDATE/DELETE RESTRICT. This basically makes sure a master row cannot be updated or deleted if referenced by any foreign keys when the UPDATE/DELETE statement is issued against it.

Following is an example of specifying the Referential Integrity for the emp and dept tables.

```
create table emp(
  empno        number(4) PRIMARY KEY,
  ename        char(20),     not NULL,
  deptno       number(4)
     constraint work_in FOREIGN KEY (deptno)
        references dept(deptno) on DELETE CASCADE,
  mgrno        number(4)
     constraint work_for FOREIGN KEY (mgrno)
        references emp(empno),
  salary       number(6)
    constraint  sal_range check(salary < 9000),
  hiredate     date          default SYSDATE,
)
create table dept(
  deptno       number(4) PRIMARY KEY,
  dname        varchar2(20)  unique,
  deptmgr      number(4)
     constraint run_by FOREIGN KEY (mgrno)
        references emp(empno),
)
```

7 How to enforce the referential integrity if the relational system does not directly support it

The crucial importance of primary and foreign keys to relational database design has been stressed. Following is a way that you can enforce the primary and foreign key disciplines yourself if the relational system that you work with does not provide the direct support. (For Oracle, Version 6 only accepts the syntax for Referential Integrity. It does not enforce it in the RDBMS. Version 7 will support both the Entity Integrity and Referential Integrity)

For each primary key in your design:

- Specify NOT NULL for each field in the primary key.
- Create a UNITQUE index over the combination of all fields in the primary key.
- Ensure that this index is in existence right after the table creation. This is to make sure all inserts and updates of the primary keys will follow the discipline.

For each foreign key in your design:

- Find out if the foreign key can be null. Specify NOT NULL for each field in the foreign key if it is not allowed to be NULL.
- Create an index over the combination of all fields in the foreign key. If the foreign key and its matching primary key will often be used as the basis for join operation.
- Take the foreign key constraints as part of the application specification. Have one module/routine which does the updates or inserts of the foreign key. However, have every one who wants to insert/update the foreign keys all go through this same routine.
- Use the authorization mechanism to control the insert/update/delete to the primary key table and the foreign key table.

General business rules:

It specify complex business conditions. Traditionally, all of the business rules are specified in the applications. With the offering of "triggers" and stored procedures", a big percentage of the general business rules can move to the databases.

- Sequence Numbers

  System generated unique identifiers.

  The sequence numbers can be produced by means of a table, each row of which contains the name of a sequence and the next value to be allocated. However, if many tables with system-generated identifiers are subject to frequent INSERTs, having a single sequence table may lead to contentions. In such a case, a considerable improvement can be made by having a separate control table for each sequence to be generated. Oracle Version 6 provides its own sequence number generator which multiple users may generate unique integers. Sequence numbers may be used to generate primary keys automatically.

  Example:

  CREATE SEQUENCE [user.] sequence

  INCREMENT BY 1

  START WITH 1

  The numbers are generated when the pseudo-column NEXTVAL is accessed.

  INSERT INTO table

  VALUES

  ([user.] sequence.NEXTVAL)

- Selective Denormalization

  The process of normalization and the technique of Entity Relationship Modelling both aspire to a design which is devoid of redundancy. The reason for eliminating redundancy is to remove the difficulties associated with updating duplicated information.

  On the other hand, a normalized design involves separate table which may have to be joined to satisfy given queries, and this in itself is an overhead.

So the database designer may introduce some redundancy for performances when two or more tables are:

* relatively static with respect to DML.
* relatively active with respect to cross-table queries.

- Artificial Primary Keys

  In practice, it may be hard to find primary keys. Many entities, such as people, do not come with unique identity codes. Also, some unique identifiers in a table may turn out to very complex, a combination of many columns or even the whole row(composite key). The composite keys can lead to redundancy if a composite key is used to link the relationship between a master and detail. Also, it is clumsy that the users have to write their queries like:

```
WHERE master.key_part1 = detail.key_part1
AND   master.key_part2 = detail.key_part2
AND   master.key_part3 = detail.key_part3
.....
```

  In these cases, the designer may consider specifying an invented artificial primary key. The user can use the sequence numbers as the artificial primary keys. One disadvantage is that if the original composite key content in the master relation is always needed when querying the detail relation contents, then more joins will now be needed.

9 Use of CASE tools

Computer-Aided Software Engineering(CASE) tools are available from many different vendors. Most offer some form of entity relationship of data modelling capability. A large amount of the database design can be carried out automatically by these CASE tools. However, this is only a starting point, as the database design now needs careful scrutinizing to ensure that it provides full support in a performance/space efficient manner for the applications. This may require careful denormalization, controlled replication across a network, and detailed physical design of indexes and disk utilization.

10 Conclusion

Database design has traditionally been regarded as a very difficult task, requiring very specialized skills. With the data integrity enhancement in SQL, relational database systems simplifies the relational database design task. The designer can describe the complex data requirements in a very declarative way, and application programmers can skip many data checking and enforcement in the code. All these allow us to concentrate in building a more complex database now and in the future.

TITLE: TurboIMAGE/XL's Standard Interface to Third-Party

AUTHOR: Eric Savage

Dynamic Information Systems Corp.

652 Bair Island Road
Suite 101

Redwood City, CA 94063

(415) 367-9696

FINAL PAPER NOT AVAILABLE, HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

PAPER NO. 3233

Paper Number 3234

CASE ME
Computer Aided Software Engineering Tools for Managing User Expectations in a Software Migration Project

By

Garry L. Smith
Charles McMurray Company
2520 N. Argyle
Fresno, CA 93727
(209) 292-5751

This paper presents the use of Computer Aided Software Engineering(CASE) tools to Manage user Expectations(ME) of a finished software product. After decades of meeting the end user needs, the computer software industry has only in the last five years started to provide tools for the software developer. Now at last CASE tools are out of the infancy stage and actively being used by Information Systems departments. This paper focuses on the use of CASE tools to upgrade an 'over-the-hill' software application.

Replacing software that is at the end of its life-cycle is a task that will continue to be at the top of the agenda for many Information Systems departments. Software which has been modified over time to meet changing business strategies and to satisfy external requirements leads to a patch work of highly customized application software.

This paper details the use of Data Flow Diagrams(DFDs) and data dictionaries to document the entity relations, the system functions, the operational flow of data and the existing manual procedures of the company. This paper does not attempt to teach the concepts of structured analysis, data flow diagrams, nor data dictionaries, but merely presents how these tools were used during a software migration process. The CASE tools provide a methodology for standardizing and maintaining documentation about the organization's management information system.

This paper describes a software migration project for the Charles McMurray Company which had a custom software application which had evolved over a fourteen year period and over two different vendor hardware platforms. The lack of integration, standards and program documentation propelled management to make a decision on acquiring a new software application.

Part and parcel of the Information Systems life cycle is the replacement/upgrade of hardware and software. The McMurray company had recently acquired a new hardware system(HP-3000/922RX) so there was no question as to the hardware on which the application software needed to run on. There are four major software replacement strategies: 1) modify the existing system to provide integration and data integrity, 2) develop a new system, 3) purchase a new integrated software system and 4) purchase a system and modify the application to meet the company's needs. This is the basic scenario faced by many Information Systems department at the end of the software life cycle. The four alternatives are typically generic, and with the application of internal values/politics and the ubiquitous cost/benefit analysis each company reaches its own course of action. The optimal solution for Charles McMurray Company was to find an integrated software application the provided 80% of the company's information processing requirements and provide the flexibility for modification. Therefore, McMurray has chosen option number four(4), modify the 'off-the-shelf' system.

Having decided on the appropriate course of action, a project plan was created and signed by the president of the company(see appendix "A"). A project plan is crucial to the success of any project involving more than two people and greater than 40 man-hours. This project plan summarized the rational for acquiring a software package, defined the project boundaries, the objectives, the approach to be used, the resources required, individual's responsibilities and a general time-line of target dates for the major milestones needed to complete the project. This project plan was then presented and distributed to all the employees of the company. Future enhancements were also discussed and input was solicited from all employees at the meeting.

The use of a project plan is imperative to maintain project focus and timeliness. The detailed tasks and milestones provided by the project plan allow for periodic reviews of the project's progress. The official approval of the project by the company president provides the license for the Information Systems department to collect data, interview employees, and allocate company resources to achieve the completion of the project.

Before serious evaluation of vendor packages could occur there were several questions that needed to be resolved; what is the current functionality being provided by the existing software?, what are the problem areas?, what areas can be improved upon? The projected was started in September of 1990. By using some simple front-end CASE Tools such as a Data Flow Diagrammer, Data Dictionary, Word Processor the existing system was documented. Using the diagrams generated(see appendix "B") the major

deficiencies and lack of integration were highlighted for the users. After having analyzed the existing system, a new proposed system generated(see appendix "C"). Diagram modifications were easily accomplished by using a diagramming tool. The analysis occurred within about a two month time-frame.

It is important to clarify the context in which the acronym CASE is used. CASE is the new buzzword in information technology, just as was 4GL and Relational. What many vendors call CASE are in essence back-end CASE Tools used for programming productivity. Front-End CASE is typically used for analysis productivity. The acronym I-CASE stands for Integrated Computer Aided Software Engineering, which includes a both a front-end and back-end CASE tools integrated to form on seamless process from analysis-design through database maintenance and program revisions.

The use of front-end CASE tools greatly facilitated the documentation of existing system functionality. The diagrams in Appendix show what currently existed and what will be. These can be understood by the user and are easily modified. The use of the data dictionary(see appendix "D") provides the documentation of data elements and their attributes for the systems department to validate detailed program logic.

In the development of complex software systems from the 'ground-up' integration between front-end(design) and back-end(program productivity) is important. Integrated CASE tools help minimize the software development costs and increase the completeness and cohesion of the final product. There are few products on the market today that accomplish this task. Those products that are available are typically beyond the budgets of most small I.S. shops. Therefore, Charles McMurray uses a combination of front-end and back-end CASE tools to accomplish the software migration. For a front-end tool Charles McMurray uses a personal computer based data flow diagrammer with data dictionary capabilities.

In the past CASE tools comprised of: 4GLs, Word processors, copy libraries, Full Screen Editors. Today, any shop that does not provide personal computers to their programmer/analysts and other senior Information systems staff is severely impeding the productivity of their organization. The concept of the information **workbench** which uses the micro-computer for its memory and processing capabilities, quality of graphics, networking, multi-tasking and connectivity as the major hardware tool is of vital importance to the CASE system.

At Charles McMurray all Information Systems staff are equipped with Vectra compatible 386 machines with multi-sync color monitors

and a minimum of 40 megabytes of disk. This allows the use of the full array of products available for the MS-DOS/Windows and/or New Wave environment. The ability to upload and download information between systems, capture screens for documentation, present structured english of detailed conversion logic/algorithms and document the cross-reference of data elements between the old to the new system reduced conversion errors. The documentation generated and discussed with the user provided a method for involving the user in the software migration process.

The search for software vendors occurred in parallel with the specifications definition and three final vendor packages identified to reviewed in detail by December of 1990. These packages were reviewed based on the primary criteria of cost versus functionality. The other major criteria included availability of source and ease of system modification. At the end of December a software vendor was selected and an integrated General Ledger, Accounts Payable, Accounts Receivable, Order Processing, Inventory and Purchasing system was delivered to the Information System department.

Since cost was a primary factor in the selection criteria the acquired system was written in COBOL. The overwhelming task of modifying of several hundred COBOL programs, was slightly reduced by the extensive use of copy libraries and the purchase of a full screen editor for the HP-3000. Here again, any systems department which does not provide a full screen editor on the HP-3000 for their programmers is losing the full potential of their programming staff. Other future tools that will be proposed during the budget approval process include: revision/module management system, system wide dictionary, database tools, and a report writer. The future acquisition of a back-end CASE tool such as a report writer, will provide some additional gains over using just the QUERY/V capabilities.

The continued contact between systems department and the users is accomplished by structured walk-throughs which provide a forum for discussing the project progression and any issues or problems that need to be resolved. Since, one of the goals of the software migration was to continue to provide the Charles McMurray users with the same functions currently available on their existing system, the statement - "We want the report [and/or screens] to look like the old system." was common during these meetings. By stressing the concept of **functionality not similarity** many reports and screens did not have to be re-written. A software migration project is the best time to question the user on why they are performing certain procedures and suggest changes to eliminate redundancy. The use of data flow diagrams also helped implement changes in manual filing procedures, since the new system will

maintain and track most of the data required by the users. Many of these manual procedures were developed to overcome the deficiencies of the old system.

As the conversion progresses to a production environment additional enhancements are requested by the user. This is due to their increasing familiarity with the new system and the revelation of the new system potential. An important part of managing the users' expectations is based on the project plan and the presentation of data flow diagrams indicating the functionality that will be provided in the initial migration. Therefore, the initial project plan details only the conversion process of migrating from the existing software system to the new software environment. In order for management and the users to understand the impact of the new system on the organization and communicate the changes in information flow which will occur as a result of the new software, the data flow diagrams were helpful as a visual model of the new software(See attachment "C").

Depending on the company's financial resources and the success of the original migration project the Information Systems department should have the support of the company's management and user community for continuing to increase the information technology used in the company. The long term goals for the software replacement include: increase customer service, flexibility to allow for company expansion without major increases in personnel. The new software will provide the ability to implement cost effective information technology to increase McMurray's competitive advantage. The possibility exists for decision support systems for inventory pricing and purchasing. The new software will provide capability for customer and vendor analysis reports, begin to automate inventory control by using bar codes, and finally install lap-top personal computers with each salesman for inventory pricing, availability and customer order entry. All of these projected changes can be readily incorporated into the documentation as the result of using automated CASE tools and can be reviewed for completeness and consistency.

Conclusion

The basic modules of CASE technology have been in use for years by many shops. However, the concepts of the integrated CASE tools and the software workbench are a refinement of the islands of automation that exist to increase the productivity of the information worker.

Top management and Information Systems managers must budget the resources for systems departments to continue to upgrade and invest in tools for their staff. They must also continue to educate their management and user staff on the use of Information Technology. The challenge of 1990's are multi-faceted involving integrated networks, voice-data-video transmissions, high-level code management, multi-vendor software and hardware platforms, object-oriented programming and holistic databases. The evolution of the I.S. department to new areas of processing require the information systems department to document large complex systems. Nowadays their are many levels and players in an organization's information processing: the individual processor, the work-group, the organization processing, inter-organization processing (Electronic Data Interchange(EDI)), the manual user procedures and the continued support of all the old and existing systems nobody has the resources nor understanding to replace nor upgrade!!! When the resources are available to implement a new organizational information system, the ability to comprehend and document the current state of affairs is crucial to designing and implementing a replacement software system. CASE tools and the related concepts provide a standardized consistent methodology for accomplishing the challenging task of software replacement in the 1990's. There are several excellent books by noted authors that will provide a great source of information for your staff, purchase these books for department and encourage your staff to read and actively discuss the concepts presented in these books. As an Information System manager you should be attending seminars and classes to further you own knowledge and understanding of the on-going Information Technology evolution.

Glossary of Terms

4GL - Fourth Generation Language - Program development environment. Typically a database system with a set of software development tools such as menu generators, screen builders, report writers, compilers, linkers and debuggers.

Back End CASE Tools - CASE tools which provide automation in programming, implementation and maintenance portion of the software development life cycle.

CASE - Computer Aided Software Engineering

Context Diagram - the top level diagram of a multi-level data flow diagram.

Data Dictionary - a collection of all the data elements used in a software. Each data element in the dictionary is described as to its inclusion in data flows, files, sources, sinks and attributes.

Data Flow Diagram - a graphic representation used to show the interfaces and functions of the components in a system; used to determine the sources and sinks of data. Graphical representation of the data flows between components.

Entity-Relation Diagram - shows only the entity types and their relationships.

Front End CASE Tools - CASE tools which provide automation in the design and analysis portion of the software development life cycle.

Pseudo Code - high level, machine independent program logic specifications.

Structured English - a subset of the English language with restricted syntax and vocabulary, used for process specifications. Used in conjunction with logical constructs of structured programming to create program pseudo code.

Structured Analysis - a disciplined step-by-step approach for performing system analysis and producing a system specification which conforms to a specific set of rules and principles. The major methodologies are: Yourdon structured design, Gane-Sarson structured analysis, DeMarco structured analysis, Orr structured design, Jackson structured design.

Bibliography

Yourdon, Edward Modern Structured Analysis. Englewood Cliffs, N.J.: Yourdon Press 1989.

Martin, James Database Design. Englewood Cliffs, N.J.: Prentice-Hall 1985.

McClure, Carma CASE is Software Automation. Englewood Cliffs, N.J.: Prentice-Hall 1989.

Kronke, David Management Information Systems. Santa Cruz, CA : Mitchell Publishing, Inc 1989.

Visible Systems Corporation Visible Analyst. 950 Winter Street, Waltham, MA 02154

Cognos Corporation PowerDesign, PowerCASE, PowerHouse. 67 South Bedford, Burlington, MA 01803-5164

Hewlett-Packard Company HP SoftBench. 19310 Pruneridge Ave, Cupertino, CA 95014

Informix Software Inc. Informix. 4100 Bohannon Drive, Menlo Park, CA 94025

Infocentre Corporation Speedware. 2300 East Katella Ave #150, Anaheim, CA 92806

Project Name : Integrated Software Implementation for Charles McMurray
Project Number : ISI001.MST
Original Date : October 5, 1990
Revision Date :

I. Project Background

Charles McMurray started its computerization in 1976 with the purchase of an NCR computer system running inventory control software. This software was developed and enhanced over a period of time by several individuals. In 1983, Charles McMurray, purchased a Hewlett-Packard 3000(HP-3000) series 39 computer system and converted all the existing NCR software to the HP-3000.

Recognizing the limitations of the existing application software, Charles McMurray has started on a project for evaluating, purchasing, modifying and installing an integrated application software system.

II. Project Definition

This project plan will define the steps for acquisition, modification, installation and conversion to an integrated application software for Charles McMurray.

The steps are as follows:

2.1 Define the modifications required for the new [redacted] system, so that existing functions will also be available in the new system.

2.2 Define the data migration strategy from the existing software to the new integration application software system.

2.3 Define the installation and training criteria for the new [redacted] system.

III. Project Objectives.

3.1 Identify all existing functions that are performed at Charles McMurray for Order Processing, Inventory Management, Accounts Receivable, Accounts Payable, Purchasing and General Ledger.

3.2 Identify the modifications to the new application that will be required to continue performing the functions defined in step 3.1.

3.3 Establish a concrete plan for the testing, acceptance and control of the implementation for the new [redacted] system.

IV. Approach.

4.1 Analyze the current system , file structure and data elements.

4.2 Define the files and data elements to be converted to the new [redacted] system either manually and/or by a conversion program. See attachment 'B' for the data element conversion cross-reference table.

4.3 Define the testing steps needed to validate the fundamentals of the [redacted] system with the functional requirements. See attachment 'C' for the functional test definitions. The test scripts will provide for a structured training format. If during the training/testing the system fails to meet the requirement of the user then an Information System Service Request will be completed with the specific structured english describing the required function.

4.4 Define the framework and steps needed for implementing the modified application in a production environment. See attachment 'A' for details.

V. Project Scope.

5.1 Define the division of labor for project tasks with the Development Team for the implementation of the [redacted] function requirements in a production environment.

5.2 Users will perform the tests and quality assurance phase of the project as defined in attachment 'A' and in Section VI Project Resource Requirements.

VI. Project Resources Requirements.

6.1 [redacted] Development Team:

6.2 Training & Documentation: Garry L. Smith

6.3 Documentation and standards provide by the software vendor.

6.4 Software Resources: [redacted] - CASE tool.
Personal Computer productivity tools.
[redacted] Application Software.

6.5 Hardware Resources: HP-3000 Series 922RX.
Nine(9) Track 1600 BPI tape Drive.
Personal Computer System.

VII. Project Completion Definition.

This project will be considered finished upon the successful completion of the criteria noted below.

7.1 Quality assurance results have been formally approved by the development team.

VIII. Estimated Completion Date.

8.1 See Attachment 'A' , 'B' and 'C'

8.2 The estimated completion date is June 1, 1991.

IX. Start Date.

Project is started upon formal approval.

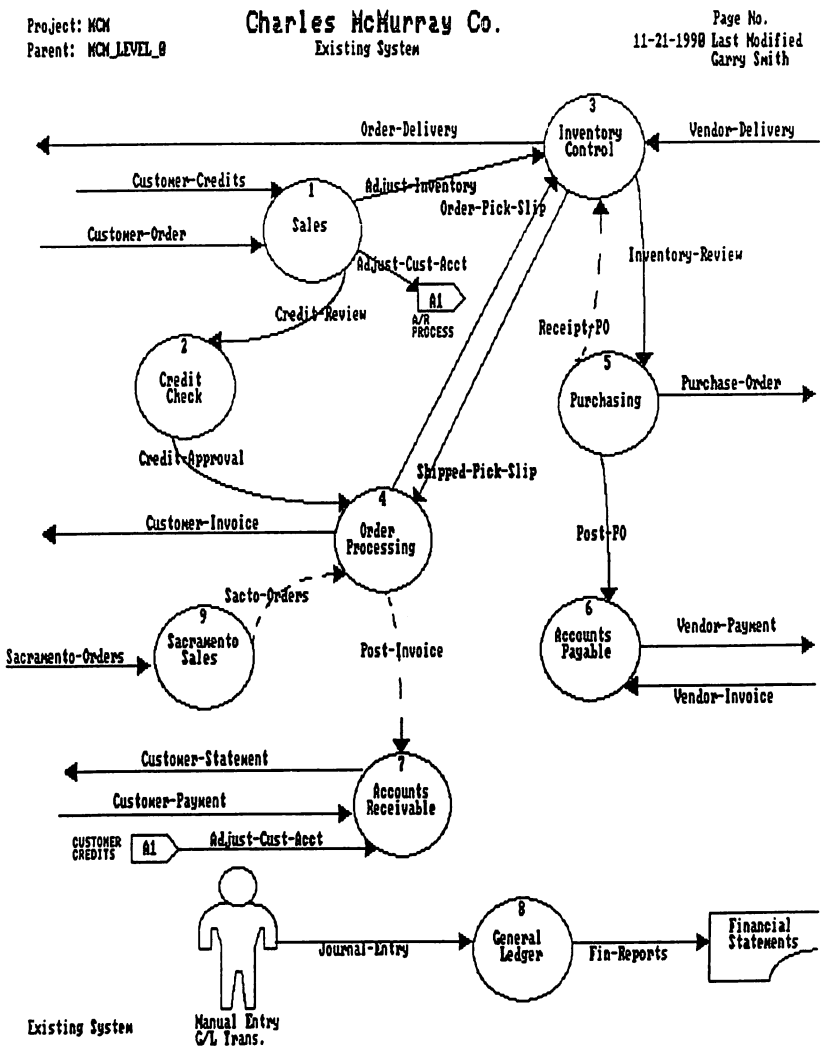
Approved. Date: 10/10/90.

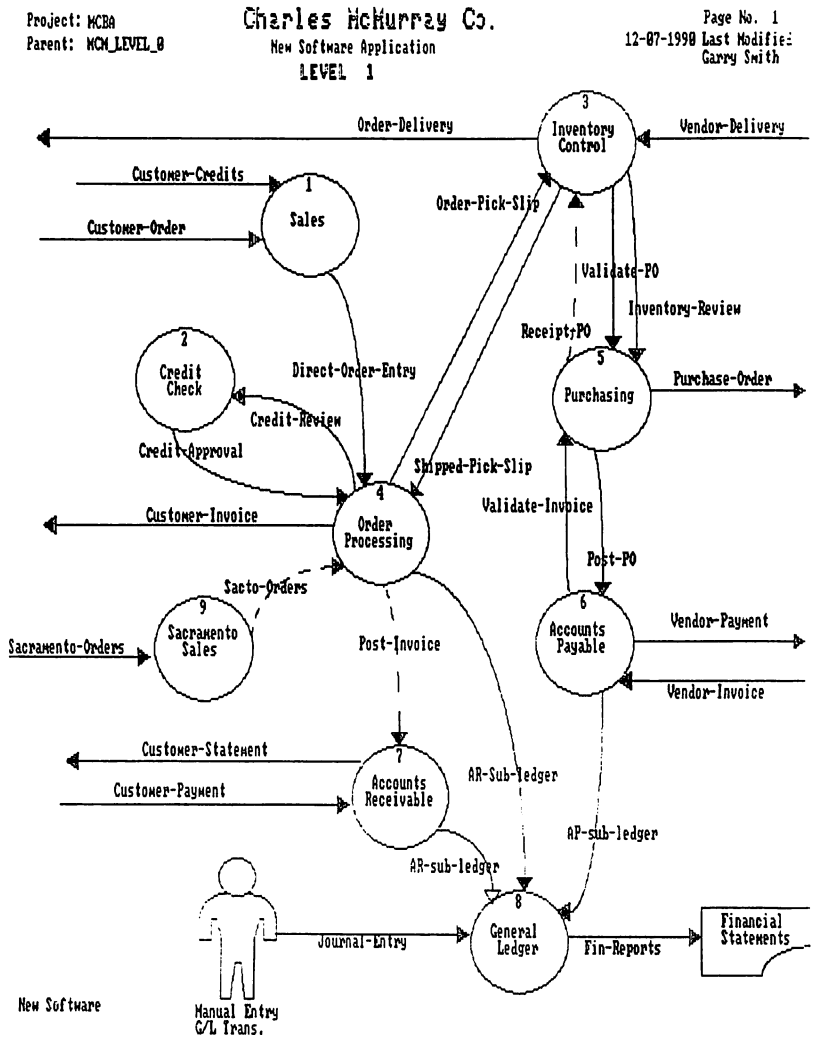
[redacted]
President, Charles McMurray Company.

Revised: 12/20/90

[redacted] Implementation and Conversion Plan

No.	Task Description	By	Plan	Actual
1	Analyze existing procedure and system requirements.	GS	10/10 - 11/20/90	12/03/90
	1.1 Accounts Receivable	MS/KT	10/10/90	
	1.2 Accounts Payable	MC	10/16/90	
	1.3 General Ledger	MC	10/23/90	
	1.4 Inventory Management	DC	10/26/90	
	1.5 Purchasing	DC	11/05/90	
	1.6 Order Entry	DC/CS BY/KT	11/15/90	
2.	Review the [redacted] data elements and define the conversion procedures needed to convert the existing KSAM files.	GS/DC	11/20/90	12/03/90
3.	Identify KSAM data elements that need to be added to the [redacted] databases or modification required to the conversion procedures.	GS	11/30/90	12/03/90
	3.1 Convert the KSAM file structures to the [redacted] system for use in testing.	GS	12/15/90	12/17/90
4.	Begin modifications to [redacted] source code as specified in step 1 process.	GS	01/05/91 - 03/25/91	12/17/90
	4.1 Identify any additional changes to the conversion procedures and/or changes to the [redacted] System.	GS	03/25/91	
5.	Begin implementing the General Ledger system in the production environment.	GS	03/25/91	
	5.1 General Ledger Training and Setup	GS/MC	03/18/91	


Project: MCM
Parent: MCM_LEVEL_0
Charles McMurray Co.
Existing System
Page No.
11-21-1990 Last Modified
Garry Smith
3
Inventory Control
Order-Delivery
Vendor-Delivery
Customer-Credits
1
Sales
Adjust-Inventory
Order-Pick-Slip
Customer-Order
Adjust-Cust-Acct
A1
A/R PROCESS
Inventory-Review
Credit-Review
Receipt/PO
2
Credit Check
5
Purchasing
Purchase-Order
Credit-Approval
Shipped-Pick-Slip
4
Order Processing
Customer-Invoice
Post-PO
Sacto-Orders
9
Sacramento Sales
Sacramento-Orders
Post-Invoice
6
Accounts Payable
Vendor-Payment
Vendor-Invoice
Customer-Statement
7
Accounts Receivable
Customer-Payment
CUSTOMER CREDITS
A1
Adjust-Cust-Acct
Journal-Entry
8
General Ledger
Fin-Reports
Financial Statements
Existing System
Manual Entry G/L Trans.


Project: MCBA
Parent: MCM_LEVEL_0
Charles McMurray Co.
New Software Application
LEVEL 1
Page No. 1
12-07-1990 Last Modified
Garry Smith
3
Inventory Control
Order-Delivery
Vendor-Delivery
Customer-Credits
1
Sales
Customer-Order
Order-Pick-Slip
Validate-PO
Inventory-Review
Receipt-PO
2
Credit Check
Direct-Order-Entry
5
Purchasing
Purchase-Order
Credit-Review
Credit-Approval
Shipped-Pick-Slip
Validate-Invoice
Customer-Invoice
4
Order Processing
Post-PO
Sacto-Orders
9
Sacramento Sales
Sacramento-Orders
Post-Invoice
6
Accounts Payable
Vendor-Payment
Vendor-Invoice
Customer-Statement
7
Accounts Receivable
AR-Sub-ledger
Customer-Payment
AP-sub-ledger
AR-sub-ledger
8
General Ledger
Financial Statements
Journal-Entry
Fin-Reports
New Software
Manual Entry G/L Trans.

Appendix D

Date: 05-09-1991 Project: MCM Page:
Time: 12:06:49
Single Entry Listing

EMU DATA ELEMENT

Description:

Estimated Montly Usage is based on historical weighting of quantities sold in the previous 12 months.

Location:

DATA FLOW --> Item-SOQ

Date: 05-09-1991 Project: MCM Page: 1
Time: 12:12:05
Single Entry Listing

Item-SOQ DATA FLOW

Description:

Create purchase orders by vendor based on the EMU formula. See Misc. EMU label description.

Composition:

safety, on-order, EMU

Notes:

This requires a 24 period history for the EMU calculation. 12-current, 12-prior year.

---

qty-shipped DATA ELEMENT

Description:

Quantity Shipped on this order.

Values & Meanings:

Num(7.2)

Location:

DATA FLOW --> Confirm-Order

Paper Number: 3235

The Evolution of Relational Technology

Howard Rosenfield
400 Oracle Parkway
Redwood Shores, CA 94065
415/506-6161

Introduction

The old axiom that "knowledge is power" applies more than ever in today's highly competitive markets. The ability to access the right data quickly and in useful formats is critical to success. Companies continue to invest millions of dollars in technology to ensure that their various organizations can access up-to-date information and beat the competition.

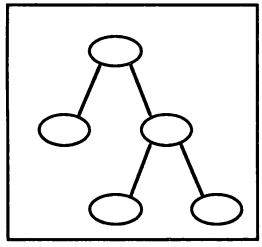
The increasing need for data management tools, like Database Management Systems (DBMS), clearly illustrates how dependent businesses of all types and sizes are on timely and accurate information. DBMS technology is constantly evolving to meet their increasing demands, and has moved beyond the hierarchical and network based implementations of the past. Today's relational implementation of DBMS technology provides data managers with an easy-to-use, flexible, timely and powerful way to provide the needed information to run their businesses.

This paper traces the evolution of relational DBMS technology from its inception in the 1960's to the present. A discussion of what makes a DBMS relational and how it works are included. Advances in hardware and software that have contributed to the development of the relational databases as they exist today will also be covered. Key current and future trends will also be touched upon to highlight what the technology can, and will do for the data processing professional.

Technology Trends Leading Up To Relational DBMSs

Early implementations of DBMSs made full use of the hardware that they were designed to run on. These systems have limited functionality and flexibility by today's standards. The relational DBMS evolved partly out of earlier data modeling schemas, and has gone far beyond the limitations of its predecessors.

Hierarchical A hierarchical DBMS can be thought of as set of parent-child relationships that form a tree structured database. Each node has only one parent, but can have many children. When a query for information is put to the hierarchical DBMS the basic operation is a tree search. Each node is traced from the root node through the 'tree' until the conditions of the query are met. The nodes of the database hierarchy (or hierarchical tree) consist of records connected to each other via links.



When the first hierarchical DBMSs (IBM's DL/1 and IMS) were introduced in the 1960's, they managed data in a manner that was previously unknown. Data administration was being performed with flat files and COBOL. The introduction of the first commercially viable hierarchical DBMS, IBM's IMS, came in 1968. This pioneering

DBMS represented a significant step in the evolution of data management. The applications developed using hierarchical databases were efficient and fast for situations where the type of information that the user required was static. However, a great deal of real world information was not well suited for the hierarchical implementation. In these systems, data is contained in strictly nested hierarchies. Each node has only one parent. Data had to be represented several times. This was a waste of data storage space and made the database difficult to update or correct. Hierarchical databases limited how you could modify data structures, and how the data was logically represented. Therefore, the type and amount of information that could be extracted from a given body of data was restricted.

Network The network model of DBMS design, such as Hewlett-Packard's IMAGE & Cullinet's IDBMS, built upon the earlier hierarchical model. While the hierarchical model permits only one parent to each child node (one-to-many relationships between data), the network implementation was designed to let each node have several pointers that point to many other nodes (one-to-many & many-to-one relationships). These pointers or data links are clearly defined in the DBTG (Data Base Task Group) data definition language (DDL).

This new concept of a DDL allowed the network DBMS architecture to move beyond its hierarchical cousin. The data structure was no longer severely restricted. You could now define many different types of data links to get at the data. Still, the network model is a limited tool for data management. For instance, the data links are often extremely complex and since any node can be linked with any other (nested data links) it is often difficult to figure out how to modify the database. Even when it is known what the data access paths look like, it is often difficult to change them.

In a network DBMS, there is no notion of the ad hoc query. This restricts the type of information that can be extracted from the database. Even though this method of data modeling is more flexible than its hierarchical predecessor, it is limited by the complexity of its data structure and remains inflexible.

The Relational Model

In the late 1960's Dr. E.F. Codd, who was working for IBM at the time, first conceptualized the relational data model. The first working implementation of the relational model was developed at IBM San Jose in the mid-70's, and was known as System/R. The first commercially viable implementation of a relational DBMS was delivered by Oracle in the late 1970's.

SQL

Manipulation of data with relational databases is done using SQL (pronounced "*sequel*"). SQL stands for Structured Query Language. It was developed for the relational data model in the mid 1970's by IBM (C.J. Date). It has since been adopted by ANSI (American National Standards Institute) as the standard language for relational databases.

SQL has three distinct components:

- DML (Data Manipulation Language)
- DDL (Data Definition Language)
- DCL (Data Control Language)

The relational model represents a simple yet very robust way of viewing data. It is relatively easy to maintain, and data can be represented in many different ways. In addition to the One-to-Many and Many-to-One relationships that could be performed with network architected databases, the relational model permits Many-to-Many data relationships. Relational data modeling allows for greater flexibility in data analysis than its predecessors. This has led to the development of powerful 4GL tools that are used by application developers and end users to access the database.

In relational databases, all data is conceptually stored in tables. A relational database consists of a set of tables. The tables consist of columns (also referred to as fields) and rows of data records. Each row of a table is a representation of a set of attributes defined by the columns. Tables are joined on their mutual columns. Every table has a column known as a *primary key* to uniquely identify each row. Sometimes a table requires a combination of two columns to form a primary key. A *foreign key* is a column in a table that is not used to uniquely identify each row of the table, but is a primary key in another table. For instance, given the two tables EMPLOYEE and DEPARTMENT we can point out the primary and foreign keys:

DEPARTMENT

Department Number	Department Name	Location
10	Sales	San Diego
20	Operations	San Francisco
30	Accounting	New York

EMPLOYEE

Employee Number	Employee Name	Job	Manager	Department Number
8941	Jones	Salesman	8941	10
6578	Smith	Clerk	7465	30
3276	Roberts	Analyst	9287	20
7642	Johnson	Manager	9933	10

Primary Keys **Foreign Key**

If we needed to join the two tables, we would do so with the department number column since it is the common column between the two tables.

Relational data modeling does not rely on nested pointers to join each of its nodes or entities together. The difficulties of maintaining and modifying a database that exist in the network data model are no longer an issue because these nested data links are not used. The inability to accurately reflect real world data that exists with hierarchical

---

DML handles functions such as retrieving, updating, inserting and deleting rows. The commands SELECT, UPDATE, INSERT and DELETE are used to manipulate tables, with SELECT being the most widely used. The structure of a SELECT statement is as follows:

SELECT [DISTINCT] *item-list*
FROM *tables*
[WHERE *search-conditions*]
[GROUP BY *columns*]
[HAVING *expression*]
[ORDER BY *columns*]

Items are selected from rows in a table or tables. Specific search conditions can be described by using a WHERE statement. If the search conditions are met, the resulting data can be organized by GROUP BY, HAVING, and/or ORDER BY statements. An example using the

databases is also greatly reduced with the relational model. Data structures are no longer inflexible. Instead, there is logical and physical data independence. Data independence is defined by C.J. Date (a developer of System/R and a relational DBMS authority) as, "the immunity of applications to change in storage structure and access strategy." In other words, each table has its own theme, and two or more tables can be connected or *joined* as long as they share a common attribute. This can be done without effecting an application. This strategy eliminates the data redundancy that occurs in hierarchical databases. You can extract information from the database simply by doing something like the following from the SQL command line:

```
SELECT *      (* indicates a wildcard)
FROM DEPARTMENT;
```

This would retrieve all information contained in the DEPARTMENT table.

The concept of a view is very important to relational databases. Views can be thought of as *virtual* tables since they are not physically stored in the database and are not permanent. They are extremely useful to the user for one-time extraction of information from the database without having to alter the data structure. They also maintain data independence and enhance security by insulating users from knowing in which tables the data they' re working with is actually stored.

Data Dictionary

The basic outline of a relational DBMS requires that there be a data dictionary (also known as a system catalog) to store information about all of the objects (table names, indexes, views, etc.). The data dictionary is comprised of tables, just like any other part of a relational database. Anyone can query the data dictionary to find out the names of tables, column names of a particular table, or what tables share a common column name. The data dictionary is also used by the DBMS itself to provide security. For example, all information pertaining to the GRANT commands issued are contained in the data dictionary, and the DBMS uses this information to check whether or not a particular user has access to a particular database object. Distributed databases (to be covered later) are another area where the data dictionary is used by the DBMS. Information about which remote computer contains which parts of the database is contained in data dictionary tables.

EMPLOYEE and DEPARTMENT tables could be:

```
SELECT *      (* indicates a wildcard)
FROM EMPLOYEE
WHERE DEPARTMENT NUMBER = 10
ORDER BY EMPLOYEE NUMBER;
```

This would retrieve all information from the EMPLOYEE table where the department number is equal to 10. The data would be ordered by employee number, with Johnson (#7642) being listed first and Jones (#8941) being listed next.

DML also includes the transaction management commands COMMIT and ROLLBACK. The COMMIT statement instructs the database to physically write any changes that have been made. The ROLLBACK command can be used by database administrators to literally roll back any updates that have been made to the database since the last commit. The ROLLBACK

Why Relational?

Initially, relational DBMS technology was not seen as a viable solution to the problems of data management. In the early 1980's relational database technology was described as too slow for businesses in their pursuit to manage and use their data efficiently. It was considered to be strictly an academic endeavor with little chance of becoming a practical commercial tool.

In fact, this was not so much a limitation of the relational design but a limitation of the hardware that existed at the time. As a result, the relational approach initially appeared less robust than what was available at the time. Opinions have changed with advances in hardware and software technology.

The hardware configurations of the late 1970's and early 1980's did not have the high-powered CPUs that exist today. Consequently, the software of the time was designed to be I/O intensive, and not CPU reliant. This is the case with hierarchical and network DBMSs. To extract information from the database, rigidly defined data links are used. This requires relatively little CPU processing since the data links explicitly define what data access paths should be used.

The relational data model was designed for flexibility and ease-of-use. It was also designed for flexible and efficient search algorithms, which have the effect of limiting the amount of I/O done by the system. These advanced search algorithms are transparent to the end user. The goal in relational design is to minimize the number of access paths to a particular piece of information. A minimal number of data blocks are examined, which results in less disk I/O. This method of data retrieval does place higher demands on the CPU because the DBMS is required to do a lot of the 'thinking' that would be taken care of by predefined data links in hierarchical and network DBMS implementations.

Reliance upon CPU power was a painful fact of life for relational DBMS users in the early 1980's. The CPUs of the time had limited horsepower by today's standards. However, the increasing speed of today's processors has exceeded the advances made in disk I/O. Hardware designs no longer impede the performance of relational DBMSs In fact, transaction rates required for OLTP (On-Line Transaction Processing) environments are satisfied by relational DBMSs Relational databases are now achieving transaction rates in the hundreds of TPS (Transactions Per Second) range. In a few years, thousands of TPS will be the norm.

---

command maintains the integrity of the databases in cases where transactions are not completed for some reason, such as power or communication failures.

SQL Data Definition Language (DDL) is used to create tables, indexes and views. The creation of a table would have the following format:

```
CREATE TABLE employee
(Employee_Number num,
Employee_Name char(15),
Employee_Address char(40),
Department char(5))
```

SQL's Data Control Language (DCL) is used for database security purposes. The two main commands are GRANT and REVOKE. With these commands, database administrators can give users the authority to SELECT, UPDATE, INSERT and DELETE from tables or views.

All of this is not to say that the relational DBMS was mature software when it was first introduced in the late 1970's. Deficiencies in the relational model prevented it from becoming commercially viable. The majority of these 'missing' features are included in today's relational model, which is now an efficient and reliable tool for managing data.

Row level locking is a recent improvement. Until recently, when a user wanted to extract or read data from a particular table the database engine locked the entire table to other users. This limited the usability and speed of applications. Row-level locking means that users can proceed normally while someone is updating a particular row of a table in the database. Because the DBMS only locks the row that is being written to, other users can access the rest of the rows in the table. Improved SQL optimizers, advances in distributed database technology (described below), and portability of relational technology across dozens of hardware platforms are additional enhancements made to relational DBMS implementations.

Client-Server

Until recently, all database processing was done on centralized minicomputers or mainframes accessed by dumb terminals. These centralized processors often became overloaded as more users were added and applications became more complex. Personal Computers (PCs) began to be used widely as centralized processing was becoming overloaded. They provided low-cost, user-friendly alternatives to centralized computing. Networking technology began to evolve as users sought ways to link their PCs together so that resources could be shared. If additional processing power was required on a Local Area Network (LAN) another PC could be purchased and added to the network at minimal cost. However, despite all of these advantages, PC LANs still did not give the user a complete solution.

Many of the following features inherent in centralized computing could not be provided by PCs:

- *Data Integrity*—The PC environment could not protect against the loss or corruption of data if system failure occurred.
- *Security*—It was difficult to restrict access to unauthorized users.
- *Availability*—Hardware vendors have invested heavily to assure that their mini's and mainframes are running at all times of the day and night. PCs do not offer this high level of availability.
- *Centralized Database Administration/Centralized Processing*—Having one group responsible for data management and system performance helps to maintain a more stable data processing environment.
- *Performance*—The PC does not have the CPU throughput processing required for most data management.

Client-Server computing is a relatively recent phenomena. It combines the best of the centralized computing world with the PC LAN environment by dividing an application up into two parts. The database server's tasks are handled by the mini or mainframe with their high availability and performance. Data integrity and security are enforced here, and here is where the database resides. The smaller client machine is usually

a PC, Macintosh, or UNIX workstation, which handles all processing required by the application and communicates with the database server via the network. SQL statements are passed from the client to the server. The requested information is then sent back over the network from the database server to the client machine running the front-end user interface tools.

Multiple clients can concurrently access the same database server. Should the demands of the clients surpass the capacity of the database server, extra resources can be added in two ways (known as *scalability*):

- *Vertical Scaling*—The server can be replaced with a larger machine with more capacity (disk, CPU, memory, etc.).

- *Horizontal Scaling*—Additional servers can be added to the configuration, thus spreading the demands of client machines over several servers.

Client-server technology is beginning to become widely used by data processing professionals with tools like Microsoft Windows becoming available. It gives them a way to optimize their large investments in mini/mainframes and PC LANs by combining them into heterogeneous computing environments that take the best of both worlds without the limitations of either.

Distributed Databases

A distributed DBMS (DDBMS) is a database spread out over multiple computers. These computers are usually located at different physical sites, and are connected by a communications network. The concept of a local database, with its own database administrator, still exists with DDBMSs However, there is also the concept of a global database made up of autonomous, local databases spread out over many physical nodes. The global database appears as one logical database to the user or application developer. All global operations conducted by the DBMS are transparent to the user. The relational data model supports this concept because data is located by value and not via pointers or physical position, as is the case with hierarchical and network DBMSs

The DDBMS allows a more accurate representation of how information exists within an enterprise. Since companies typically have many geographically dispersed computers, it is logical to assume that data is also organized this way. A global database draws from multiple, physically dispersed databases. This is of great advantage to the user. For instance, let's say you were the chairman of Ford Motor Company and you wanted to design a new ad campaign for the Escort. Before you sat down with your advertising people, perhaps you felt it would be good to review some demographic information on the people that bought Escorts over the past year. With a DDBMS, you could issue a SQL query that would retrieve the needed data from various databases residing in San Francisco, New York, Paris, Rome etc. How this information is retrieved and delivered to your screen would be transparent to you, and would be handled by the DDBMS.

A distributed DBMS allows the user to request information from a database without having to worry about how it is being retrieved. The data integrity on the various nodes of a DDBMS is maintained, and each node is controlled locally. The data on multiple databases can also be easily accessed or modified.

LOCATION TRANSPARENCY of data is accomplished by a data dictionary that maintains and coordinates distributed transactions. The locations of all tables, rows, indexes, etc. are found in the data dictionary which is referenced by the DBMS when a distributed transaction is issued. The user issuing a query against a remote database does not need to know where the data is located since the database engine performs the search.

When a table is modified that is referenced by another table on a different node, both the referencing and referenced table need to maintain the same value in both tables for a common column. This concept is known as *referential integrity.* It can be defined as the enforcement of relationships among data. It is based on the concepts of master/detail relationships between tables and primary (PK) and foreign keys (FK). This can be illustrated with the DEPARTMENT and EMPLOYEE tables below. If a new employee, Clark, is added to the EMPLOYEE table with a department number of 15 an error will be returned. The detail table (EMPLOYEE) will reference the master table (DEPARTMENT) and find that 15 is not a valid department number.

Referenced Table (master table)

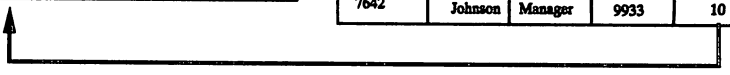
DEPARTMENT

Department Number	Department Name	Location
10	Sales	San Diego
20	Operations	San Francisco
30	Accounting	New York

Referencing table (dependent table)

EMPLOYEE

Employee Number	Employee Name	Job	Manager	Department Number
8941	Jones	Salesman	8941	10
6578	Smith	Clerk	7465	30
3276	Roberts	Analyst	9287	20
7642	Johnson	Manager	9933	10



Consistency of data across database nodes is enforced with a mechanism known as *two-phase commit.* In a perfect world with no system failures, there would be no need for such a mechanism. However, a system can breakdown during a distributed COMMIT transaction (save to the database) which can create an inconsistent state in the database. To maintain data integrity, the transaction is done in two phases:

- *Prepare Phase*—The node that issues the COMMIT (parent node) asks each of its dependent nodes (child nodes) to notify it when they are prepared to commit their part of the transaction. They are told not to take any action, until instructed to by the parent node.

- *Commit Phase*—The parent node commits, unless there has been a failure during the Prepare Phase. *[If a failure occurs, the parent node rolls back—does not execute—the transaction.]* The parent node then instructs each child node to commit or roll back the transaction. Each child node then informs the parent node that it has committed or rolled back the transaction.

Relational DBMS will lead the way to the widespread use of distributed databases. They will profoundly effect how data management is handled. For example, users will be able to tap global databases and draw inferences from data residing on separate databases located all over the country or the world. This will be possible without the user having to know about where the data is located, or how it is extracted. DBMSs

with *open architecture* will make distributed databases even more invaluable to the data manager. A DBMS with an open architecture provides tools that permit access to heterogeneous DBMSs (i.e., relational to network, relational to hierarchical). Any MIS manager that has to deal with more than one DBMS to manage data will find an open architected DBMS essential. It protects the considerable investment made in one DBMS architecture and still allows work to proceed in another. Many relational database vendors already offer heterogeneous DBMS tools.

Future DBMS Trends

The amount of information that is currently accessible is growing at an exponential rate. Take online text databases as an example. By the late 1980's there were over two billion documents stored in databases. In 1979, there were 400 online text databases worldwide. By 1988, the number was in excess of 3000. This is only an approximation of the public online databases. There are many proprietary databases that are not accessible to the general public. There are also millions of databases containing information on almost anything imaginable (such as sales, inventory, financial information, and market research).

This information is only useful if the user can retrieve what is needed. The database engine must have highly sophisticated data access paths. Just as relational DBMSs handle more database searching than their hierarchical and network predecessors, future DBMS technology will be even more intelligent when it comes to retrieving information. The user's ability to process and locate information may not improve dramatically even though the amount of online data is increasing at a steady rate. The DBMS *must* retrieve data more efficiently. It is a safe assumption, given its simple yet powerful design, that the relational data model will be able to accommodate the next stage of DBMS evolution without forcing its users to make drastic changes in how they manage their data. Users will be able to use their existing applications, and take advantage of the new technology as it becomes available.

One way of making a relational database more intelligent is to use a more sophisticated indexing process where the index is based on concepts or objects rather than individual keywords. This approach is often referred to as *object-oriented* data modeling. An object can be thought of as anything that can be defined as a noun or a noun phrase. This approach represents data in a format which is very close to the user's perception of real world data. A person, a concept like DNA, a picture, or an architect's design can all be thought of as objects. An object is defined by its attributes. For a car, the attributes might include model, year, color, and horsepower. The concept of the *class* is important to object-oriented databases. Every object is a member of specific category or class. A class may also be defined as a *subclass* of one or more other classes. An example would be the Chevrolet Corvette, which is a member of the American sports car class, which could be a subclass of the sports car class, which could be a subclass of the automobile class, and so on. How the classes and subclasses interrelate is a matter of database design.

The relational model will be able to handle the evolution to object-oriented design without traumatizing data managers. Every row in a table can be thought of as containing a database object. The object can be identified by the tables primary key, and the class of the object can be thought of as the table in which the row resides. By enhancing the current relational data model to include features of the object-oriented model the evolution of the DBMS will move towards an object-oriented approach where the relational model survives as a subset of the overall object-oriented model.

New DBMS architectures take a few years to mature to the point where they can be used in production environments. Because the relational model will evolve from its current state to a more object-oriented model, the data manager will not have to start from scratch in order to take advantage of the latest DBMS technology. Applications will be enhanced gradually by taking advantage of appropriate features as they are developed. There will also be no need to decide whether an application will run against a relational database or the newer object-oriented database. It is possible that an application will span relational and object-oriented designs. A single query could return results by accessing both tables and objects which will coexist in the same database.

This step in the evolution of DBMS design will be transparent to the end user. However, the enhanced power and flexibility of future database designs will permit the development of highly advanced user interface tools that will make information highly accessible to those who need it. This will result in more efficient use of information and greater productivity.

Conclusion

Relational database technology provides today's data managers with an easy-to-use, flexible, and powerful way to access the information required to run their businesses. While it was once considered to be strictly an academic endeavor, it has evolved beyond its hierarchical and network predecessors to become the state of the art in database management systems. By taking advantage of today's powerful CPUs relational technology has been able to provide the speed once reserved only for hierarchical and network databases and the usability that was previously lacking in those earlier DBMS designs. Relational database performance will continue to improve by taking advantage of future advances in CPU power.

Client-server computing and distributed databases based on relational technology will help users and application developers find new and innovative ways to access and utilize their information. Object-oriented databases will eventually evolve out of the relational model. Data will be conceptualized in a different way. This will allow businesses to use the information at their disposal to its fullest potential. The relational data model will be a big part of this evolution, so data managers will not have to make drastic choices about which DBMS technology to use. Relational database technology is the tool that will allow businesses to manage their complex data needs today and in the future.

HP Motif XL: The X Window System on MPE XL
Paper Number 3236

Scott Cressler

Hewlett-Packard
19447 Pruneridge Avenue
Cupertino, CA 95014
(408)447-5548

HP Motif XL is the implementation of the X Window System, the Xt Toolkit and the OSF/Motif widget library for MPE XL. This paper will discuss the components of an application which uses HP Motif XL to present a graphical user interface from an HP 3000 Series 900.

X OVERVIEW

The X Window System is an industry standard application programming interface. Developed and maintained by the Massachussets Institute of Technology (MIT), X can be used by an application developer to present a graphical user interface. This interface can be displayed on a high-resolution display connected to the machine on which the application is executing or on another machine on the network. This distributed nature is achieved through the client/server architecture of the X Window System. The application acts as a client, requesting user interface services from a process called the X display server. This process manages the high-resolution display, the keyboard and the mouse of the machine acting as the X display. It also informs the client application when the user provides input.

Applications which use X to present their user interface enjoy two types of portability. The interface part of the application can easily be ported, through recompilation and relinking, to any system which supports the X libraries. Additionally, the protocol used to communicate from the client to the display server is standardized so a client can display on any X display server, regardless of hardware or operating system.

The ability to present a windowed, graphical user interface over a network is provided through a set of programmatic interfaces called the X library (Xlib). This library is very flexible, imposing few constraints on the X developer. However, this flexibility also causes program development to be quite tedious. Fortunately, another library out of MIT, the Xt Toolkit, provides a set of programmatic interfaces which significantly ease the creation of an X user interface. Xt supports the creation and use of widgets. "Widget" is the name given to a set of functions and data structures which, when used by an application with the Xt interfaces, will display and manage user interaction with a piece of a user interface. For example, a "push button widget" would display a button and then inform the application when the user has "pushed" the button with the mouse. A library of these widgets can be used to create an entire X user interface without having to use Xlib functions directly.

One such library of widgets, which is gaining much acceptance in the industry, is from the

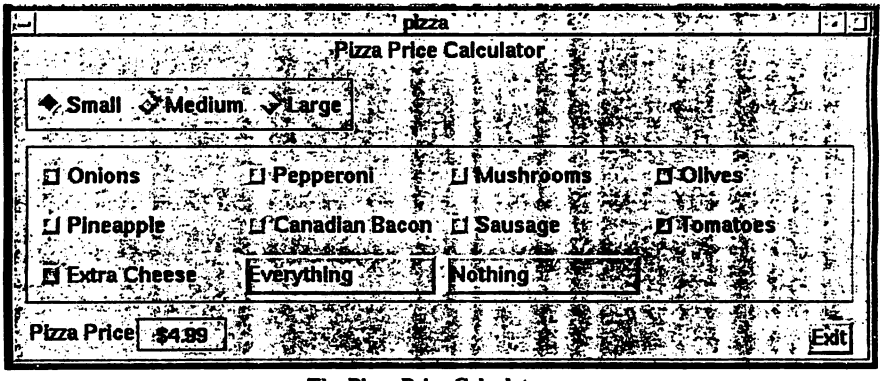
Open Software Foundation (OSF) and is called OSF/Motif. With the OSF/Motif widgets an application can present an X user interface which has a three-dimensional appearance and behavior. This user interface would also be consistent in appearance and behavior with Microsoft's OS/2 Presentation Manager (PM) (and Windows 3.0), reducing the training time needed for a user accustomed to PM to learn an OSF/Motif application. The ease of a user who is familiar with one type of user interface to use other programs with a similiar user interface is called "user portability." This is a portability benefit enjoyed by OSF/Motif applications which is not as easy to achieve using X alone.

HP Motif XL is the implementation of these three programmatic interface libraries on MPE XL. It consists of the Xlib, Xt and OSF/Motif libraries and the header (or include) files necessary to allow an OSF/Motif application to be compiled, linked and run on an HP 3000 Series 900. This application, written in the C language, will then be an X client application and can display on an X display server connected to the same LAN. An HP 9000 workstation is an example of an X display server.

THE APPLICATION

This paper will center around a small HP Motif XL application. The application is a Pizza Price Calculator, which is called PIZZA. Through a set of buttons, the user chooses the size of the pizza and the toppings. As each topping is chosen, the new price of the pizza is displayed. There are also buttons for choosing all toppings, for choosing no toppings and for exiting the application. A complete listing of the C language source code of the application can be found at the end of this paper.

This is the appearance of the user interface of the PIZZA application:



The Pizza Price Calculator

Although this application is fairly trivial, it is a good example to demonstrate the use of X and OSF/Motif. It can also be seen in more general terms as an application to determine pricing of a product being ordered.

TERMINOLOGY

This section contains the definition of several terms which will be used in the paper. You might scan it now and use it for reference later.

Application Class Name -- This name is used to group applications into similar classes. Any resources (see below) which are defined for a given application class name will apply to any application which specifies that name during initialization.

Events -- The display server informs the application of input in its window, the need to redraw its window because it was overlapped and is now exposed, or other interesting occurences by sending the application "events."

Gadgets -- A simplified form of a widget. Although gadgets lack some of the functionality of widgets, such as the ability to have their colors individually customized by the user, they are faster than widgets and use less memory. For this reason, gadgets should be used wherever possible.

Geometry -- The geometry of a widget is its size and placement in the window.

Resources -- A resource is a customizable attribute of an application and its user interface, e.g., the foreground color of the application window or the font used in displaying text. Resource values can be specified by the developer, system manager or user through configuration files, called resource files. They can also be specified on the command line when the program is executed.

Widgets -- A widget is a set of functions and data structures which can be used and reused by an application to display and manage user interaction with a piece of a user interface.

MAIN APPLICATION BODY

The main() function of a C program is the one which is executed when the program is run. X applications all have fairly similar main() functions. This section will "step through" the major components of the main body of the PIZZA application. The following is the source of the main() function:

```
main(argc, argv)
int argc;
char    **argv;
{
    Arg al[1];
    int       ac;
    XtAppContext app;

    initializeInfo();

    topLevel = XtAppInitialize(&app,    /* (default) application context */
                               "Pizza", /* application class name */
                               NULL,    /* no options */
                               0,       /* number of options */
                               &argc,   /* used to get standard command line */
                               argv,    /* options and application name */
                               NULL,    /* no fallback resources */
                               al,      /* no args */
                               0);      /* no args */

    /* Tell Shell to resize if its children (specifically geometry
       managing children like Forms) ask it to. */
    ac = 0;
    XtSetArg(al[ac], XmNallowShellResize, True);
    ac++;
    XtSetValues(topLevel, al, ac);

    createInterfaceWidgets();

    XtRealizeWidget(topLevel);

    XtAppMainLoop(app);
}
```

Declarations

The declarations of al and ac are peculiar to an application which uses widgets through Xt. "Al" stands for "argument list" and "ac" for "argument count". An argument list is a structure which is used to specify resource values when creating a widget or modify the resource values

of an existing widget. This topic is discussed more in the section "Allow Shell Resizing" below.

Initialization

After the declaration portions, the following statements are the first statements of the program:

```
initializeInfo();

topLevel = XtAppInitialize(&app,    /* (default) application context */
                           "Pizza", /* application class name */
                           NULL,    /* no options */
                           0,       /* number of options */
                           &argc,   /* used to get standard command line */
                           argv,    /* options and application name */
                           NULL,    /* no fallback resources */
                           al,      /* no args */
                           0);      /* no args */
```

The first function called, initializeInfo(), is a local function to the PIZZA application which simply initializes some arrays of information about the prices and names of the pizza toppings. In a real application this function would probably access a database to obtain this information.

The next function, however, is a call to the Xt function XtAppInitialize(). Every application which uses Xt must call this function before using any other Xt interfaces. XtAppInitialize() performs several important tasks. It reads all the user and system resource files to create a database of resource values for use by the application and its widgets. This function also makes the connection with the display server process.

XtAppInitialize() also searches the command-line arguments to the program for some standard parameters, such as the display server name and the background and foreground colors of the application.

Finally, XtAppInitialize() creates a shell widget. It is the first widget created and handles any communication with the window manager. This widget will also be the "parent" of all other widgets in the application. It has a widget ID which is functionally returned and saved in the topLevel variable.

The first parameter of XtAppInitialize() is the address of an application context structure which will be initialized by this function. The application context is a structure maintained by Xt which contains information about the application. Most applications, including PIZZA, do not directly manipulate the application context, but it must be passed to other Xt routines.

The second parameter identifies the class name of the application. This name is used when determining which resources found in resource files apply to this application. Multiple, related applications can share an application class name. The program name of the application

(obtained by XtAppInitialize from argv) is also used for finding resources specific to this particular application, as opposed to all applications in the same class. Resources are too large a feature of X applications to cover in this paper. For an excellent discussion of resources and application classes, see the "X Toolkit Intrinsics Programming Manual" in the bibliography.

The next two parameters of the function are used when the application wishes to define some of its own command-line arguments to be handled by XtAppInitialize(). The PIZZA application does not use this feature, so these parameters have been initialized to NULL and 0.

Argv and argc are passed in the next two parameters to XtAppInitialize(). Argv and argc are standard parameters in the main function of any C language program. On MPE XL, the INFO string of a program is parsed (by C startup code), and the resulting array of string arguments is passed to the program as argv. Argc is a count of the number of these arguments. When running a program using Xt, some standard parameters, such as the display server and the foreground and background colors, may be specified in the INFO string. This information is used by Xt and is obtained by passing argv to XtAppInitialize().

As discussed earlier argv is parsed by XtAppInitialize(). Any standard command-line options found in argv, such as the display name, are removed from the argv array.

The NULL passed in the next parameter indicates that PIZZA will not be using the fallback resources feature of XtAppInitialize().

The final two parameters are used if the caller wishes to customize the shell widget which is created by this function.

Allowing Shell Resizing

The next section of the main() function is the following:

```
ac = 0;
XtSetArg(al[ac], XmNallowShellResize, True);
ac++;
XtSetValues(topLevel, al, ac);
```

XtSetValues() is an Xt function which can be used to set resource values after the widget has been created. As discussed earlier, al is an argument list which consists of an array of resource name and resource value pairs. The function XtSetArg() is used to add an entry to the list. In this call to XtSetArg(), an entry is being added to the argument list to specify that the resource referred to by the XmNallowShellResize constant is to be set to true. Setting this resource to true allows children widgets of the Shell widget to request geometry changes, including resizing and repositioning. This is necessary because the PIZZA application requires some of its widgets to resize during the execution of the program. For example, the Label gadget in which the calculated price is displayed changes size to fit the length of the price.

Create And Manage Widgets

In main(), the function createInterfaceWidgets() is called next. This function creates and manages all the widgets in the user interface of the application. Creation of a widget consists of allocating and initializing the data structures in the application to control the appearance and behavior of each widget.

Widgets which are created specifying a widget as their parent are called children of that widget. A parent widget usually controls the layout, that is, the location and size, of its children. The only widgets which have children are widgets whose purpose is to control layout. The Form and RowColumn widgets of OSF/Motif are such widgets and will be discussed in the "Geometry Managers" section.

After creating a widget its parent is informed that it must manage this child, which consists of controlling its geometry.

This createInterfaceWidgets() function is discussed in detail in the sections "Creation Of Widgets" and "Callback Functions".

Realize The Widgets

Creating the widgets does not involve communication with the display server and so does not result in anything visible on the display. This process of creating windows on the display and making them visible is called realizing the widgets. The function XtRealizeWidget() performs this task for the widget passed to it and all descendants of that widget. For this reason, XtRealizeWidget() is called with the widget ID of the topLevel widget which is the parent of all the widgets in the user interface.

Process Events

The main part of any X program is a loop which waits for the next event from the server and then processes it. In an Xt application, all events are processed by the Xt code. Application work is done through callback functions, which are discussed in the "Callback Functions" section. The loop which processes events is, therefore, contained in the function XtAppMainLoop(), which is called at the end of the main() function.

An Example Creation

The following code from the createInterfaceWidgets() function creates and manages the PushButtonGadget which will be used to exit the application. This code illustrates the process of creating and managing a widget:

```
ac = 0;
xmStringPtr =
    XmStringCreateLtoR("Exit", XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac], XmNlabelString, xmStringPtr);
ac++;
XtSetArg(al[ac], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET);
ac++;
XtSetArg(al[ac], XmNtopWidget, pricePushButtonGadget);
ac++;
XtSetArg(al[ac], XmNrightAttachment, XmATTACH_FORM);
ac++;
XtSetArg(al[ac], XmNrightOffset, 10);
ac++;
exitPushButtonGadget =
    XmCreatePushButtonGadget(form, "exitPushButtonGadget", al, ac);
XtManageChild(exitPushButtonGadget);
XmStringFree(xmStringPtr);
XtAddCallback(exitPushButtonGadget, XmNactivateCallback,
   exitCallback, (XtPointer)0);
```

This code performs the following tasks which are common to the creation of most widgets:

- Specify the values of several resources in the argument list to control the creation of the widget.
- Create the widget.
- Inform the parent of the widget that it is to manage this child widget.
- Register a callback function for the widget.

The call to XmStringCreateLtoR() is an OSF/Motif function to create a structure called a compound string. Compound strings allow control over the direction of the display and the character set used when displaying a string. The function call used here specifies that the string "Exit" is to be displayed from left to right using the default character set. A pointer to the structure which has been created is returned and passed to XtSetArg() to specify the string to use as the label of the push button.

The Form widget is the parent of the exitPushButtonGadget gadget. The other four calls to

XtSetArg() set up resources used by the Form. The Form widget uses these resources to determine how to control the geometry of the exitPushButtonGadget gadget. The Form widget is discussed in more detail in the "Geometry Managers" section.

The call to XmCreatePushButtonGadget() creates the data structures needed by Xt to implement this widget. The first parameter indicates that the Form widget whose ID is stored in the form variable is to be the parent of the new push button gadget. The next parameter is the resource name of the widget, which can be used when specifying resources specific to this widget. The last two parameters are the argument list and the count of the number of arguments which are used to influence the creation of the widget.

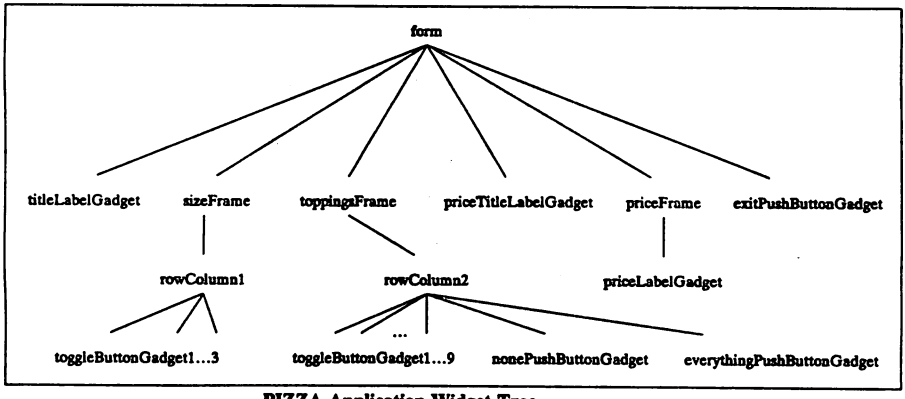
Next the parent of the widget is informed that it is to manage the size and placement of this new widget through a call to XtManageChild().

The call to XmStringFree() simply frees the memory allocated by XmCreateStringLtoR().

The final function call adds a callback function for this widget. Callback functions are discussed in the "Callback Functions" section.

Widget Hierarchy

The widgets which are created by createInterfaceWidgets() form a tree structure of parents and children. The following figure illustrates this widget tree:



PIZZA Application Widget Tree

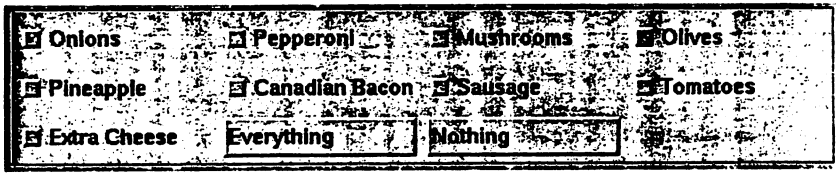
The Form widget is used to manage the placement of the other widgets. Frame widgets simply create a visible frame which resizes to enclose its children. The first frame, sizeFrame, has one child which is a RowColumn widget. RowColumn widgets are used to organize their children

into rows and columns. This widget is used to align its three ToggleButtonGadget children in a horizontal row. ToggleButtonGadgets are buttons which represent a state, either they are set or unset (pushed down or popped up). They usually provide the user a choice. These toggle buttons are used to choose the size of the pizza, small, medium or large.

Size Buttons Appearance

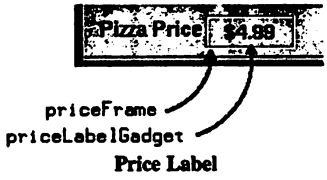
The diamond shape used for these toggle buttons is specified using the XmNindicatorType resource during creation. This shape indicates to the user that the buttons will follow "radio-button" behavior, that is, only one of the buttons can be set at a time.

The next frame, toppingsFrame, also has a RowColumn as its child. This RowColumn has several ToggleButtonGadgets as its children which it organizes into three rows. These toggle buttons are used to specify what toppings the pizza is to have (the square shape of the toggle buttons indicates to the user that more than one button can be set at one time). There are also two PushButtonGadgets which are used to specify that the pizza is to have "everything" (all toppings) or no toppings.



Topping Buttons Appearance

The last frame, priceFrame, contains a LabelGadget. A LabelGadget simply displays a string. This is where the price of the pizza is displayed.



Price Label

The exitPushButtonGadget was discussed above. A PushButtonGadget presents a button which appears to be pressed when the mouse is clicked on it and then rebounds to its unpressed appearance. They usually represent an action the user can perform, such as exiting the program.

Exit Button

CALLBACK FUNCTIONS

Since all user input, through events, is processed by the Xt and widget code, how does the application get a chance to react to user activity? The answer is a mechanism called callback functions. A callback is a function which has a predefined interface and which is called when certain events or combinations of events have occurred. For example, when a PushButtonGadget is pressed, several events are generated. These events inform the client application that the mouse button was pressed, that it was pressed in the PushButtonGadget, and that it was released while still in the PushButtonGadget. Functions registered as Activate callbacks for this PushButtonGadget will only be called if all these events are generated. For instance, if the mouse button is pressed in the PushButtonGadget but released outside of the gadget, the Activate callbacks are not called.

The documentation for a widget itemizes the callbacks which make sense for a given widget and under what conditions each callback will be called.

A widget maintains a list of the functions which are to be called when a given callback condition occurs. The application adds its function to that list by calling the Xt function XtAddCallback().

Here is an example Activate callback for the Exit push button:

```
static void
exitCallback(widget, client_data, call_data)
Widget widget;
caddr_t client_data;
caddr_t call_data;
{
        exit();
}
```

When this function is called, the first parameter, widget, will contain the ID of the widget which called it. The second parameter is called client_data because it can be used by the application (remember, X applications are called clients) to pass information to the callback.

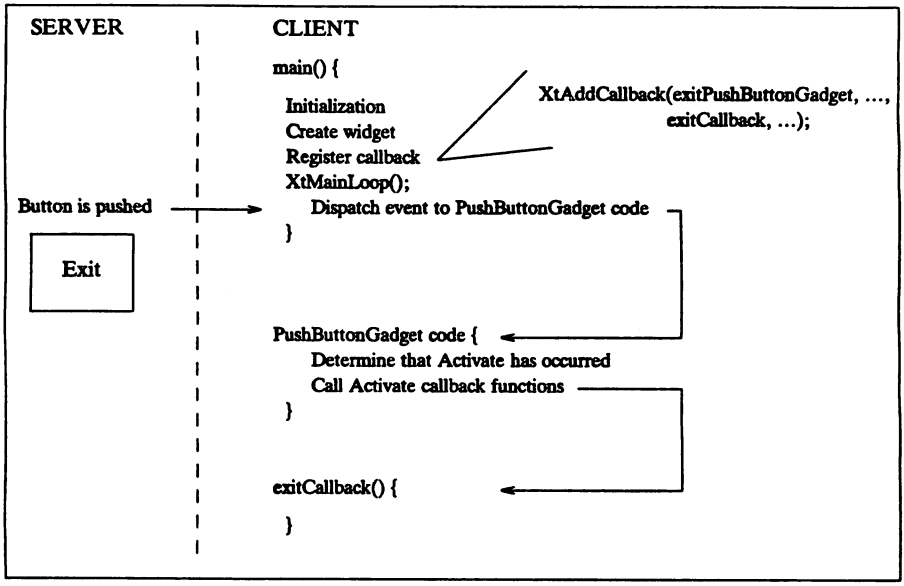
This is most useful when the same callback function is called by several widgets. The client_data field can easily be used to identify the reason the callback was called. Client_data will be discussed further below. The third parameter, call_data, is used by some widgets (as opposed to the application) to pass information to the callback.

This is the code which adds this callback function to the Activate callback function list for exitPushButtonGadget:

```
exitPushButtonGadget =
  XmCreatePushButtonGadget(form, "exitPushButtonGadget", al, ac);
XtManageChild(exitPushButtonGadget);
XmStringFree(xmStringPtr);
XtAddCallback(exitPushButtonGadget, XmNactivateCallback, exitCallback,
              (XtPointer)0);
```

The widget must be created before a callback can be added. XmNactivateCallback is an OSF/Motif constant which specifies the type of callback to the widget. The last parameter is not used in the case of the exitPushButtonGadget. This parameter can be used by an application to specify client data. The value of this parameter will be passed in the client_data parameter when the widget code calls the callback.

Here is a graphic representation of the use of a callback function to implement the Exit button:



Exit Callback

A good example of the use of client_data is the callback triggered when the "pizza size" toggle buttons are selected:

```
static void
sizeButtonValueChangedCallback(widget, sizeIndex, call_data)
Widget widget;
caddr_t sizeIndex;
caddr_t call_data;
{
        chosenSize = sizeIndex;

        calcPrintPrice();
}
```

And some code fragments from the creation of the toggle buttons:

```
for (i=0; i < NUM_SIZES; i++) {

    ac = 0;
    if (i == chosenSize) {
       /* This is the default size, must start the button as pressed. */
       XtSetArg(al[ac], XmNset, True);
       ac++;
    }
    xmStringPtr = XmStringCreateLtoR(sizeArray[i].sizeName,
                                     XmSTRING_DEFAULT_CHARSET);
    XtSetArg(al[ac], XmNlabelString, xmStringPtr);
    ac++;
    XtSetArg(al[ac], XmNindicatorType, XmONE_OF_MANY);
    ac++;
    sizeArray[i].widgetID = XmCreateToggleButtonGadget(rowColumn1,
                            sizeArray[i].sizeName, al, ac);
    XtManageChild(sizeArray[i].widgetID);
    XmStringFree(xmStringPtr);
    XtAddCallback(sizeArray[i].widgetID, XmNvalueChangedCallback,
                  sizeButtonValueChangedCallback, (XtPointer)i);

}
```

The above "for" loop creates three toggle buttons. The widget IDs are saved in an array. As each gadget is created, a callback is added to its valueChanged callback list. The valueChanged callbacks are called when the toggle button is pressed by the user, changing its state from set to unset or vice versa. The index in the sizeArray of widget information is passed as the fourth parameter of the XtAddCallback() call. This is the client data parameter and will be passed in the second parameter when the callback is called. This parameter of the sizeButtonValueChangedCallback() function, sizeIndex, is assigned to the chosenSize global variable. The chosenSize variable is used when calculating the price of the pizza to index into sizeArray to get the base price of a pizza of the chosen size.

The effect is that when a size toggle button is pressed, the widget calls the sizeButtonValueChangedCallback() function, chosenSize is updated, and calcNPrintPrice() is called. This latter function uses chosenSize, the sizeArray and the information about the chosen toppings to calculate the current price of the pizza and display it in the priceLabelGadget.

GEOMETRY MANAGERS

As mentioned earlier, some widgets exist simply to constrain the layout of other widgets. The layout of a widget consists of its size and location in the window. Two of these "constraint widgets" are used in the PIZZA application: the Form and the RowColumn.

Forms

The purpose of an OSF/Motif Form widget is to manage the layout of its children. First the Form is created and then each widget is created, specifying the widget ID of the Form as its parent. When a widget is the child of a Form, some new resources can be specified for that widget. They define to the Form what kind of positioning on the Form the widget should have. Attachments for the left, right, top and bottom sides of the widget can be specified.

Here is an example of the use of these constraints:

```
ac = 0;
xmStringPtr =
    XmStringCreateLtoR("Pizza Price", XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac], XmNlabelString, xmStringPtr);
ac++;
XtSetArg(al[ac], XmNbottomAttachment, XmATTACH_FORM);
ac++;
XtSetArg(al[ac], XmNbottomOffset, 10);
ac++;
XtSetArg(al[ac], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET);
ac++;
XtSetArg(al[ac], XmNleftWidget, toppingsFrame);
ac++;
XtSetArg(al[ac], XmNtopAttachment, XmATTACH_WIDGET);
ac++;
XtSetArg(al[ac], XmNtopOffset, 10);
ac++;
XtSetArg(al[ac], XmNtopWidget, toppingsFrame);
ac++;
priceTitleLabelGadget =
    XmCreateLabelGadget(form, "priceTitleLabelGadget", al, ac);
        .
```

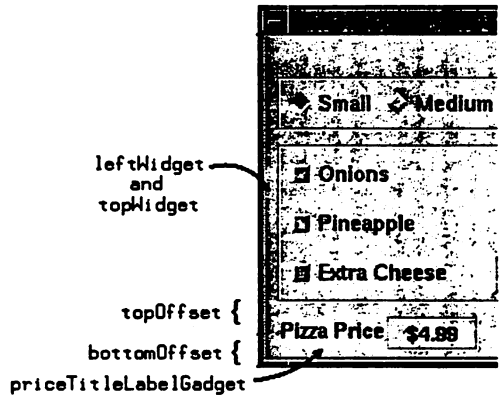
This is a code fragment from the creation of the LabelGadget which is displayed next to the calculated price. It displays the string "Pizza Price". The values given to the XmNbottomAttachment and XmNbottomOffset resources define that the bottom of the LabelGadget will be attached to the bottom of the Form with an offset of ten pixels. If the Form is resized, the LabelGadget will stay ten pixels from the bottom of the Form.

The next two resources specify that the left side of the LabelGadget is to be attached to the left side of the toppingsFrame widget. Notice the use of ATTACH_OPPOSITE_WIDGET rather

than ATTACH_WIDGET. ATTACH_WIDGET would attach the left side of the LabelGadget to the right side of the frame. Since the goal is to align the left sides of these widgets vertically, ATTACH_OPPOSITE_WIDGET must be used.

Finally, the next three resource value assignments will cause the top of the LabelGadget to be attached to toppingsFrame with an offset of ten pixels.

Here is what the LabelGadget looks like when positioned:



priceTitleLabelGadget Form Positioning

This practice of attaching widgets to each other is much more flexible than specifying the X and Y coordinates. It is also easier because determining the X and Y coordinates in pixels can be tedious.

RowColumn

The main purpose of the OSF/Motif RowColumn widget is to arrange its child widgets in rows and columns. Using a RowColumn widget removes the responsibility of determining exact positions from the application developer. The RowColumn will also automatically adjust positions of its children when new child widgets are added.

When the number of widgets which must be organized into rows and columns is unknown at the time of writing the application, a RowColumn is indispensible. In the PIZZA application, the toppings widgets are currently "hard-coded" into the initialization function. However, the application is written to create the topping toggle buttons in a loop, and they are all created as children of a RowColumn. This means the application could easily be changed to read the toppings from a file and create the number of toggle buttons necessary. They would all be arranged nicely in the RowColumn, regardless of how many were created.

An additional feature of the RowColumn widget is its ability to manage behavior as well as

layout. The following code fragment is from the creation of the RowColumn containing the toggle buttons which are used to choose the size of the pizza:

```
XtSetArg(al[ac], XmNradioBehavior, True);
ac++;
rowColumn1 = XmCreateRowColumn(sizeFrame, "rowColumn1", al, ac);
```

Only one of the size toggle buttons should be set at a time, since only one size of pizza can be selected at any time. By setting the XmNradioBehavior to true for the RowColumn, the RowColumn itself will prevent more than one of its child toggle buttons from being set at a time. When one of the toggle buttons is pressed, the one which was currently pressed is reset. Although this feature could be implemented by the application developer, it is a great boost to productivity to have the RowColumn provide it. To give the use a visual queue that these buttons are radio buttons, the XmNindicatorType resource of each toggle button is set to XmONE_OF_MANY when they are created. This gives them the diamond shape which is defined for radio buttons by the OSF/Motif Style Guide.

BUILDING THE APPLICATION

The focus of this paper so far has been a discussion of the use of OSF/Motif and the Xt Toolkit. These discussions are general to any platform supporting these two application programming interfaces (APIs). This section, however, will discuss topics specific to HP Motif XL, the implementation of these APIs on MPE XL.

Product Structure

The HP Motif XL product will be located in an MPE XL account called HPX11. All header (include) files, including those for X, Xt and Xm (OSF/Motif), will be included in the group H.HPX11. The relocatable libraries (RLs) with which an HP Motif XL application must be linked are located in LIB.HPX11.

Compiling

So how is the PIZZA application compiled? Assuming that the file PIZZAC contains the source, the following command would perform the compile:

```
:ccxl pizzac,ypizza,lpizza;info="-IH.HPX11"
```

This command will compile the source in PIZZAC into an object file called YPIZZA and produce a listing file called LPIZZA.

The INFO= string in this command is very important, but before discussing it you should understand how the C Compiler on MPE XL (CCXL) handles include statements. The

following is a typical set of include statements in an OSF/Motif application and, in fact, are used in the PIZZA application:

```
#include <Xm/Xm.h>
#include <X11/Shell.h>
#include <Xm/Form.h>
#include <Xm/RowColumn.h>
#include <X11/IntrinsicP.h>
```

For those versed in the use of X and OSF/Motif on a UNIX platform, the hierarchical organization of header files is familiar. These statements in a C program specify that the files named Xm.h, Form.h and RowColumn.h are to be included from the Xm directory located in the system default header directory (/usr/include on most UNIX platforms). Notice that the case of the file names is significant, e.g., Form.h is different from form.h.

All of these include statements would have to be changed when compiling on MPE XL except for a feature of CCXL. When the CCXL preprocessor encounters something of the form:

```
#include <Xm/Xm.h>
```

It strips the prefix and postfix and searches for a file named XM in the H group of the SYS account. However, the include files for HP Motif XL are located in H.HPX11. To resolve this problem, the "-I" option passed in the INFO= string allows the developer to override the standard search path for header files. Given the above CCXL command, when the preprocessor encounters the above include statement, it will search for the file XM in the group H.HPX11 before searching in H.SYS.

Without these two features of CCXL (the preprocessor's stripping of prefixes and postfixes and the "-I" option) all includes in an X application would have to be different on MPE XL from other platforms.

Linking

After an object file is produced by a successful compile, the program file can be generated by linking the object file. The following command would link the PIZZA application:

```
:link from=ypizza;to=pizza;rl=libxm.lib.hpx11,libxt.lib.hpx11,&
      libx11.lib.hpx11,libc.lib.sys,libcrand.lib.sys,libcansi.lib.sys
```

The FROM parameter specifies the name of the object file, and the TO parameter specifies the name of the program file being generated. The RL parameter specifies a list of relocatable libraries (RLs) to be linked to the object file. The code needed by the object file will be

included from the RLs in the program file. Due to dependencies in the relocatable libraries, they must be specified in the order shown above.

Compatibility And Portability

Because HP Motif XL is a port of industry standard API libraries, its use has significant implications for the portability of an application. HP Motif XL is based on X Version 11, Release 4 and OSF/Motif 1.1. An application written for those APIs will port easily to MPE XL. An additional benefit for those shops supporting multiple platforms, the user interface code of an application written with HP Motif XL will port easily to other platforms supporting X and OSF/Motif.

There are several areas which must be addressed when either porting code to HP Motif XL or writing code which is to be portable to another platform:

1. Include file names
2. Resource file names
3. XmFileSelectionBox behavior

The previously mentioned features of CCXL prevent the need for changes in most include statements found in an OSF/Motif application. However, the file naming limitations on MPE XL prevent CCXL from handling includes where the name of the file is longer than eight characters.

The following include statements are from the PIZZA application:

```
#include <Xm/RowColumn.h>
#include <X11/IntrinsicP.h>
```

In this example, both RowColumn and IntrinsicP are longer than eight characters. For this reason, the names of some of the header files are different on MPE XL from other systems and the include statements must be changed. The following example is a good way to modify the source so it will compile both on an MPE XL system and on other platforms:

```
#ifndef mpexl
#include <Xm/RowColumn.h>
#include <X11/IntrinsicP.h>
#else  /* mpexl */
#include <Xm/RwColumn.h>
#include <X11/IntrnscP.h>
#endif /* mpexl */
```

The standard define "mpexl" is defined by CCXL. When this source is compiled on MPE XL,

the files RWCOLUMN.H.HPX11 and INTRNSCP.H.HPX11 will be included.

The correspondence between the standard names of the header files and their names on MPE XL will be documented with the HP Motif XL product.

Related to this discussion of header files, the following set of instructions must be included in the source before including any HP Motif XL header files:

```
#ifdef mpexl
#include <mpexl.h>
#endif
```

This code will cause (when compiled for MPE XL) the header file MPEXL.H.HPX11 to be included. MPEXL.H.HPX11 contains some definitions needed by the HP Motif XL header files.

Another difference between HP Motif XL and X on other systems is also related to names. Because of the limitations on file names imposed by the MPE XL file system, the names and locations of the resource files used by X applications are different. For descriptions of the use of these files, please see the "X Toolkit Intrinsics Programming Manual" listed in the bibliography. The following is a summary of the differences:

- Application Class Default files are found in the group APPDEFLTS.HPX11 on MPE XL, not in the directory /usr/lib/X11/app-defaults
- Application class names must follow MPE XL file naming rules if an Application Class Defaults file is to be used
- The user resource file is called XDEFAULT on MPE XL, as opposed to .Xdefaults on UNIX
- The Xdefaults-host User Environment Specific Resource file is not supported on MPE XL, only the use of the XENVIRONMENT variable

Another HP Motif XL portability concern is related to the OSF/Motif FileSelectionBox widget. This is a sophisticated widget which is used when an application requires the user to choose a file. The widget presents a window with a scrollable list of files from which the user may choose. It is very specific to the UNIX-style hierarchical file system. In porting this widget to MPE XL, its behavior was changed to reflect the structure of the MPE XL file system. The function interface has not changed. However, an application which is being ported to MPE XL from a UNIX platform and expects file names of the form found on UNIX may need changes. This is because once the user has chosen a file and the appropriate callback is called, the application can get the file name string of the chosen file. On MPE XL this will be (if fully qualified) of the form FILE.GROUP.ACCOUNT, whereas on UNIX this might have been of the form /dir1/dir2/dir3/dir4/dir5/file. If the application processes the file name at all, it will probably have to be changed for MPE XL. However, if it simply passes the file name string to something like the C library open() function, it may continue to operate correctly without change on MPE XL.

CONCLUSIONS

The release of HP Motif XL on MPE XL will provide new options for HP 3000 Value Added Business (VAB) partners. It can be used to port the user interface of applications written for other platforms to MPE XL, increasing the number and type of applications available for HP 3000 customers. It can be used to create user interface code on the HP 3000 which is portable in a multiple platform shop.

HP Motif XL can even be used to add a bit-mapped graphic portion to an otherwise forms oriented application. Imagine a VPLUS application which is used for analyzing the data in a database. If this application were being used from a window on an X display, at the touch of a function key the application could open an X Window and display a pie chart of the data.

WHERE TO GO FROM HERE

There are many good books available today discussing X, Xt and OSF/Motif programming. HP also offers excellent classes in C programming and X and OSF/Motif programming.

BIBLIOGRAPHY

Hewlett-Packard, "HP OSF/Motif Programmer's Reference", Hewlett-Packard Company, 1989.

Nye, Adrian, "Xlib Programming Manual, Version 11", O'Reilly & Associates, Inc., 1990.

O'Reilly, Tim, "X Toolkit Intrinsics Reference Manual for X Version 11", O'Reilly & Associates, Inc., 1990.

Nye, Adrian and O'Reilly, Tim, "X Toolkit Intrinsics Programming Manual for X Version 11", O'Reilly & Associates, Inc., 1990.

Young, Douglas A., "X Window Systems Programming and Applications with Xt", Prentice Hall, 1989.

APPENDIX A: PIZZAC

```
/*****************************************
**    A Simple HP Motif XL Example     **
**                                     **
**    Author: Scott Cressler           **
**    Date  : 1/8/91                   **
**                                     **
*****************************************/

#ifdef mpexl
#include <mpexl.h>
#endif

#include <stdio.h>
#include <string.h>
#include <Xm/Xm.h>
#include <X11/Shell.h>
#include <Xm/DialogS.h>
#include <Xm/Form.h>
#include <Xm/Frame.h>
#include <Xm/LabelG.h>

#ifndef mpexl
#include <Xm/RowColumn.h>
#else  /* mpexl */
#include <Xm/RwColumn.h>
#endif /* mpexl */

#include <Xm/ToggleBG.h>

#ifndef mpexl
#include <X11/IntrinsicP.h>
#else  /* mpexl */
#include <X11/IntrnscP.h>
#endif /* mpexl */

#include <Xm/DialogS.h>

#ifndef mpexl
#include <Protocols.h> /* will be in /usr/include/X11 or .../X11/Xt */
#else  /* mpexl */
#include <Prtocols.h>
#endif /* mpexl */

/* Constant definitions for toppings. */
```

```
#define NUM_TOPPINGS 9
#define ONIONS              0
#define PEPPERONI    1
#define MUSHROOMS    2
#define OLIVES              3
#define PINEAPPLE    4
#define CANADIAN_BACON      5
#define SAUSAGE             6
#define TOMATOES     7
#define EXTRA_CHEESE 8
#define EVERYTHING   1000
#define NOTHING         1001

struct  topping {
        Widget widgetID;
        char   *toppingName;
        int    chosen;
        float  price;
      };

struct topping toppingArray[NUM_TOPPINGS];

#define NUM_SIZES       3

struct  pizzaSize {
        Widget widgetID;
        char   *sizeName;
        float  price;
      };

struct pizzaSize sizeArray[NUM_SIZES];

int   chosenSize = 0;

float toppingsPrice = 0.0;

/* Global variables used by main and other routines */

XtAppContext app_context;

Widget   topLevel;

/* Widgets defined/created by this application, global so they can be
   accessed by multiple creation functions. */

static Widget form;
static Widget titleLabelGadget;
static Widget sizeFrame;
```

```
static Widget rowColumn1;
static Widget toppingsFrame;
static Widget rowColumn2;
static Widget everythingPushButtonGadget;
static Widget nothingPushButtonGadget;
static Widget priceTitleLabelGadget;
static Widget rowColumn3;
static Widget priceFrame;
static Widget priceLabelGadget;
static Widget exitPushButtonGadget;

/* Forward declaration */

void createInterfaceWidgets();
void initializeInfo();


/*
 * main routine.
*/
main(argc, argv)
int argc;
char    **argv;
{
    Arg          al[1];
    int               ac;
    XtAppContext app;

    initializeInfo();

    topLevel = XtAppInitialize(&app,    /* (default) application context */
                               "Pizza", /* application class name */
                               NULL,    /* no options */
                               0,       /* number of options */
                               &argc,   /* used to get standard command line */
                               argv,    /* options and application name */
                               NULL,    /* no fallback resources */
                               al,      /* no args */
                               0);      /* no args */

    /* Tell Shell to resize if its children (specifically geometry
       managing children like Forms) ask it to. */
    ac = 0;
    XtSetArg(al[ac], XmNallowShellResize, True);
    ac++;
    XtSetValues(topLevel, al, ac);

    createInterfaceWidgets();
```

```
    XtRealizeWidget(topLevel);

    XtAppMainLoop(app);
}



/* Function to initialize the array of topping choices and prices. */
void
initializeInfo()
{
        int i;

        for (i=0; i < NUM_TOPPINGS; i++) {
          toppingArray[i].chosen = False;
          toppingArray[i].price  = 0.0;
        }
        toppingArray[0].price       = 0.12;
        toppingArray[1].price       = 0.52;
        toppingArray[2].price       = 0.23;
        toppingArray[3].price       = 0.15;
        toppingArray[4].price       = 0.27;
        toppingArray[5].price       = 0.56;
        toppingArray[6].price       = 0.47;
        toppingArray[7].price       = 0.13;
        toppingArray[8].price       = 0.25;
        toppingArray[0].toppingName = "Onions";
        toppingArray[1].toppingName = "Pepperoni";
        toppingArray[2].toppingName = "Mushrooms";
        toppingArray[3].toppingName = "Olives";
        toppingArray[4].toppingName = "Pineapple";
        toppingArray[5].toppingName = "Canadian Bacon";
        toppingArray[6].toppingName = "Sausage";
        toppingArray[7].toppingName = "Tomatoes";
        toppingArray[8].toppingName = "Extra Cheese";

        /* Until values are read from a file... */
        sizeArray[0].sizeName = "Small";
        sizeArray[1].sizeName = "Medium";
        sizeArray[2].sizeName = "Large";
        sizeArray[0].price = 4.99;
        sizeArray[1].price = 7.99;
        sizeArray[2].price = 9.99;

}

static void
calcNPrintPrice()
```

```
{
        Arg al[11];
        int    ac;
        XmString xmStringPtr;
         float   pizzaPrice;
         static char    priceString[10];
         int     i;

         pizzaPrice = sizeArray[chosenSize].price;

         for (i = 0; i < NUM_TOPPINGS; i++)
             if (toppingArray[i].chosen)
                pizzaPrice = pizzaPrice + toppingArray[i].price;

         ac = 0;
         sprintf(priceString, "$%.2f", pizzaPrice);
         xmStringPtr = XmStringCreateLtoR(priceString,
                         XmSTRING_DEFAULT_CHARSET);
        XtSetArg(al[ac], XmNlabelString, xmStringPtr);
        ac++;
         XtSetValues(priceLabelGadget, al, ac);

}

/* Callback Procedures */
static void
toppingButtonValueChangedCallback(widget, toppingIndex, call_data)
Widget widget;
caddr_t toppingIndex;
caddr_t call_data;
{
        Arg al[11];
        int    ac;
        XmString xmStringPtr;
         float        pizzaPrice;
         static char  priceString[10];

        toppingArray[(int)toppingIndex].chosen =
           !(toppingArray[(int)toppingIndex].chosen);

         calcNPrintPrice();
}

static void
sizeButtonValueChangedCallback(widget, sizeIndex, call_data)
Widget widget;
caddr_t sizeIndex;
caddr_t call_data;
```

```
{
        chosenSize = sizeIndex;

        calcNPrintPrice();
}

static void
allOrNothingCallback(widget, client_data, call_data)
Widget widget;
caddr_t client_data;
caddr_t call_data;
{
        int i;

        if (client_data == EVERYTHING) {
           for (i = 0; i < NUM_TOPPINGS; i++) {
               toppingArray[i].chosen = True;
               /* Set the state of the button to be set but don't call the
                  valueChanged callback since we have already set the chosen
                  fields. */
               XmToggleButtonGadgetSetState(toppingArray[i].widgetID,
                                            True, False);
           }
        } else {
           for (i = 0; i < NUM_TOPPINGS; i++) {
               toppingArray[i].chosen = False;
               /* Set the state of the button to be unset but don't call the
                  valueChanged callback since we have already set the chosen
                  fields. */
               XmToggleButtonGadgetSetState(toppingArray[i].widgetID,
                                            False, False);
           }
        }

        calcNPrintPrice();
}

static void
exitCallback(widget, client_data, call_data)
Widget widget;
caddr_t client_data;
caddr_t call_data;
{
        exit();
}


/* Function to create a set of radio buttons in a frame to indicate the
```

```
   size of the pizza. */
void
createSizeRadioButtons()
{
       Arg al[11];
       int    ac;
       XmString xmStringPtr;
        int     i;

       ac = 0;
       XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM);
       ac++;
       XtSetArg(al[ac], XmNleftOffset, 10);
       ac++;
       XtSetArg(al[ac], XmNtopAttachment, XmATTACH_WIDGET);
       ac++;
       XtSetArg(al[ac], XmNtopOffset, 10);
       ac++;
       XtSetArg(al[ac], XmNtopWidget, titleLabelGadget);
       ac++;
       sizeFrame =
           XmCreateFrame(form, "sizeFrame", al, ac);
       XtManageChild(sizeFrame);

       ac = 0;
       XtSetArg(al[ac], XmNorientation, XmHORIZONTAL);
       ac++;
       XtSetArg(al[ac], XmNpacking, XmPACK_TIGHT);
       ac++;
       XtSetArg(al[ac], XmNradioBehavior, True);
       ac++;
       rowColumn1 =
           XmCreateRowColumn(sizeFrame, "rowColumn1", al, ac);
       XtManageChild(rowColumn1);

        for (i=0; i < NUM_SIZES; i++) {

          ac = 0;
          if (i == chosenSize) {
             /* This is the default size, must start the button as pressed. */
             XtSetArg(al[ac], XmNset, True);
             ac++;
          }
          xmStringPtr = XmStringCreateLtoR(sizeArray[i].sizeName,
                                           XmSTRING_DEFAULT_CHARSET);
          XtSetArg(al[ac], XmNlabelString, xmStringPtr);
          ac++;
          XtSetArg(al[ac], XmNindicatorType, XmONE_OF_MANY);
```

```
            ac++;
            sizeArray[i].widgetID = XmCreateToggleButtonGadget(rowColumn1,
                                        sizeArray[i].sizeName, al, ac);
            XtManageChild(sizeArray[i].widgetID);
            XmStringFree(xmStringPtr);
            XtAddCallback(sizeArray[i].widgetID, XmNvalueChangedCallback,
              sizeButtonValueChangedCallback, (XtPointer)i);

        }

}

/* Function to create a set of toggle buttons in a frame so the user can
   choose which toppings go on the pizza. */
void
createToppingsButtons()
{
        Arg al[11];
        int    ac;
        XmString xmStringPtr;
         int     i;

        ac = 0;
        XtSetArg(al[ac], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET);
        ac++;
        XtSetArg(al[ac], XmNleftWidget, sizeFrame);
        ac++;
        XtSetArg(al[ac], XmNrightAttachment, XmATTACH_FORM);
        ac++;
        XtSetArg(al[ac], XmNrightOffset, 10);
        ac++;
        XtSetArg(al[ac], XmNtopAttachment, XmATTACH_WIDGET);
        ac++;
        XtSetArg(al[ac], XmNtopOffset, 10);
        ac++;
        XtSetArg(al[ac], XmNtopWidget, sizeFrame);
        ac++;
        toppingsFrame =
            XmCreateFrame(form, "toppingsFrame", al, ac);
        XtManageChild(toppingsFrame);

        ac = 0;
        XtSetArg(al[ac], XmNnumColumns, 3); /* Actually num of rows */
        ac++;
        XtSetArg(al[ac], XmNorientation, XmHORIZONTAL);
        ac++;
        XtSetArg(al[ac], XmNpacking, XmPACK_COLUMN);
        ac++;
```

```
    rowColumn2 =
        XmCreateRowColumn(toppingsFrame, "rowColumn2", al, ac);
    XtManageChild(rowColumn2);

     for (i = 0; i < NUM_TOPPINGS; i++) {
       ac = 0;
       xmStringPtr = XmStringCreateLtoR(toppingArray[i].toppingName,
                        XmSTRING_DEFAULT_CHARSET);
       XtSetArg(al[ac], XmNlabelString, xmStringPtr);
       ac++;
       toppingArray[i].widgetID = XmCreateToggleButtonGadget(rowColumn2,
                                    toppingArray[i].toppingName, al, ac);
       XtManageChild(toppingArray[i].widgetID);
       XmStringFree(xmStringPtr);
       XtAddCallback(toppingArray[i].widgetID, XmNvalueChangedCallback,
         toppingButtonValueChangedCallback, (XtPointer)i);
     }

    ac = 0;
    xmStringPtr =
        XmStringCreateLtoR("Everything", XmSTRING_DEFAULT_CHARSET);
    XtSetArg(al[ac], XmNlabelString, xmStringPtr);
    ac++;
    everythingPushButtonGadget =
        XmCreatePushButtonGadget(rowColumn2, "everythingPushButtonGadget",
                                 al, ac);
    XtManageChild(everythingPushButtonGadget);
    XmStringFree(xmStringPtr);
    XtAddCallback(everythingPushButtonGadget, XmNactivateCallback,
                  allOrNothingCallback, (XtPointer)EVERYTHING);

    ac = 0;
    xmStringPtr =
        XmStringCreateLtoR("Nothing", XmSTRING_DEFAULT_CHARSET);
    XtSetArg(al[ac], XmNlabelString, xmStringPtr);
    ac++;
    nothingPushButtonGadget =
        XmCreatePushButtonGadget(rowColumn2, "nothingPushButtonGadget",
                                 al, ac);
    XtManageChild(nothingPushButtonGadget);
    XmStringFree(xmStringPtr);
    XtAddCallback(nothingPushButtonGadget, XmNactivateCallback,
                  allOrNothingCallback, (XtPointer)NOTHING);

}

/* Function to create a set of toggle buttons in a frame so the user can
   choose which toppings go on the pizza. */
```

```
void
createPriceWidgets()
{
        Arg al[11];
        int    ac;
        XmString xmStringPtr;

        ac = 0;
        xmStringPtr =
            XmStringCreateLtoR("Pizza Price", XmSTRING_DEFAULT_CHARSET);
        XtSetArg(al[ac], XmNlabelString, xmStringPtr);
        ac++;
        XtSetArg(al[ac], XmNbottomAttachment, XmATTACH_FORM);
        ac++;
        XtSetArg(al[ac], XmNbottomOffset, 10);
        ac++;
        XtSetArg(al[ac], XmNleftAttachment, XmATTACH_OPPOSITE_WIDGET);
        ac++;
        XtSetArg(al[ac], XmNleftWidget, toppingsFrame);
        ac++;
        XtSetArg(al[ac], XmNtopAttachment, XmATTACH_WIDGET);
        ac++;
        XtSetArg(al[ac], XmNtopOffset, 10);
        ac++;
        XtSetArg(al[ac], XmNtopWidget, toppingsFrame);
        ac++;
        priceTitleLabelGadget =
            XmCreateLabelGadget(form, "priceTitleLabelGadget", al, ac);
        XtManageChild(priceTitleLabelGadget);
        XmStringFree(xmStringPtr);

        ac = 0;
        XtSetArg(al[ac], XmNmarginWidth, 10);
        ac++;
        XtSetArg(al[ac], XmNbottomAttachment, XmATTACH_NONE);
        ac++;
        XtSetArg(al[ac], XmNleftAttachment, XmATTACH_WIDGET);
        ac++;
        XtSetArg(al[ac], XmNleftWidget, priceTitleLabelGadget);
        ac++;
        XtSetArg(al[ac], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET);
        ac++;
        XtSetArg(al[ac], XmNtopWidget, priceTitleLabelGadget);
        ac++;
        priceFrame =
            XmCreateFrame(form, "priceFrame", al, ac);
        XtManageChild(priceFrame);
```

```
        ac = 0;
        priceLabelGadget =
            XmCreateLabelGadget(priceFrame, "priceLabelGadget", al, ac);
        XtManageChild(priceLabelGadget);
        XmStringFree(xmStringPtr);
         calcNPrintPrice();

}

/* Create the widget tree */
void
createInterfaceWidgets()
{
        Arg al[11];
        int    ac;
        XmString xmStringPtr;

        ac = 0;
        form =
            XmCreateForm(topLevel, "form", al, ac);
        XtManageChild(form);

        ac = 0;
        xmStringPtr =
            XmStringCreateLtoR("Pizza Price Calculator", XmSTRING_DEFAULT_CHARSET);
        XtSetArg(al[ac], XmNlabelString, xmStringPtr);
        ac++;
        XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM);
        ac++;
        XtSetArg(al[ac], XmNrightAttachment, XmATTACH_FORM);
        ac++;
        titleLabelGadget =
            XmCreateLabelGadget(form, "titleLabelGadget", al, ac);
        XtManageChild(titleLabelGadget);
        XmStringFree(xmStringPtr);

        createSizeRadioButtons();

         createToppingsButtons();

         createPriceWidgets();

        ac = 0;
        xmStringPtr =
            XmStringCreateLtoR("Exit", XmSTRING_DEFAULT_CHARSET);
        XtSetArg(al[ac], XmNlabelString, xmStringPtr);
        ac++;
        XtSetArg(al[ac], XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET);
```

```
        ac++;
        XtSetArg(al[ac], XmNtopWidget, priceTitleLabelGadget);
        ac++;
        XtSetArg(al[ac], XmNrightAttachment, XmATTACH_FORM);
        ac++;
        XtSetArg(al[ac], XmNrightOffset, 10);
        ac++;
        exitPushButtonGadget =
            XmCreatePushButtonGadget(form, "exitPushButtonGadget", al, ac);
        XtManageChild(exitPushButtonGadget);
        XmStringFree(xmStringPtr);
        XtAddCallback(exitPushButtonGadget, XmNactivateCallback, exitCallback,
                      (XtPointer)0);

}
```

Applied Computerized Telephony: You won't be left on hold

Steve Aliamus

Hewlett-Packard Direct
1320 Kifer Road
Sunnyvale, California
408-730-6046

"Hey Brian, what does this 'TONER LOW' message on the printer mean?"

For the past week, Brian had been regularly removing the LaserJet's toner cartridge, shaking it, and reinstalling it into the printer. He had hoped to avoid buying a new toner cartridge for as long as possible, but the daily shaking ritual was now occurring every hour. Time to break down and call HP Direct's toll free 800 number.

When Brian's call reached the HP Direct switchboard, an Applied Computerized Telephony (ACT) system detected the phone call, accessed information on Brian's account and past purchases, and displayed the information on the terminal of a waiting sales consultant. Simultaneously, the phone call was ringing on the same sales consultant's telephone.

"Hello, this is Maya with Hewlett-Packard Direct. How may I help you?"
"Hi, this is Brian Hunt, and ..."
"Brian! How does your department like the LaserJet II printer?"

Not only does Maya know that Brian has a LaserJet printer, but what model, when it was purchased and any optional accessories. With minimal effort, Brian orders his toner cartridge. Several months later, Brian receives a call to remind him that his toner cartridge may be running low, based upon the usage rate determined from his last order.

The above scenario can happen because of ACT, which addresses the need of businesses to process both voice and data technologies. ACT consists of hardware and software components which integrate HP Computers with Northern Telecom Meridian 1 and SL-1 telephone switches (PBXs) for a new generation of integrated voice and data applications.

This paper discusses the use of ACT and the components needed for its installation. Included are details of a pilot ACT project developed and installed at HP Direct, and the potential benefits seen in our organization.

ACT provides information about incoming telephone calls, such as the calling number and number dialed, directly to application programs. This benefits businesses in telesales, telemarketing, customer assistance, service and support, help desks, collections, distribution and purchasing. Applications can use ACT to originate, answer and manipulate telephone calls automatically, allowing these telephone intensive businesses to increase productivity, improve customer service, and increase revenue.

Increased Productivity

ACT provides the ability to route phone calls to the most appropriate agent, and simultaneously present customer information on the same agent's screen. This increases the volume of calls that can be received. Needless call transfers between agents are eliminated. Information about the purpose of the call, based upon the number dialed by the caller, is presented on the terminal screen before the call is answered. Agents answer each call appropriately on an individual basis. Telephone agents no longer need to be separated by product line.

Outbound environments also benefit from ACT. With automated dialing applications, agents process calls quickly, and avoid manually dialing numbers and listening to busy signals and unanswered ringing.

Improved Customer Service

Callers are handled efficiently and professionally. The identity of the caller stays with the call as it is transferred, so customers need not repeat any information to each new agent. Intelligent routing of calls ensures customers are sent to the most appropriate destination.

Increased Revenue

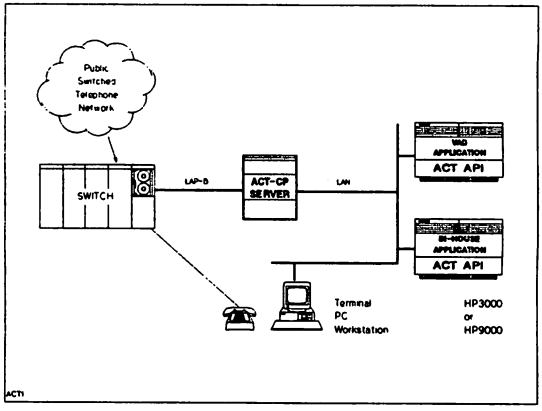
Since calls from customers are processed more intelligently, agents are able to handle more calls. Increased call volumes lead to more customer contacts and sales. Telephone calls are also shorter, since customers can be identified immediately by their inbound phone number, and transfers and hold times are minimized.

Corporate Networks Operation (CNO)

Hewlett-Packard and Northern Telecom formed an alliance organization composed of both technical and marketing personnel dedicated to ACT solutions. Consultation, installation and configuration of ACT can be handled by CNO. An agreement for the exchange of technical information and synchronized escalation of hot sites has been signed by both companies. Problem resolution for the ACT product is handled through the established support arms of both HP and NT.

The ACT components

The ACT product consists of an ACT Call Processing Server, Server Software and the ACT Application Programming Interface (API). (See figure 1) The server software resides on an HP9000 Series 300 server connected to the PBX. The server manages connections between application programs and the call processing features of the PBX. ACT messages originating from application programs are converted automatically into specific PBX messages to invoke the required functions on the PBX.



(figure 1)

Standard IEEE 802.3/Ethernet and TCP/IP networking are used for the communications between the application programs and the server. This open networking environment allows maximum flexibility in designing a solution for multiple computers with a single ACT server.

Northern Telecom Meridian 1 and SL-1 PBX

The Meridian 1 and SL-1 family of PBXs are customer premise digital telephone switches with a variety of advanced customer calling features, data connectivity, and networking.

The messaging interface on the PBX which passes and receives command and status information to the ACT Server is the Meridian Link, a LAP-B (RS-232 or 422) synchronous link which operates at speeds up to 19.2 Kbps. The Meridian Link consists of a software package and an Enhanced Serial Data Interface (ESDI) card on the PBX.

Telephone companies offer a service which passes the caller's telephone number to the called party. This caller identification, Automatic Number Identification (ANI), is passed across the Meridian Link to the application program. In order to receive ANI information, the appropriate trunks must be ordered from the phone company, and specialized trunk interface hardware and software on the PBX must be purchased. In addition to ANI information, Direct Number Identification Systems (DNIS) is passed to the application program. The DNIS number is the phone number that the customer dialed to reach the PBX. For businesses that handle several incoming numbers, DNIS allows the application program to display the appropriate information for each different incoming call.

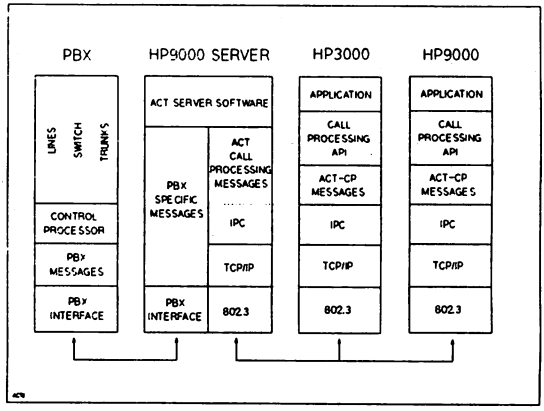
Although the application programmer will probably not need to know all the details about the PBX, a general understanding of it is helpful during the coding and debugging stages of your ACT project.

The ACT Call Processing Server

The ACT Call Processing Server translates ACT messages originating from the application program and converts them to Meridian Link messages. The ACT server is a dedicated system, and should only be used for ACT and telephone switch related activities. The ACT server consists of:

- HP-UX based platform
- Internal Disk
- Standard TCP/IP networking
- Telephone switch specific interface

The ACT Server utilizes standard HP LAN Link/300 and TCP/IP software for end-to-end connection oriented message transport with the client application. Interprocess communications provide the higher level communications services. For communications with the PBX, the HP X.25/9000 interface is used. (See figure 2) The server is also responsible for enforcing security, allowing users to access only the set of telephone capabilities for which they are authorized.



(figure 2)

During normal operations, the ACT server will generally display PBX and ACT status messages. While developing the application program, it is important to locate the ACT server near the desk of the programmer. Certain types of debugging are possible only if status codes and messages displayed on the ACT server terminal are visible, so having the server nearby makes these tasks easier.

The host computer

Both the HP3000 and HP9000 can be used as platforms for application programs that will interact with the ACT server. On HP3000 systems, the LAN link product, which includes the necessary TCP/IP and Net/IPC software, is required. For HP9000 systems, the LAN/300 or LAN/800 link product, and NS-ARPA services, which includes Berkeley Sockets, are required.

Application Program Interface (API)

The ACT API is designed to provide a simple means for the application programmer to generate ACT protocol messages, and to shield the programmer from the underlying low level network procedure calls. The API consists of callable subroutines, such as ACTMAKECALL, ACTANSWER, ACTHOLD, ACTTRANSFER, and ACTDROPCALL, and makes it easy for the programmer to integrate ACT functionality into new and existing applications.

The role of the API is to take the parameters passed from the application program, place the parameters into an ACT formatted buffer, then send the request to the server via the network. Similarly, when a response is returned by the server, the API accepts the response from the network and returns data into application variables.

HP Direct's interest in ACT

HP Direct is both a marketing and a distribution channel for a specific line of HP products consisting of computer supplies and accessories. In addition, the telemarketing channel sells items such as calculators, plotter supplies and low-cost instruments manufactured and marketed at other HP sites.

In its role as a channel of distribution, HP Direct uses catalogues, mini-catalogues, fliers and promotional pieces to stimulate sales through a telemarketing force of over 100 people. This sales force handles a host of supplies and accessories, totaling over 20,000 parts.

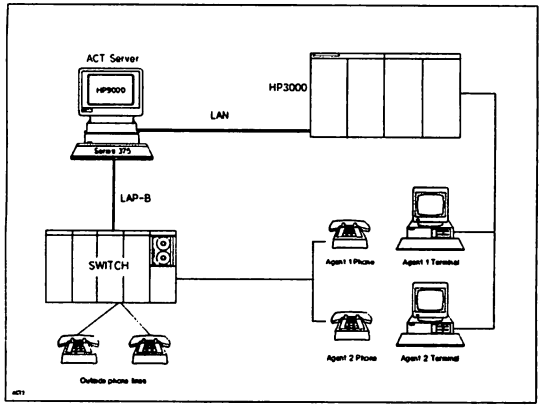
With several thousand phone calls per day coming through HP Direct's switchboard, we are a prime candidate for ACT technology. Our telephone sales consultants would benefit in several areas. Customers who call our 800 number often do not have their customer account number handy, causing a search of their name to be made on our database. Although searching only takes a moment, it is time that could be used to lookup the customers past purchases or buying trends. In addition, the phone call is longer, which means bigger phone bills. ACT eliminates the need for customers to have to provide their account number, as it can be obtained automatically from their phone call.

Another benefit of the use of ACT at HP Direct is the transferring of calls. Many times, a customer needs to speak to a product specialist. With so many different products being sold through HP Direct, certain sales consultants are given training on specific products. When a phone call reaches our switchboard, and a sales consultant realizes after a short conversation that the customer really needs to speak to a product specialist, the phone call is transferred. Without ACT, the customer must again identify himself and restate his request, which is extremely frustrating. ACT provides the ability to transfer the telephone call and a screenful of data (including a short remark) to a waiting agent.

HP Direct Configuration

Before attempting to implement ACT within our Telemarketing department, we created a test environment separate from our 800 number. By working closely with CNO and our Telecom department, we were able to configure our PBX so that only two phones were actually using ACT. That way, we could test all the features of ACT in a controlled environment, without adversely affecting our production systems.

For our pilot ACT project, two terminals, two agent telephones and two "outside" telephones were configured. (See figure 3) The outside lines would simulate a customer call, with both ANI and DNIS (customer phone number and number dialed) being delivered to the PBX.



(figure 3)

Using the API intrinsics

The first thing our sales order entry application needed was a connection to the ACT server. A connection is established by calling the **ACTINITIALIZE** intrinsic. If successful, a channel identifier, and the software revision level of the server and switch, are returned. The channel identifier is used in all subsequent requests to the server.

Our application was not going to use ACT to make outbound calls. Yet, the **ACTMAKECALL** intrinsic can place a telephone call from a phone which is under server control to any phone in the global telephone network. In the future, ACTMAKECALL will be able to automatically detect busy signals and unanswered calls, further maximizing agents on-line time.

Normally, when an application program issues an FREAD or FWRITE on a device, the program is interrupted until the read or write has completed. There are, however, situations where the application program should have the ability to interrupt a read or write. With many of the ACT intrinsics, NOWAIT I/O is used, because it is not known when a phone will ring, or when a customer may hang up the phone. To address this issue, the **ACTEVENTMONITOR** intrinsic is used to monitor activity on an agent's telephone extension. Activities include going on-hook or off-hook (hanging up or picking up the receiver), incoming calls (with ANI and DNIS information) and other events dependent on switch features. The application program can monitor multiple phone extensions, and choose to poll or wait for event completion using IOWAIT or IODONTWAIT.

When the phone rings, you need to answer it. The **ACTANSWER** intrinsic responds to notification of an inbound call by placing the called telephone in the off-hook state. This has the same effect as lifting the handset from the cradle. But, before the called phone can be answered, it must be monitored with ACTEVENTMONITOR. So, during the initialization stage of the application program, it is mandatory that the agent and his phone extension are provided to the ACT API. Otherwise, ACTANSWER will have no affect on the ringing phone.

If a customer phones, and needs to speak to a different agent, the call must be transferred. **ACTTRANSFER** performs an unsupervised (blind) call transfer to a new number by connecting the transferred party to the specified number. The transferring party will be dropped from the call at the time the transfer begins. (Our version of ACT did not yet have consultive transfer implemented). The agent that receives the transferred call is provided all the original information about the call (ANI and DNIS included).

After the telephone call is completed, you must hang up the phone. The **ACTDROPCALL** intrinsic disconnects a party from a call. It is common to receive an error while dropping a call, as customers will generally hang up their phone before ACT has a chance to process. In this case, ACTDROPCALL returns a "261:ERR_DIS_FLD" error, indicating that the call has already been marked as disconnected because one of the parties has manually hung up.

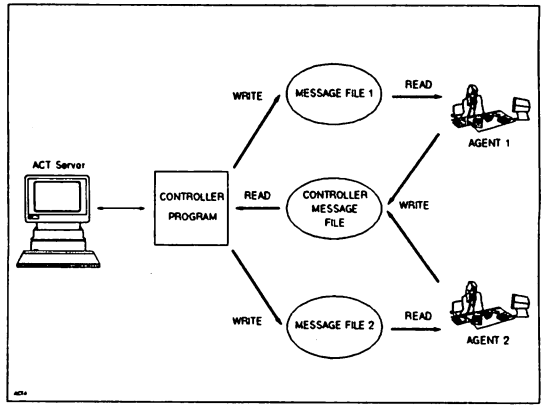
To disconnect from ACT completely, **ACTTERMINATE** is used. It will shutdown the connection between the application and the ACT server. All event monitoring for the channel is terminated as well.

Using a controller program

The ACT API is complicated slightly by the fact that telephone operations take a relatively long time in computer terms. To allow this waiting time to be productive, the API returns control to the application immediately after accepting requests that may require a long wait (waiting for the phone to ring, for example). The application can then choose to poll or wait for event completion.

To the programmer, using the API directly within your own application can become complicated. It may be advantageous to place the ACT intrinsics in another program to avoid the complexity of adding NOWAIT I/O to any new or existing application. Another reason to write a separate program is to facilitate the transfer situation. When transferring a call from one phone to another, the screen data associated with the call also needs to be sent. Every application must be able to communicate with every other application user. Adding all the code to handle each call processing function and error condition, and communication information for other users on the system would cause a significant change in an existing application.

In our application, we created a separate controller, acting as a signal program that would interface to the ACT intrinsics on behalf of our application. Each application needing to communicate with the controller would write a request to a message file that was writable by all users. When the application needed to read information, another message file, readable only by the specific user, was used. (See figure 4) It was much easier to modify our existing application to read and write message files as a short term solution. In the future, we plan to make the investment to integrate ACT directly into our application.



(figure 4)

Our prototype ACT project demonstrated the ability to answer an incoming telephone call and display a customer record in a TurboIMAGE database from the ANI, via a single function key. The DNIS was displayed on the screen, so our sales consultants could see what number the customer dialed to reach our switchboard. In addition, a message appeared on the screen indicating whether the phone call was a new call or a transfer. By the time you could say "Welcome to Hewlett-Packard Direct, how may I help you?", all the information had appeared on the screen.

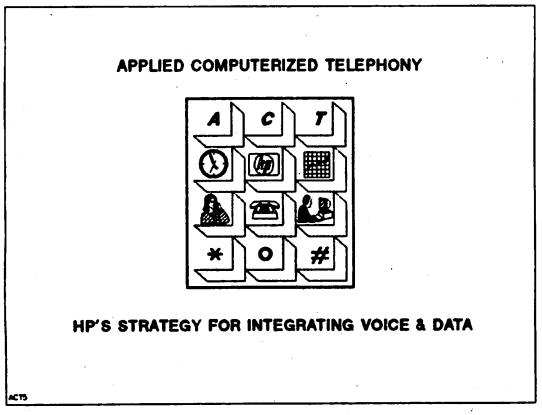
To perform a call transfer, our application displayed a list of agents currently connected to the ACT server, and prompted the sales consultant for a short message to be sent along with the transferred call. On the sales consultant's terminal receiving the transfer, a message indicating that a call was being transferred, who was transferring it, and the short message were displayed. Also, the same customer record appeared on the screen, so the new sales consultant would not have to ask the customer to repeat any information.

Although our pilot ACT project has not been installed into production, many potential benefits are anticipated. In addition to automatic customer identification and call transfers described above, we found that the ANI from abandoned calls (when a customer hangs up before reaching a sales consultant) could be captured. Our telemarketing department plans to use this data to improve service levels for our customers.

Conclusion

Numerous forces are driving companies to implement new telephone and computer technologies to enhance sales, marketing and customer service operations. Some of these include rising costs of face-to-face sales efforts, expanding geographic market areas, cost effectiveness of centralization, emphasis on customer service as a product differentiator, and overall increases in global competition. Marketing and service organizations are expected to represent a larger percentage of corporate information systems expenditures over the next several years.

Applied Computerized Telephony integrates telephone switches with HP3000 and HP9000 applications, creating an exciting new generation of voice and data systems. HP Direct's prototype ACT system demonstrated to us that the technology can create a solution that provides impressive productivity gains, increased customer satisfaction, and sales opportunities never before possible. These same benefits can be seen in your organization when ACT is integrated into your telephone applications.



AIFs ON MPE/XL

by
Rajesh Desai
Jeanne Elmer

Commercial Systems Division
Hewlett-Packard
19447 Pruneridge Ave
Cupertino, CA 95014
(408)725-8900

INTRODUCTION

In the past, Independent Software Vendors have often required access to operating system internal information in order to provide sophisticated end-user solutions. Disseminating dynamically changing data structures and information control flow has been a challenge to platform suppliers, and not always done in a timely or consistent fashion. Independent Software Vendors face the challenge of maintaining their products conforming to the modifications done to the operating system from release to release. With the advent of MPE/XL, Hewlett-Packard embarked on an ambitious program to provide equivalent means for developing advanced solutions while addressing the problems posed by system evolution. These program objectives led to the development of the Architected Interface Facility (AIF) products.

The Architected Interface Facility products are low-level system interfaces designed to expose internals in a controlled manner or, to export or enhance existing system functionality and yet remain independent of a system release. These products in many cases, are not unlike intrinsics. However, they do differ in one significant way: Architected Interfaces assume a privileged user and therefore limits their error checking and allow access to sensitive system data and functionality. Conversely, intrinsics are "bullet- proof" and often don't meet the performance or functional needs of Independent Software Vendors. Currently there are three Architected Interface products - Operating System, Measurement Interface and Procedure Exits.

ARCHITECTED INTERFACE FACILITY : OPERATING SYSTEM

The Architected Interface to the MPE/XL operating system provides access to internal system data and functionality. Correspondingly, there are two types of interfaces available within this product, Information Access and Functionality Access interfaces. Information Access AIFs allow read or write access to internal system tables. For each class of objects for which interfaces are provided, there are two procedures: a GET and a PUT. The GET interface will return information about a specific instance of a class as identified by input keys. The PUT interface also accepts an instance of class from the caller and updates system tables to reflect the state requested. Additionally, an anchor interface is provided to retrieve one or more instances of an object class reflecting the current state of the system. Functionality Access interfaces allow the developer to take advantage of operating system functionality. Below is a list of the class of objects for which Information Access Interfaces and Functionality Interfaces are currently available.

Information Access

- Accounting
- File
- Job/Session
- Process
- Reply Information
- System Configuration
- Spooler
- System Wide

Functionality Access

- Change Logon
- Convert Address (CM to NM)
- File Close
- Ports
- Global Object
- Time

ARCHITECTED INTERFACE FACILITY : MEASUREMENT INTERFACE

The Architected Interface to the Measurement Interface allows read access to internal measurement counters. Counters are the method that the operating system uses to track events that occur on the system. A counter unit of measure can be either count or time. Count is the number of times an event occurred or the quantity of an event that happened. Time is the length of time that an event happened or the time stamp when an event occurred. Counter values are returned for four types of information by the AIFs to the measurement interface :

- Global Counter Information
- Process Counter Information
- I/O Counter Information
- Processor Counter Information

ARCHITECTED INTERFACE FACILITY : PROCEDURE EXITS

The Architected Interface Facility: Procedure Exits product enables you to replace or augment system functionality on MPE/XL. Software solutions may be accomplished through run time interception of MPE/XL procedures residing in NL.PUB.SYS or other system libraries. It does this by letting the developer specify certain procedures to be executed in place of, or in addition to, existing procedures within the system or user code in both compatibility mode

(CM) and native mode (NM). This specification may either be performed on a system-wide or a process-local basis, to allow limiting the scope of effect. User supplied procedures to execute at procedure interception time are defined as handlers. The procedure that has been intercepted is defined as a target. Handlers can execute prior to (invocation handler) or upon completion (termination handler) of a target. They may also execute instead of the target procedure, defined as stubbing out a procedure. Access to the target procedure's parameters is made available to handlers for inspection and/or modification. The Architected Interface Facility: Procedure Exits product binds and unbinds handler routines to targets dynamically, without the need for rebooting or relinking the system. Binding and unbinding affects all processes currently running as well as those subsequently created.

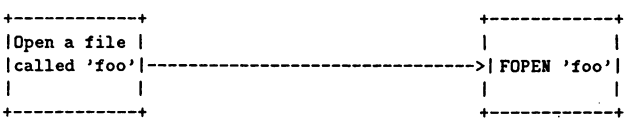
EXAMPLES

Below we will discuss some examples of how you could use the Architected Interface Facility products to accomplish a variety of tasks.

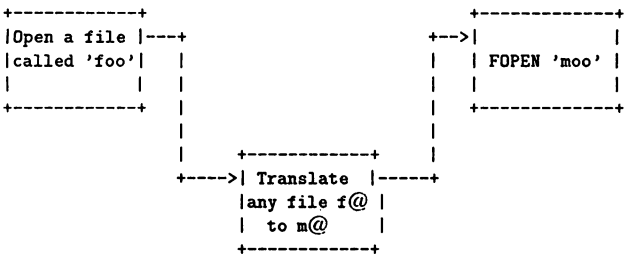
Generalized File Equations :

One way to accomplish generalized file equations is through the use of the Architected Interface Facility: Procedure Exits. By enabling an invocation (a handler that executes prior to the execution of a target) procedure exit on file open, an application could catch every file open on a system. The procedure exit could then examine the file name, compare it to a list of special file equations, and then pass file open a modified file name. The following diagram illustrates the execution:

Normal Execution



Execution with a Procedure Exit



In this example, the application tries to open a file called 'foo'. Under normal circumstances, the application calls file open with the file name, and file open opens the file. The application user can specify a normal file equation such as :

FILE FOO = MOO

This will make the file system open a file called 'moo', instead of 'foo'. But what if the user does not specifically know what the file will be called. Furthermore, suppose the user wants all files that are in a group called could enter file equations for each file, if the names are known, and if the number of files does not exceed the file equation table. But it would be more convenient, if the user could just tell the applications to replace all files from the group 'group1' with all files from a group called 'group2'. The user wants to write a file equation like this:

FILE @.GROUP1 = @.GROUP2

This feature would be nice, but it does not exist. Wild carded file equations would be relatively simple to implement, however, the application writer could use the Architected Interface Facility: Procedure Exits. Since all calls to file open are intercepted by a procedure exit, the procedure exit just needs to look for the file equation (the details are left for the designer of such a utility), and modify the file name appropriately in the parameter list to FOPEN upon interception. This could all be done with the user being totally unaware of what is happening.

Imagine trying to compile a program that includes several other files. Now let's say that the included files need to be modified. The programmer could use wild carded file equations to point the compiler to the new include files in a work group, leaving the original files untouched. This could be very useful if several people are working on the same group of files or if they must use different versions of the same file. Each version of the files could be kept in separate groups. The programmer could use wild carded file equations to point the compiler to the correct group.

Event Handling :

Currently, several events are monitored throughout the operating system via the measurement interface which are made accessible to the developer with the Architected Interface Facility: Measurement Interface. However, not all events of interest may be monitored. A developer interested in monitoring an event can take advantage of the Architected Interface Facility products to do so. Take for example the event of logging on to the system. A system manager may wish to limit the number of logons per user for performance reasons. This could be done with a combination of Architected Interface Facility: Procedure Exits and Architected Interface Facility: Operating System. A product could intercept the :HELLO command using AIF: Procedure Exits. This product could then call AIFACCTGET within the AIF:OS product and retrieve the logon count for the user logging on. If the logon count is zero, the handler would allow the logon to continue. Otherwise, the procedure could be stubbed out with the handler reporting a failed logon.

Automatic File Archiving :

The purging of files is a common event on the operating system. People from time to time mistakenly purge a file that they didn't mean to. The file in some cases can be recovered from a back up tape, but may often not be current or even backed up. A nice feature to have would be the ability to save off those files purged in a backup group for a period of time so that it may be recovered if it is decided that the file is still needed. One way to accomplish this task is to use a combination of the Architected Interface Facility: Procedure Exits and the Architected Interface Facility: Operating System products. In this example, we will arm the procedure FCLOSE with both an invocation handler and a termination handler. AIFs to the Operating system will be used to gather information needed for for both handlers and initial setup. Below is an algorithm that could be used to accomplish this task. The algorithm assumes that every group on the system has a corresponding back-up group within the same account in which to save discarded files. This algorithm is an outline and does not give specific calling sequences to the procedures used or other specific implementation details.

Initial Setup :

(a) Call AIFSCGET to determine the maximum number of pins allowed on the system.

(b) Call AIFGLOBACQ to acquire a global object. In this object, set up an array from one to maximum pin with the following record structure :

```
shared_data_array = record
                      file_close : boolean;
                      file_name  : packed array [1 .. 48] of char;
                    end;
```

We will index into this array based on pin number to share information between invocation and termination handlers to FCLOSE.

(c) Initialize data structure to false and blanks.

(d) Install an invocation handler and termination handler on FCLOSE.

Invocation Handler :

```
{ Examine parameters to fclose and determine disp parameter passed }
disp := FCLOSE(disp);

{ only worry if we are deleting the file }
if (disp = delete) then
  begin

  { Determine our pin to index into global array shared by this handler }
  { and the termination handler }
  my_pin := AIFPROCGET;

  { fill in data we want to communicate to the termination handler, }
  { namely that a fclose with a disposition of delete has occurred and }
  { the name of the file being deleted }
  shared_data_array[my_pin].file_close := True;
  shared_data_array[my_pin].file_name  := file_name;{in parameter list}

  { Modify the disp parameter to FCLOSE to close as a permanent file }
  FCLOSE(disp) := CLOSE AS A PERMANENT FILE;

  end;
```

Termination Handler :

```
{ Determine our pin to index into global array to retrieve information }
{ from invocation handler }
my_pin := AIFPROCGET;

{ Check array in global object indexed by pin to determine the file }
{ was being closed with a delete disposition }
if shared_data_array[my_pin].file_close = True then
   begin

   { Issue a programmatic COPY of the file to the corresponding back-up }
   { group using the HPCICOMMAND intrinsic. }
   HPCICOMMAND(command     = copy,
               source      = shared_data_array[my_pin].file_name,
               destination = corresponding archive group );

   { Issue a programmatic PURGE to now purge the file, again using the }
   { the intrinsic HPCICOMMAND }
   HPCICOMMAND(command     = purge,
               file        = shared_data_array[my_pin].file_name );

   { reset the fclose_disp parameter in the global array to false }
   shared_data_array[my_pin].file_close := False;

   end;
```

CONCLUSION

In summary, the Architected Interface Facility products, allow for supported access to internal system data, functionality, or provide enhanced functionality. The Architected Interface Facility: Operating System and Architected Interface Facility: Measurement Interface are available on MPE/XL 2.1 and later. The Architected Interface Facility: Procedure Exits is available on MPE/XL 3.0 and later.

Paper #3239

ALLBASE/SQL High Availability Features

Alex Tsukerman
Hewlett-Packard
19447 Pruneridge Ave
Cupertino, CA 95014
(408) 447-6728

Abstract

A significant work has been put into ALLBASE/SQL to increase product availability, i.e. minimizing of planned database downtime. The new high availability features include:

1. Online STORE facility, which allows the DBEnvironment to be backed up without interrupting transaction processing.
2. Multiple log files. This feature allows log files to be added/deleted/stored during normal system operation. It also provides flexibility in managing the log space.
3. Improved algorithm for dual logging. This feature provide fault-tolerance of log files with respect to a single device failure.
4. More robust rollforward recovery algorithm. This feature makes the process of the forward recovery less error prone.
5. Support for log files on raw devices. Raw devices provide higher performance.
6. Dynamic space expansion where DBEfiles which become full are automatically expanded within the user-specified limits.

This paper describes how to use the above features to achieve smooth and continuous operation of the DBEnvironment with respect to DBEnvironment backup/recovery. It also provides a short overview of transaction logging.

The paper is primarily intended for the Database Administrators and systems personnel.

Logging overview

ALLBASE/SQL uses logging for the following purposes:

1. To recover a DBEnvironment to a consistent state after a system crash. Updates of all committed transactions are guaranteed to be recovered. This feature is called soft crash (or rollback) recovery.
2. To re-create the DBEnvironment from the previous backup copy up to the current time in the case of media failure on the DBEfile(s). This is a rollforward recovery.
3. To re-create the DBEnvironment from the previous backup copy up to some other in the past (partial rollforward). This feature may be used to remove all database updates after certain time.
4. To rollback user transactions which cannot be successfully completed.

In other words, Atomicity, Consistency, and Durability properties of the transaction model are achieved through logging. The other property, Isolation, is achieved using locking and will not be discussed here.

The log consists of multiple files, and each of them may be duplicated for increased robustness. The system maintains a directory of currently defined log files. The log files may be added/deleted/displayed while the system is operational.

The contents of a log file is log records. Most log records describe updates to the user data. Typically, such record contains the previous value of the data (a before-image), and the new value of the data (an after-image). Before-images is used to remove the results of incomplete transactions, and after-images are used to make the results of completed transactions persistent. The log records must be written the disk before the corresponding data records are, and they also must be written to the disk before the user received confirmation that the transaction has committed. This strategy (write-ahead logging) is common for most commercial database systems.

An individual transaction may span multiple log files. The system itself automatically switches from one log file to another when the current log becomes full.

Depending on the user requirements, a DBEnvironment can run in either an archive mode or a non-archive mode. In archive mode, it is possible to recover from media failures using DBEnvironment backup (archive) copy, and the log files, created after the backup was taken. It is also possible to recover from system crashes by using only the DBEnvironment on the disk and the log file(s) on the disk at the time of crash. In a non-archive mode, it is only possible to recover from system crashes. Although one can use a backup copy of the DBEnvironment in non-archive mode, all the updates to the dababase which happened after this copy was taken, are lost. From the operational point of view, an archive logging may require more log space, and the log space is reused differently - the logs must be periodically stored to become usable again. In this article, we will be mostly talking about archive mode, since this is where our availability enhancements really pay off.

Continous operation

It is easier to explain our availability enhancements by painting a picture of a (if not ideal, then at least desirable) continous DBEnvironment operation. The user should be able to run in a environment as follows:

- Periodically, say once a week, store the DBEnvironment to some archival media (usually, tape) without having to bring the DBEnvironment down.
- More frequently, say once a day, store the log file(s) consumed by this period's operation to some archival media. Again, the DBEnvironment is up when this is done.
- If it is desirable to change the logging parameters, such as to create a new log space or to remove the old one, this can also be done without bringing the DBEnvironment down.

In order to store the DBEnvironment while the user transactions are still active, STORE has been enhanced to do its operations online. In order to manage log space while the user transactions are still active, the log space can be divided into multiple parts (files), where each part can be processed with a significant degree of independence from other parts. Now we explain online STORE and multiple log files in more detail.

Online STORE

Online STORE is used to create a backup copy of a DBEnvironment when the DBEnvironment is active. The copy which is created can be used in the subsequent rollforward recovery. In fact, the copy can only be used in the rollforward recovery, since the DBEnvironment image it contains is inconsistent. This is in contrast with the backup copy made when the DBEnvironment is not active. The user should be aware of it, although ALLBASE will enforce the rule automatically, so that CONNECT to an inconsistent DBEnvironment is not possible. During rollforward recovery, the logs are applied to create a consistent (preferably, up-to-date) copy of a DBEnvironment. It is also important to know that the TSTORE-II is required on XL to do an ALLBASE online backup.

ALLBASE requires that at least the log files which were in use during an online STORE window must be applied during rollforward recovery, otherwise the DBEnvironment will not be consistent. Therefore, the user must ensure that all these logs are available at recovery time.

On a cautionary note, an Online STORE is a data-intensive operation which competes for system resources with the user transactions. Therefore, even if the system does not have to be shutdown, it may be desirable to run the STORE concurrently with non-prime time batch jobs, as opposed to running it at the peak of OLTP workload. If one really has to run it concurrently with OLTP, it may be desirable to lower the STORE priority, in which case it will take longer, but the impact will be less noticeable to the OLTP user. We don't have any hard numbers to quantify the above recommendations.

Multiple logs

Multiple log files allow for smooth management of both the log space and the log backup schedules (referred to as a Switchlog enhancement in the ALLBASE literature). If the log space is insufficient, which results in frequent log full conditions, a new log file may be added while the system is still operational. And if the log space seems to be excessive, the extra log files may be deleted. It is also possible to move log files from one device to another.

Log files which are not currently in use by the system can be stored on some archival media. The frequency of storing log files depends on the considerations of possible data loss and of necessity to free the logs as soon as possible. Data loss may result from a damaged log file on the disk. Such log cannot be used for the rollforward recovery. To solve this problem, the DUALLOG feature could be used (described later), and/or the logs should be stored off with sufficient frequency. But there is another reason for storing the logs with some regularity - the system will not reuse the log (in archive mode) until it has been stored.

In addition to the functionality of add/delete/store logs, a fairly comprehensive display facility is provided to show the current state of the log directory. The information includes log file names, their respective sizes, and whether or not the logs have been damaged. There is also information about the latest online backup and the overall amount of free log space.

Dual Logging

Dual logging is used to increase log availability. It employs a mirrorring scheme, where instead of one file for each log, two files are used. Log writes are issued to both logs. If a write to one log file fails, it is marked as "bad", and the system writes only to a "good" log file. "Bad" logs are not used for recovery, and when the log is stored, only a "good" file is chosen. When a system switches into a log again, the "bad" indicator is reset. In other words, "bad" simply means that the log does not contain complete information, and is therefore unusable for recovery. Of course, a primary log file

and a backup log file should never be placed on the same physical device, otherwise they would fail together, thus defeating the purpose of mirrorring.

It is natural to compare dual logs with mirrorred disks. Mirrorred disks use more space, since the whole disk is mirrorred, not just individual log file(s), and therefore are more expensive. On the other hand, the mirrorred disk I/Os are issued by the Operating System in parallel, while ALLBASE serially writes to the primary and then to the mirror. Therefore, performance of the mirrorred disks is better. In the future, however, ALLBASE may come up with a parallel scheme for dual logging.

Rollforward recovery

Although the feature existed in previous releases, some new work has been done to make the process of rollforward recovery more robust and user-friendly. The most important reason for performing rollforward recovery is media failure on either a DBEfile or a log file. Another reason for the rollforward recovery may be massive logical contamination of data. For example, consider a case where a lot of data which were entered after May 10 was absolutely meaningless. In this case, partial rollforward may be recommended to bring a DBEnvironment to some consistent point in the past.

Rollforward recovery is performed using SQLUTIL. The following sequence of steps is recommended:

1. Store all the log files which are still on the disk and have not been stored before to the tape (or other archive media) using the RESCUELOG command. In order to simplify this step, the user may attempt to perform a SHOWLOG command (with an offline flavour), which would display all the log information. Only good logs which have not been stored should be rescued. It is possible, however, that even an offline SHOWLOG will fail as a result of media failure. But in any case, it is always worth trying.

2. Restore the previous online backup copy using the RESTORE command.

3. Do an offline SHOWLOG command on the restored DBEnvironment. It will display the sequence number of the first log to be applied during forward recovery.

4. Initiliase recovery using the SETUPRECOVERY command. This command specifies whether the DBEnvironment should be rolled forward completely up-to-date or up to a specific timestamp. Further, it provides the logging parameters which will take effect after the DBEnvironment has been recovered.

5. For each log to apply, use the RECOVERLOG command. If a log is not on disk, it has be preceded by a RESTORELOG command to bring the log in. The system will automatically verify the correctness of log sequence numbers. When RESTORELOG is used, it is desirable to rename the log file, otherwise different incarnations of the same log file may overwrite each other.

6. If recovery to a specific timestamp was specified, the system will automatically stop when this timestamp has been reached. Otherwise, the user should issue ENDRECOVERY command after the last log has been applied.

If, during rollforward recovery, there was an I/O error reading the log, but the DBEnvironment is physically consistent, the rollforward recovery will terminate with a warning. In this case, the DBEnvironment is usable, but it is most probably not up-to-date.

If a system crashes during rollforward recovery, the user is advised to reapply the last log. The system may return a warning that this log is out of sequence. This warning can be ignored; it simply means that the system has already completed work with the log, and the next one should be applied.

Logs on raw devices

Raw logs is a performance feature. The additional performance is gained by bypassing the UNIX file system, and by removing indirect blocks. On the other hand, there are some yhings about the raw devices that the user should take into account:

- The raw files require all I/O transfers to be 1K multiples. In order to accomodate this rule, the system may have to round the current log block to a 1K boundary. As a result, all Unix (not necessarily raw!) logs may be somewhat larger than their XL counterpart.
- DBA should exercise some additional care when using partitions beacuase of possible overlapping with other partitions.

Dynamic space expansion

Dynamic space expansion is a feature which allows the DBEnvironment system to increase the DBEfile space when needed without any user intervention. Before, if a DBEfileset would become full, the user would have to manually create another DBEfile and add it to the DBEfileset. In the meantime, no user transactions which need this additional space would be able to proceed. Also, adding another DBEfile is a DDL operation, and if the DBEnvironment is run in DML only mode, it has to be shut down, brought up with DDL, shut down again, and brought up without DDL. Now, all the user has to do is to define a DBEfile as being expandable. More precisely, at the time of creation of the DBEfile the user may specify the initial space allocation for the file, the maximum space allocation for the file, and the expansion increment. Whenever the system runs out of data or index space in a specific DBEfileset, it checks all the expandable DBEfiles in the fileset and expands the one whose ratio of the maximum size to the current size is the smallest. This algorithm ensures an uniform growth of all the expandable DBEfiles, and it works even if DDL is disabled.

3240: ALLBASE/DB2 CONNECT - SQL Gateway to IBM's DB2

Jim Nagler
Hewlett Packard
Commercial Systems Division
Cupertino, California 95014
408 447-4048

INTEREX Conference; August 5-8, 1991; San Diego, CA

ABSTRACT

ALLBASE/DB2 CONNECT provides transparent access to DB2 data in a standard relational format. Users may query, update, load, unload, create, or drop data from DB2 using ISQL, ISQL command files, Information Access, NewWave Access or NewWave Agent scripts.

The benefits of this product to the user are numerous:

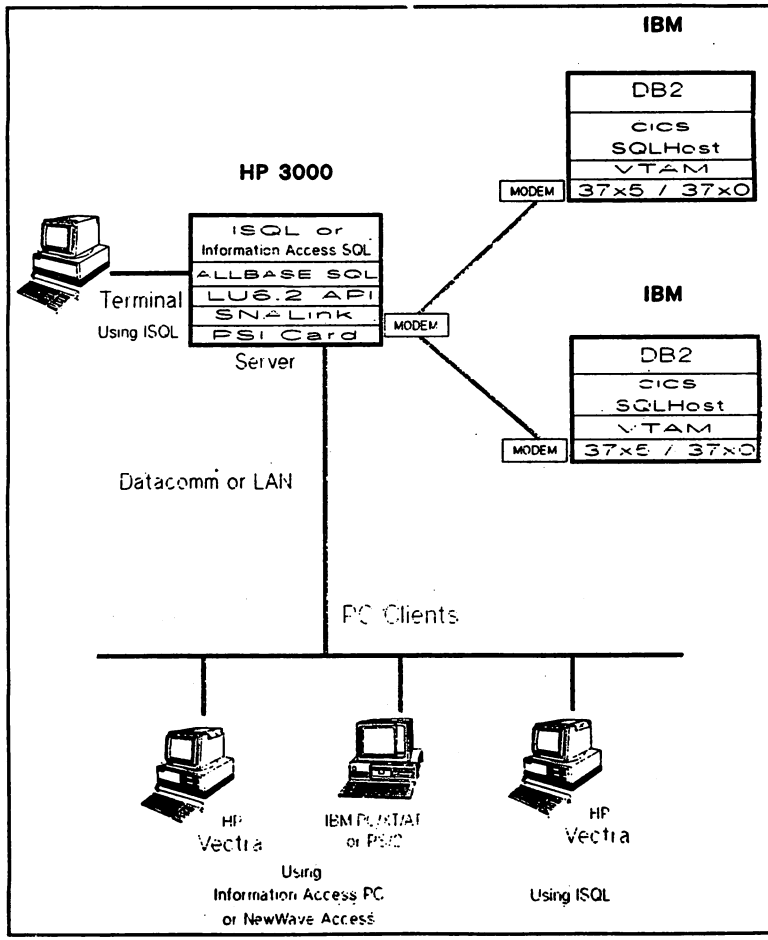
* DB2 data can be accessed using products that support ALLBASE in a manner that is consistent with accessing ALLBASE/SQL

* Data may be exchanged with DB2 via unload or load sequences during off load periods

* Data translation is performed in a transparent manner

* No knowledge of IBM MVS JCL is required for data exchange

The following figure shows one of the many possible client-server relationships with ALLBASE/DB2 CONNECT. In this case, an IBM mainframe is the server for DB2 data and the HP3000 is a client to the DB2 server. The HP3000 is also a server for the PC Clients. More than one DB2 server can be accessed.


IBM
DB2
CICS
SQLHost
VTAM
37x5 / 37x0
HP 3000
MODEM
ISQL or
Information Access SQL
ALLBASE SQL
LU6.2 API
SNALink
PSI Card
Terminal
Using ISQL
MODEM
Server
IBM
DB2
CICS
SQLHost
VTAM
37x5 / 37x0
MODEM
Datacomm or LAN
PC Clients
HP
Vectra
IBM PC/XT/AT
or PS/2
HP
Vectra
Using
Information Access PC
or NewWave Access
Using ISQL

1. INTRODUCTION

As Open Systems and Database Standards continue to evolve, more emphasis is placed on the ability of customer systems to not only meet these standards, but to also share data among applications that span the more frequently encountered environments. ALLBASE/DB2 CONNECT extends HP's connectivity to include heterogeneous systems.

In this environment, the DB2 Data Base Administrator continues to control access to the DB2 data. HP connecting users can be assigned surrogate DB2 userids, and the normal DB2 GRANT and REVOKE commands can be used to control access.

The DB2 connection is configured using installation and configuration techniques that are similar to the methods used with ALLBASE/NET. Standard SNA LU6.2 connections are configured between the IBM host and the HP system using techniques that are familiar to ALLBASE/NET users. Users connect to DB2 in a manner that is similar to the method used to interconnect ALLBASE XL to Unix Database Environments. These connections are available to authorized ALLBASE users.

2. DB2 and ALLBASE - Two Relational Worlds

In a general way, ALLBASE and DB2 are two implementations of the same relational model. Some differences exist in the SQL language used for each implementation, but both languages are moving toward compliance with an evolving standard. ALLBASE is approaching compliance with the ANSI SQL standard.

When users access DB2 using ALLBASE/DB2 CONNECT, they will be using an ALLBASE data access tool. Currently available tools include ISQL, Information Access and NewWave Access.

ALLBASE/DB2 CONNECT does not change SQL statements obtained from the connecting tools. DB2 users or tools must enter SQL statements applicable to the DB2 release involved with this connection. Error messages produced by DB2 are normally displayed after the offending statement by the connecting tool.

While this connection provides for transparent data access, some differences may be observed. A few differences exist in the Data Manipulation Language statements. A few differences also exist in the range and significance of floating point numbers. Some differences also exist in the Data Definition Language statements. More differences exist when the system catalogs are compared.

ALLBASE/DB2 CONNECT supports dynamically prepared SQL statements only.

2.1 Transferring Data Types

When a user selects data from a DB2 table, the data is displayed in ALLBASE/SQL data format. When ALLBASE/DB2 CONNECT selects the data from DB2, some of the IBM data may be converted to ALLBASE/SQL data types. For instance, the date and time formats for DB2 data types differ from the ALLBASE/SQL data type formats for date and time. Also, some IBM data types are not fully supported, e.g., GRAPHIC and VARGRAPHIC.

The limits on the data types differ between the two systems in some cases. The ALLBASE/SQL and DB2 data types and their conversions are described below:

ALLBASE/SQL	Comments	DB2	Comments
INTEGER		INTEGER	
INTEGER		SMALLINT	
FLOAT	defaults to 8 bytes	FLOAT	Defaults to 8 byte float. Differences exist.
DECIMAL		DECIMAL	
CHAR	1 to 3996	CHAR	1 to 254.
VARCHAR		VARCHAR	Depends on page size
TIME		TIME	
DATE		DATE	
DATETIME		TIMESTAMP	
CHAR		GRAPHIC	
CHAR		VARGRAPHIC	
CHAR		LONG VARGRAPHIC	
REAL		REAL	4 byte float

Some DB2 commands are the same as ALLBASE SQL commands. However, you cannot use ALLBASE SQL commands that are not supported in DB2. The following table shows SQL commands (either static or ALLBASE SQL commands) that cannot be used with ALLBASE/DB2 CONNECT:

Command	Command
ADD	PREPARE
BEGIN	REFETCH
CHECKPOINT	REMOVE
CLOSE	RESET
DECLARE	SAVEPOINT
DESCRIBE	SELECT INTO
END	SQLEXPLAIN
EXECUTE	START
FETCH	STOP
INCLUDE	TERMINATE
OPEN	TRANSFER
	WHENEVER

2.2 DB2 with ALLBASE ISQL

When using ISQL, some restrictions apply. Certain ISQL commands are not enabled. These commands include:

INSTALL	This command is used with static SQL.
LIST SET	This command will not display the OWNER setting.
UNLOAD INTERNAL	This command sets the owner name to the value entered by the user. OWNER.MODULE is not defined by DB2.
LOAD INTERNAL	The internal format is only recognizable by ALLBASE/SQL. DB2 users can use the external load format.

The ISQL command LOAD EXTERNAL loads data from an ASCII file into the DB2 database. ALLBASE/DB2 CONNECT converts these rows in the file into DB2 format and adds them to the DB2 table named in the LOAD command. The ISQL command UNLOAD EXTERNAL unloads data from a DB2 database into an ASCII file.

To upload data from ALLBASE/SQL to DB2, a user can specify both the UNLOAD and LOAD EXTERNAL commands. The user can issue the UNLOAD EXTERNAL command to unload ALLBASE/SQL data to an ASCII file. The user can then issue the LOAD EXTERNAL command to load this data into DB2. These commands can be time initiated.

2.3 DB2 with NewWave Access or Information Access

Access to DB2 data can be made from a PC using Information Access PC or NewWave Access on the PC.

For either application, the ALLBASE/DB2 CONNECT alias name is used whenever a DBEnvironment name is requested. In most cases, the PC user will not notice any difference between the DBEnvironment name and the alias name.

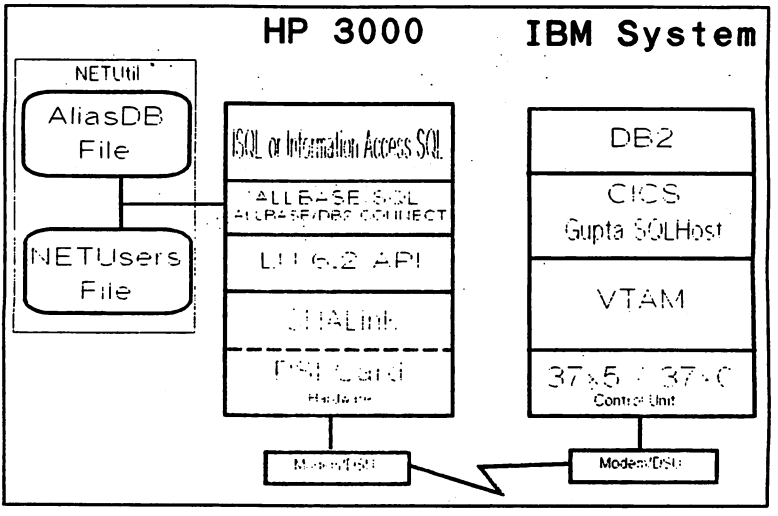
A view and table are needed in DB2 when using Information Access or NewWave Access in order to make the system tables compatible. Information Access and NewWave Access create a list of available tables when an initial connection is made. Options can be set for DB2 connections which limit the number of tables that will be extracted from DB2 when the initial connection is established.

NewWave Agent scripts are supported. These scripts can be used to combineALLBASE data and DB2 data with PC applications. Information Access batch files are supported.

3. Setting Up ALLBASE/DB2 CONNECT

ALLBASE/DB2 CONNECT can be used in conjunction with hardware and software on the IBM system, the HP 3000, and PCs. The IBM system becomes a server, the HP 3000 becomes both a client to the IBM system and optionally, a server to the PCs.

The following figure illustrates the hardware and software requirements for connecting the HP 3000 to DB2.



For the HP3000, ALLBASE/DB2 CONNECT uses the LU6.2 API, SNALink, the PSI card and either a Modem or a DSU to connect to the IBM system. On the IBM system, another modem or DSU is needed, a communications port, VTAM, Gupta Technologies' SQLHost, optionally CICS (or a direct connection), and DB2.

Two utility programs enable the connection parameters to be configured. The NMMGR utility is used to configure the LU6.2 API/XL and SNALink software. The NetUtil utility program is used to configure the AliasDB file parameters and the NETUser File parameters. The AliasDB file entry configures information concerning data buffer sizes (for exchange with the SQLHost transaction), transaction name, trace control, etc. The User file entry assigns DB2 userids and passwords to HP users. These userids and passwords can be validated at each connection.

4. Run Time Considerations

ALLBASE/DB2 CONNECT can be used to pass data in the range of 1 to 16 megabytes per hour. The rate achieved will be based on the baud rate of the communications line used. The actual data rate achieved can be improved by compression techniques. and software on the IBM system, the HP 3000, and PCs.

5. Summary

This paper has presented an overall picture of ALLBASE/DB2 CONNECT. Hopefully it has given an insight into how ALLBASE/DB2 CONNECT operates, and how ALLBASE and DB2 data can be combined.

More information concerning the usage, installation, and setup can be found in the individual manuals for ALLBASE/DB2 CONNECT and the required products.

Though the concept of combining data from heterogeneous database systems and hardware is still new, HP has made a step forward by extending the integrated ALLBASE environment for the combination of relational data from HP-UX environments to now include IBM's DB2.

TITLE: Coexistence: TurboIMAGE and SQL

AUTHOR: Tad Olsen

Hewlett-Packard Co.

19420 Homestead Ave.
M/S 44MA

Cupertino, CA 95014-9974

(408) 447-4088

FINAL PAPER NOT AVAILABLE, HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

PAPER NO. 3241

3242: The ALLBASE/SQL RDBMS: "Optimized for HP Platforms"

Rajoo Nagar
Hewlett-Packard
Commercial Systems Division
Cupertino, California 95014
(408) 447-6526

INTEREX Conference; August 5-8, 1991; San Diego, CA

ABSTRACT

This paper reviews the direction HP is taking with its relational database, ALLBASE/SQL, the application development alternatives now available with the database, and the recent advancements in the area of case and client-server tools. The paper also discusses performance enhancements and new functionality in ALLBASE/SQL with the latest release of the product, and HP's role in database standards such as the SQL Access Group. This paper is targeted primarily at MIS Managers and technical professionals.

PRODUCT OVERVIEW

HP ALLBASE/SQL is Hewlett-Packard's Relational Database Management System (RDBMS) offering on the HP 3000 Series 900 and HP 9000 computer systems. Tremendous performance improvements and new features have made HP ALLBASE/SQL very competitive with leading third party RDBMS offerings. HP ALLBASE/SQL offers leading on-line transaction processing (OLTP) performance, transparent interoperability with non-HP databases, and superior supportability and data reliability for maximum uptime in mission critical applications.

HP's strong commitment to HP ALLBASE/SQL as its strategic database for HP platforms has translated into significant improvements in both performance and features with every release. Today, HP ALLBASE/SQL offers unparalleled on-line performance on the HP 3000 Series 900 systems, and is also a strong performer on the HP 9000 systems. The RDBMS has been optimized and tuned for HP's operating systems and underlying PA-RISC (Precision Architecture reduced-instruction-set computing) architecture. This is because high OLTP performance can be most effectively and quickly delivered through a close coupling of the database and system software. Users can expect to see this competitive advantage in performance increase over time, as HP more tightly integrates HP ALLBASE/SQL with its operating systems and hardware.

The mid 1991 releases of HP ALLBASE/SQL (MPE-XL 3.0, HP-UX 8.0) incorporate significant enhancements to the database management system. An expected 100% increase in performance (TPC-A) on HP-UX will give HP ALLBASE/SQL a clear performance advantage on the HP 9000 systems. In addition, this latest release of HP ALLBASE/SQL incorporates new features such as server enforced referential integrity, on-line backup, automatic log switching, and large free text storage through BLOB (binary large object) support.

New 4GL application development and client-server tools for the database are discussed below under a separate section.

PRODUCT STRATEGY

The HP ALLBASE/SQL strategy is based on the following goals:

* To provide the fastest performing database engine on PA-RISC systems

* To provide open solutions through:
  - o a choice of leading multi-vendor application development tools
  - o interoperability with other databases in heterogeneous environments
  - o support for leading industry standards

* To provide solutions to enable co-existence with HP TurboIMAGE databases.

* To provide superior supportability and data reliability for mission critical applications

By integrating multi-vendor tools and embracing industry standards like SQL Access, HP ALLBASE/SQL represents an open database system offering users all the advantages of an engine tightly integrated with the operating system, while at the same time providing openness and application portability. The HP ALLBASE/SQL strategy is to provide users with the best of both worlds: high performance, integrity and supportability through tight integration with the operating system, and openness through multi-vendor tools integration, foreign data access and standards.

PERFORMANCE

HP ALLBASE/SQL offers leading on-line performance on HP platforms. This advantage is the direct result of HP's tightly integrating HP ALLBASE/SQL with HP operating systems and hardware, thus allowing maximum use of available CPU power and operating system features. HP's tuning efforts in the HP ALLBASE/SQL database engine and HP operating systems in the last two releases have yielded remarkable performance gains on MPEXL and HP-UX systems, placing HP ALLBASE/SQL in the forefront of relational database performance. TPC-A benchmark tests show a 15%-25% performance gain for MPE-XL 3.0, and a 100% performance improvement for HP-UX 8.0.

In addition to tight integration with the operating system, HP ALLBASE/SQL high performance can be attributed to the following features in the engine:

* Path length tuning
* Raw I/O (on HP-UX systems)
* Cost-based, statistical optimizer
* Hash indexes
* Group commits
* Read uncommitted concurrency option
* Cross-transaction cursors
* Bulk data transfer
* Improved sort algorithms
* Fast inter-process communication between front-end and back-end
* Multi-processor support

New functionality has also been added to the database to provide better on-line application performance with third party application development tools.

The HP ALLBASE/SQL strategy is to continue offering leadership TPC-A performance on HP platforms. (TPC-A is the industry standard benchmark recently adopted by the Transaction Processing Council. Increasingly, vendors are using TPC-A as the yardstick for comparing database performance results.)

MULTI-VENDOR TOOLS INTEGRATION

HP ALLBASE/SQL is compatible with several industry-leading multi-vendor

application development and client-server tools. Applications developed using these toolsets will be able to run on HP ALLBASE/SQL or on other databases supported by the tools with little or no modification.

The separation of application development tools from the database engine is a trend that is beginning to gain momentum in the relational DBMS market. HP has responded by providing linkages of popular multi-vendor 4GLs to HP ALLBASE/SQL. Cognos' Powerhouse, Ingres' Application By Forms (ABF), Information Builders' FOCUS and InfoCentre's Speedware 4GL tools support access to HP ALLBASE/SQL on the HP 3000 and HP 9000 systems.

The Cognos and Ingres application development toolsets provide compatibility across a wide variety of platforms such as HP, DEC, IBM, and Unix. FOCUS is a 4GL application development and report-writing tool dominant in the mainframe market, and Speedware is a high performance 4GL for the HP 3000 systems. HP's strategy is to continue to increase the 4GL solutions available with HP ALLBASE/SQL.

Separating the application development tools from the database engine allows customers to choose the preferred application development environment and database for their information management needs, even if they are not supplied by the same vendor. It also gives customers the flexibility to mix and match front-end toolsets with back-end database engines. Providing a high performance, integrated database engine and flexibility in tools selection differentiates HP ALLBASE/SQL from other relational databases.

In the client-server area, HP is working to provide users with a choice of PC-based client-server 4GLs that access ALLBASE/SQL on the HP 3000 and HP 9000 systems. SQLWindows from Gupta Technologies, and Powerbuilder from PowerSoft will be available in the late 1991 - early 1992 time frame. On HP-UX workstations, the Ingres/Windows 4GL will enable users to build graphical client-server applications that access ALLBASE/SQL on the HP 3000 and HP 9000 servers.

Industry standards organizations such as the SQL Access Group support the trend towards database and tools separation by providing the standard application programming interface (API) and network protocols, so that users can mix and match heterogeneous SQL products in a multi-vendor environment.

INTEROPERABILITY VIA STANDARD INTERFACES

An open system requires the full backing and support of industry leaders to ensure customers the benefits of direct interoperability. The SQL Access Group members include relational database vendors such as as Informix, Ingres, Oracle, Sybase etc. and systems vendors such as DEC, HP, NCR, Sun and Tandem. HP is one of the founding members and an active producer member of the group. The mission of the SQL Access Group is to implement an application programming interface (API) and ISO-based Remote Database Access (RDA) standard that will provide an interoperability solution for customers who own multiple databases, running on different machines, and who wish to link their databases through a standard interface. In the future, users will be free to mix and match SQL Access compliant products to meet their information

management needs. The SQL Access standard interface will be provided with HP ALLBASE/SQL in 1991, allowing interoperability between HP ALLBASE/SQL and other databases that conform to SQL Access.

HP ALLBASE/SQL FEATURES

a. Overview/Unique Features

Based on the industry standard ANSI SQL specification, HP ALLBASE/SQL is a functionally complete relational database that runs in native mode on the MPE-XL and HP-UX platforms.

Briefly, here is an overview of the current HP ALLBASE/SQL offering:

- Price/performance leader on MPE-XL and HP-UX
- Fastest RDBMS on HP platforms (based on audited and fully disclosed TPC-A and TPC-B benchmark data)
- 100% conformant with ANSI SQL Level 1 and X/OPEN XPG3
- Interactive SQL and preprocessors bundled with database
- Full Native Language Support
- Complete set of HP and multi-vendor tools for application development and connectivity
- Backed by HP commitment to quality and reliability
- Worldwide support
- Lower cost of ownership than competing products

From a features point of view, here is what the current HP ALLBASE/SQL product offers on MPE-XL 3.0 and HP-UX 8.0 releases:

- Data access via B-trees indexes and hashing
- Cost-based query optimizer
- Interactive SQL interface
- Referential Integrity (conforms to ANSI SQL1 Addendum)
- Graphics, voice, and free text storage (binary large objects)
- Dynamic SQL
- PC client tools
- NewWave integration
- Multi-vendor, multi-platform 4GL and Query/Reporting tools
- Language preprocesors for C, Cobol, Fortran, Pascal & Ada
- Variety of DBA tools: SQLUtil, SQLMigrate, SQLCheck, and SQLGen
- Automatic deadlock detection and resolution
- Concurrency control: Isolation Levels - Read Committed, Cursor Stability, Repeatable Reads, Read Uncommitted (Dirty Reads)
- Transaction Control: Manual & automatic locking, Savepoints, Checkpoints, Cross-transaction cursors
- Full recovery mechanism: Rollforward recovery, Rollback recovery, Physical image logging
- Group Commits
- High availability options: On-line backup

Automatic log switching
Dual logging capability
Dynamic database restructuring
(column deletes not allowed)
Dynamic space expansion

- Page level locking
- Flexible (DBA-assigned) security scheme
- Null data handling
- Unique "Keep Cursor" capability with LOCK option
- Transparent remote reads/writes across the network
- Support of 8- & 16-bit characters (NLS)
- Bulk data transfers
- Multi-processor support
- Raw I/O on HP-UX

b. User Environment

The current HP ALLBASE/SQL user environment consists of a complete set of HP and third-party tools centered around the HP ALLBASE/SQL database. The interactive SQL (ISQL) interface is based on the ANSI standard SQL, and provides interactive access to the database via DDL and DML commands. 4GL, reporting and query tools are available for application development, creation of reports, and ad-hoc querying of the database. Ad-hoc queries are supported via both a command-line and a menu-driven SQL interface. PC-based client-server 4GL tools facilitate PC-based application development and execution against the server database.

HP's growing list of key third party tools and applications for HP ALLBASE/SQL provides customers an even richer set of solutions for information management.

c. Application Development

Application develoment in the HP ALLBASE/SQL environment is facilitated by the use of 4GL and 3GL application development tools from HP and third parties, and by the interactive SQL (ISQL) interface to the database. Language preprocessors for C, Cobol, Pascal and Fortran allow programmers to access the database from these languages via embedded SQL (application programming interface) calls.

d. Decision Support

HP ALLBASE/SQL provides a platform for decision support when used in conjunction with NewWave Office Information Access. NewWave Access is a PC-based data retrieval tool that accesses and downloads data transparently from an HP ALLBASE/SQL database residing on the host machine. The data can then be viewed through any of the popular PC PC packages such as Lotus 1-2-3, Symphony, Dbase, etc.

Decision support tools from leading third parties are available with HP ALLBASE/SQL. For example, the FOCUS report-writing toolset, Ingres/Graphs and QBF (Query By Forms), and Powerhouse QUIZ all provide powerful decision support capabilities.

e. Remote Data Access

HP ALLBASE/NET allows access to remote HP ALLBASE/SQL databases on systems connected via LAN or X.25; both NS and ARPA protocols are supported. The NET product connects SQL databases to each other and provides both users and applications with transparent read/write capability to remote systems. No special linking is required to create applications that can access a remote database. HP ALLBASE/NET supports transparent remote access to HP ALLBASE/SQL from the ALLBASE and third party tools as well as user-written preprocessed applications. HP ALLBASE/NET supports peer-to-peer networking between databases, as well as client-server connectivity between workstations and minicomputers.

f. Client-server integration

Read access to HP ALLBASE/SQL from the PC is supported today with HP NewWave Access. Program-to-program communication between the the host SQL server and PC client applications over a LAN is under development and will be implemented using HP ALLBASE/SQL as the server database and third party PC-based 4GLs as the client component. Full read/write access to HP ALLBASE/SQL from PC applications will be supported, and this capability is expected to be available in the late 1991 - early 1992 time frame. The third party PC-based 4GLs include SQLWindows from Gupta Technologies, and Powerbuilder from PowerSoft Corp. Synergist is a PC-based client-server 4GL from Gateway Systems that currently interfaces with HP ALLBASE/SQL on MPE-XL systems.

HP will support Cognos Powerhouse and Ingres/Windows 4GL tools on the HP-UX workstation platform, and will support full read/write access from these tools to HP ALLBASE/SQL on MPE-XL and HP-UX servers.

4GL Development Environment

	Traditional	Client-Server	
Application Components	Terminal	Workstation/ X-terminal	PC
4GL	Allbase, Focus, Ingres, Powerhouse Speedware	Ingres Powerhouse	Synergist SQL Windows Powerbuilder
Database	ALLBASE/SQL	ALLBASE/SQL	ALLBASE/SQL
Operating System	MPE-XL/HP-UX	MPE-XL/HP-UX	MPE-XL/HP-UX
User Interface	4GLs available will accomodate a variety of display devices and GUIs (MS/Windows, OSF/Motif, PM, etc.)		

g. Connectivity

Transparent connectivity between HP ALLBASE/SQL databases on MPE-XL and HP-UX systems is provided through the ALLBASE/NET product. Thus applications can be developed which share common source across the HP 3000 and HP 9000 families.

Connectivity between HP NewWave Office Information Access on the PC, and HP ALLBASE/SQL on either MPE-XL or HP-UX is supported through Lan Manager and serial connection in a NewWave Office server environment.

HP ALLBASE/Turbo CONNECT provides transparent read access to HP TurboIMAGE database from any HP ALLBASE/SQL interface. The product allows customers to begin new application development using a relational database as the platform, while still being able to read HP TurboIMAGE data. By providing a link between the two databases, and a common user interface and common tools, HP has made co-existence between HP ALLBASE/SQL and HP TurboIMAGE much smoother and easier.

Read/write transparent connectivity to IBM's DB2 database from HP ALLBASE/SQL application level calls is available with the MPE-XL 3.0 release. HP ALLBASE/DB2 Connect allows DBAs, application developers, and decision support users running on an HP 3000 MPE-XL system to interactively create, read, modify, and update information in a DB2 database on an IBM MVS mainframe.

Interoperability with other relational databases through the SQL Access standard interface is expected to be available in the 1991 -1992 time frame.

h. HP TurboIMAGE - HP ALLBASE/SQL coexistence

HP ALLBASE/SQL is an ideal solution for customers already using HP TurboIMAGE but wanting to take advantage of relational technology. HP ALLBASE/Turbo Connect lets these customers' applications co-exist by allowing HP ALLBASE/SQL applications to read HP TurboIMAGE data. HP TurboIMAGE customers can begin new application development using any of several 3GL or 4GL tools that access both HP TurboIMAGE and HP ALLBASE/SQL databases. Moreover, HP's strategy is to provide a set of migration tools and consulting services for customers moving between these environments.

i. Data Integrity

HP ALLBASE/SQL has many mechanisms to preserve the integrity of customer data. Referential integrity, security controls, concurrency controls and recovery mechanisms are a few of these, and are discussed below.

1. Referential Integrity. HP ALLBASE/SQL supports referential integrity checks in the database. Integrity constraints allow the user to check data integrity at the schema level, rather than coding complex checks in application programs. In addition

to simplifying the work of coding, this leads to improved performance. Referential integrity in HP ALLBASE/SQL is implemented using primary and foreign key constraints, and conforms to ANSI SQL1 Level 1 Addendum.

2. Security. The database allows read and write access privileges to be assigned at the table level. Access restriction at the column level may be obtained by defining a view of the table which omits the sensitive information. Modification authority may be granted at the column level without requiring that a view be specified. Write access supports a combination of row modifications and deletes. Appropriate levels of access privileges are specified by the DBA for individual users or groups of users.

   Views can also be used to improve security by allowing users to access only that data for which they have a need. Since the view is not actually a physical table, the use of views does not result in redundant data.

3. Concurrency Control. This is provided by locking at the database, table, and page level. Three kinds of explicit locks are provided: An exclusive lock, which prevents other users from accessing the entity and allows the entity to be updated. A shared lock, which allows other users to read, but not later update the referenced entity. And a share-subexclusive lock, that allows users to alter part of a table and exclude others from altering the table, but allowing others to read the unaltered portions of a table. The database also provides intent update locking during read operations. Intent update locking is used when a read operation may be followed by an update for the read page. This type of locking is used to avoid the potential for deadlocks when two users try to upgrade their shared locks to exclusive on the same page.

   The database also provides a set of four isolation levels for controlling concurrency and throughput. These are Read Committed (RC), Read Uncommitted (RU), Cursor Stability (CS), and Repeatable Reads (RR). Any one of these can be selected as an option with the BEGIN WORK command. The default is Repeatable Reads.

   In addition, Keep Cursor, a special extension to the concept of a cursor, allows a cursor to span multiple transactions. This capability improves performance by allowing an application to commit or roll back a transaction and still keep a cursor open.

4. Recovery. A full recovery mechanism is provided to protect data integrity in the event of hardware and software failures: Rollback recovery, Rollforward recovery, and physical image logging. Savepoint and Checkpoint features are used to control recovery. Savepoints allow users to undo changes within a transaction upto the specified savepoint. Checkpoints are

used by the DBA to flush the buffers to disk when the log becomes full. If archive logging is not being used, checkpoints free up log space, and shorten the time to recovery in the event of a system crash.

j. High Availability

1. On-line backup. On-line backup is a process whereby a database backup takes place without bringing down the database system. At a later stage, a log file can be applied to the archived database copy, bringing the database to a consistent state. On-line backup provides nearly continuous access to HP ALLBASE/SQL data.

2. Automatic log switching. With the automatic log switch enhancement, switching to a new log is done automatically while backup is in progress.

3. Dual Log is another mechanism for ensuring data availability. The dual log option, if enabled by the DBA, results in the second log being automatically invoked by the database if the first log becomes damaged.

4. Dynamic restructuring allows the database structure, table capacities, and security to be changed without unloading and loading the database, thus improving database availability for users.

5. Dynamic Space Expansion allows DBEFile space to be expanded on-line, without having to bring down the database, and this capability significantly improves availability.

k. Large text storage/Imaging

With the latest release, HP ALLBASE/SQL supports long, binary data-types. This allows users to store very large, variable data (unlimited size columns) in their binary format. This is useful for storage of non-character data, such as graphic images or voice, without the side-effects of CHARACTER interpretation.

l. Localization

HP ALLBASE/SQL lets users manipulate databases in a wide variety of native languages. Either 8-bit or 16-bit character data can be used, as appropriate for the language selected. Truncation is performed correctly for 8 and 16-bit character data. The database will display prompts, messages and banners in the language selected, and it displays system dates and times according to the local customs. In addition, the database accepts responses to its prompts in the native language selected.

m. DB Administration

A variety of database administration tools are included with the database system, and can be used by the DBA to manage the database environment, and to facilitate database integrity verification and migration.

SQLUtil	DBE configuration (alteration) utilities
SQLGen	Schema and load generation tool
SQLCheck *	Integrity checking tool
SQLMigrate	Transparent migration from previous releases of HP ALLBASE/SQL
SQLMonitor **	Performance analysis tool

* SE tool. ** SE tool, available as a product in the next release.

STANDARDS CONFORMANCE

The two most widely recognized organizations which help shape SQL standards are ANSI SQL and X/OPEN. ANSI SQL is driven by the participating vendors and industry researchers in the US. ISO is the international counterpart of ANSI SQL that drives the European market. X/OPEN is a defacto standard that influences the UNIX-based SQL products. Another emerging standard is the SQL Access Group, which is implementing a remote database access standard for database interoperability in heterogeneous environments.

HP ALLBASE/SQL utilizes ANSI SQL for data definition language (DDL) and data manipulation language (DML) operations. Today, the product conforms 100% to ANSI SQL Level 1 and X/OPEN XPG3 standards, and almost fully complies with ANSI SQL Level 2. The SQL Level 1 Addendum that defines referential integrity is satisfied with the latest release of HP ALLBASE/SQL. The SQL Access standard interface is still being defined, and is expected to be available with HP ALLBASE/SQL in 1991.

HP uses industry standard benchmarks such as TPC-A and TPC-B to publish the performance of HP ALLBASE/SQL on the HP 3000 and HP 9000 systems.

PRICING/PACKAGING

The HP ALLBASE/SQL development package includes the core RDBMS, the interactive SQL (ISQL) product, the DB administration utilities, and the preprocessors for C, Cobol, Fortran and Pascal. The SQL Run-time product includes all the products in the development package except the preprocessors.

On the HP 3000 Series 900 systems, the HP ALLBASE/SQL development package is bundled as a preconfigured "Add SQL" option. The 4GL tools and connectivity products are purchased separately.

On the HP 9000 systems, HP ALLBASE/SQL is available both as a run-time and development package, and the price ranges from $2,050 to $109,960. Again, the 4GL tools and connectivity products are purchased separately.

STRATEGIC ALLIANCES (VABs/ISVs)

The following VABs are among those who have committed to HP ALLBASE/SQL as of Fall 1990. Many of these applications will be available on HP platforms in the next six to twelve months.

o	ASK Computer Systems	Manufacturing
o	BSA, Inc.	Direct Marketing
o	Cevan, Holland	Local Government
o	Collier Jackson	Accounting and Financials
o	Computrac	Legal Industry, Accounting
o	DPAI	Financial Management & Manufacturing Control
o	DRC	Wholesale Distribution
o	Financial Data Planning	Insurance and Pension Administration
o	Hilco	Monitor and Control System
o	IMAC	End User Reporting Tools
o	InfoCenter	Travel, Library and others
o	Jobscope	Manufacturing
o	MDSS	Manufacturing Decision Support
o	Mitchell Humphrey	Accounting and Financials
o	People Soft	Human Resource Management
o	Q-CIM	Manufacturing
o	RMS	Retail Distribution & Financials
o	SATCOM	Wholesale Distribution and Process Manufacturing
o	Western Data Systems	Manufacturing, Accounting, and Contract Management

The following independent software vendors (ISVs) have committed to integrating their tools with HP ALLBASE/SQL:

*	Cognos	Powerhouse 4GL package
*	Gateway	Synergist 4GL
*	InfoCenter	Speedware 4GL
*	Information Builders	Focus 4GL
*	Ingres	4GL, Query and Case tools
*	Gupta Technologies	SQLWindows
*	PowerSoft Corp.	Powerbuilder
*	CGI	Paclan
*	SoftLab	Maestro II

These tools will be available on both HP-UX and MPE-XL platforms, with the exception of Speedware 4GL that will initially be available on MPE-XL only.

FUTURE DIRECTIONS

The following new functionality and products are expected to be available with future releases of HP ALLBASE/SQL in the 1992 time frame:

- Stored procedures
- Business rules/triggers
- Transaction Processing Monitor (based on X/OPEN) support
- Row level locking

- Interoperability with other RDBMSs via SQL Access interface.
- Performance analysis tool
- New DBA tools

Distributed databases:

The distributed ALLBASE product is currently under development and is expected to be introduced in late 1991 - early 1992 time frame. Some of the features that are expected to be introduced in the first release are 2-phase commit, distributed transaction management, and location transparency of distributed databases.

SUMMARY

Specific opportunities for HP ALLBASE/SQL include customer environments that place a premium on:

* Leadership price/performance and performance
* Broad selection of multi-vendor application development tools
* Tight integration with HP products and support
* HP quality, reliability and service

3243

MPE XL Development in a Multi-Platform Environment

Beth Eikenbary
19447 Pruneridge Avenue 47UP
Cupertino, CA 95014
(408) 447-6146

In the past, application developers placed most of their emphasis on optimizing development for a specific hardware platform. This was a natural consequence of the proprietary operating system mentality that has permeated the computer industry in the past. Today, as a result of the Open Systems movement, application developers are now focusing on optimizing the portability of an application across hardware platforms. This paper will first look at some of the key ingredients in developing portable applications, review the porting process and then discuss how to effectively manage mutli-platform development projects within your organization.

Before jumping into the porting process, I want to provide a few key definitions. The first area of confusion is the usage of porting versus portability. Porting is a process by which an application is moved from one execution platform to another. Portability on the other hand is a measure of how well a particular application behaves during any one port. Portability variables are hardware, operating system, language and database.

There are two basic types of ports: cookie-cutter and custom. A cookie-cutter port assumes minimal changes to the application source when moving to a new platform. The benefit of a cookie-cutter port is its low-cost combined with a high degree of compatibility across the platforms. On the other hand, custom ports result in higher performance and extra features at a higher cost in porting and maintenance. Before embarking on development of a new application, it is important to determine both the type of port desired as well as the expected domain of portability. The expected domain of portability is a definition of the specific environments, including hardware, operating systems and networks, where you think the application software will need to execute. Where the type of port, cookie-cutter versus custom, dictates the degree of standardization required in your application, the domain of portability identifies those standards which will be most critical to you in your potential porting efforts.

This seems to be a logical point in discussing porting and portability to look at the role of standards in application

development. Standards are critical in any application development project where application portability is required. The rationale behind using standards is to increase programmer productivity by:

- -reducing the amount of specialized code
- -reducing the number of variants in solving a specific problem area
- -providing a consistent framework for the development team
- -increasing programmer portability by allowing them to focus on application technology, not a specific computer platform.

However, standards are not the panacea that the computer vendors want us to believe. An Open System certainly goes a long way toward ensuring a consistent interface to an operating system or networking software, but it does not eliminate all of your portability problems. Despite the best intentions of the standards bodies and the computer vendors, standards vary slightly from platform to platform. This is due in part to the fact that standards reflect old technical concepts on top of perpetually evolving and enhancing technologies; computer vendors often sacrifice 100% conformance to a standard in order to allow new technologies to emerge. In addition to the problem of compatible interfaces, the use of standards may result in a performance hit. Performance is normally a function of optimizing the unique attributes of the hardware; standards can often make it impossible for a vendor to optimize performance for their underlying hardware given the constraints of the standard definition.

Given the potential problems with the adoption of standards, it is important to pick and choose those standards which you are going to adopt. As a rule of thumb, don't enforce a standard unless required by your selected domain of portability. For example, a database intensive OLTP application should focus on adopting standards in the area of user interface, data base interfaces and the file system. It is not critical to require adoption of standards for system administration or tools but a networking standard should also be considered. As a rule it is also better to try to use standards which are self-enforcing. Application development is still part craft and, as such, the developers still gain a great deal of pride through the uniquely solving development problems. Whenever possible, invest in the development of tools and process whenever you can to enforce compliance. Developers will usually choose the path of least resistance.

As we just discussed, using standards is a key means for ensuring that your application will be portable. When considering the role of standards in your application

development process, consider not only formal, industry standards such as the POSIX interfaces, but also the creation and enforcement of internal coding standards. In many ways internal coding standards can have as big an impact on the portability of an application as the use of formal standards. Internal standards which should be considered are:

-style standards for naming and formatting
-structure standards for isolation of non-portable code
-testing standards (remember, the test suites will need to be ported as well as the application)
-interface standards which will allow for consistent end user interaction across input devices (workstations, personal computers and/or terminals).
-documentation standards to allow for easy porting

Code scanners which check for code consistency and standards adherence provide significant productivity advantages during core application development. In the area of code scanners, one product stands out as the premier solution, Mclint. This tool scans code for adherence to various standards, such as POSIX or STDC. It can also be enhanced to scan code for adherence to those internal standards you have also established. Use of this tool, or at a minimum the standard UNIX 'lint' command, can save you significant time during the porting process.

The choice of language will also have a significant impact on your ability to hide variation within the source. C is the acknowledge language of choice when porting due to its flexibility. With the adoption of ASCII C by the major vendors, problems with variations of C libraries across platforms has been significantly reduced. Within your C programs, variation can be hidden easily by the use of include files and functions. The C Preprocessor itself is a powerful tool for easing porting as well. The C Preprocessor provides for file inclusion, definition of manifest constants, definition of "call-by-name" functions (macros) and conditional compilation. One note of warning: C is a very flexible language and inherent in this flexibility is the ability to easily misuse the features and functions. It is important to establish standards around the use of include files, manifest constants and macros to ensure clean, high quality code.

Ensuring application portability is only half the battle; the other half of the battle is actuality porting the application across multiple platforms. An overview description of a porting process follows.

Most companies choose organizationally to separate application development and enhancement from the porting process. This is due primarily to the dramatically different

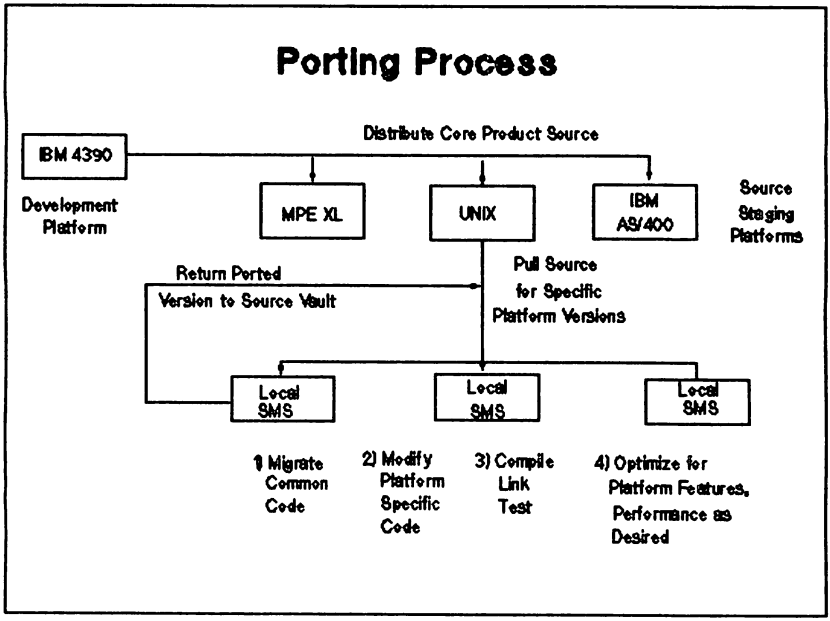
skill sets required from the engineers. In application development the emphasis is on solving the business problem with the actual implementation a secondary activity. Porting, on the other hand is a technical activity requiring very little understanding of the business problem the application solves. Another way of looking at the difference is that application development focuses on the product externals whereas porting focuses on the product internals.

A key development platform is chosen to support the initial application development. Choice of this platform is either historical: this is the platform the application was originally designed to execute upon; or based upon the availability of development and programming tools/utilities. In many companies, porting of existing applications is deemed to be unfeasible because the application itself is not portable. This does not mean that an organization has to completely re-train their programming staff on a new platform just to be able to develop portable applications. But many organization when faced with a major application re-write choose to move to a more open development platform such as a workstation or personal computer to achieve higher programming productivity. The trade-off of re-training of the development teams versus increased productivity is one each organization needs to evaluate for themselves.

Once an application is completed, the source is moved from the development platform to Source Staging Platforms. Source Staging Platforms are those platforms to which you want to port your application. In the example depicted in the following diagram, an IBM 4390 has been chosen as the Development Platform with MPE XL, UNIX and IBM AS/400 timeshares as Source Staging Platforms. A version of the application source then is moved to each of these Platforms. A team trained in the nuances of the target execution platform is assigned to port the actual application and its accompanying test suites.

The first step in a port is to get the application running on the target platform; the second step, which is optional, is to then optimize for features, functions and performance provided by a specific platform. Once the application port is completed, the source is then merged back into the Development Platform Source Management System (SMS).

Subsequent releases from the Development group also have to be ported. Depending upon the complexity of the subsequent release, the porting team may be required to completely re-port the application or merely port changed modules. Having all source under a single SMS will give you the data required to determine how to go about porting subsequent releases. The following diagram provides an overview of the porting process.



There are several key tools required to suport the porting activity. Obviously, the most critical is the Source Management and Configuration Management Systems. These two tools are critical for containing and managing the porting effort. Standard application build and integration tools are also very important. These tools make it possible to easily reconstructs the application on the target platforms. Of course, the choice of application development environment will also have a significant impact on the productivity of both your development and porting teams. More indepth discussion of each of the three tools areas follows.

First: source code control is a key to porting and supporting an application across multiple platforms. Basically, porting results in a development environment characterized by multiple applications, probably multiple versions of each of these applications compounded across multiple target systems. One can easily see that effectively managing source versions will be a key to your ability to productively support your application development teams and your end users.

A Source Management System (SMS) allows each of the development and porting groups to proceed with comprehensive

tracking of working versions, easy reconstruction of previous versions, and co-ordinating simultaneous development projects while maintaining programmer accountability. The SMS you select should allow for efficient storage of multiple versions. Although a SMS goes a long way in managing your source, you also need to development an ancillary set of tools to further improve programmer productivity. These types of tools include a application reminder system which informs others of changes to various source modules or searches for files checked out for long periods of times and nags the programmer. These tools are specific to an organization and the products it is working with. Their need should grow out of the process of automating your development and porting activities.

Simple reconfiguration also makes the porting of an application easier. There are many methods available and as with the Source Management System and ancillary tools, your standard approaches will grow out the unique needs of your organization, application product and selected platforms. As a starting point, configuration change methodologies should cover compile, link and run time changes. At compile time, configuration management needs to consider conditional compilations as well as configuration of header files. Linking features include the selection of optional features and optimizations. At run time, configuration management focuses on environment variables such as resource mapping or location of files. The configuration management system should also be a repository for documenting the what, why and how of each unique configuration. This key information is often lost during development and porting.

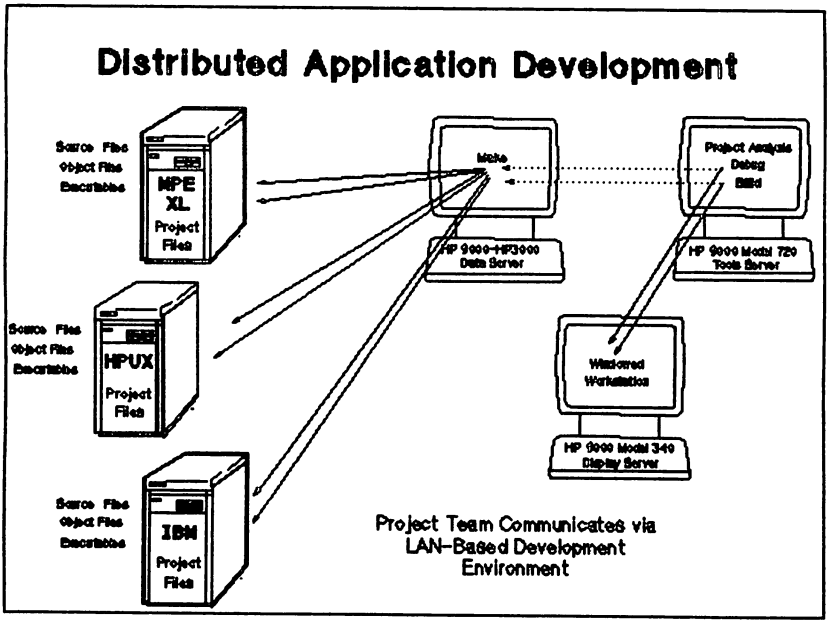
Secondly: the build tools simplify the compile and link phases of the typical edit/compile/link/debug loop used during application development. It shortens this part of the cycle by compiling only those parts of an application that require recompilation because of direct or indirect changes. In addition, the build tool should also simplify the task of building and maintaining the application by creating and maintaining the dependency control files required for intelligent builds. This build tool needs to be flexible enough to build a standalone portion of the application and powerful enough to build an entire version based upon information in the Source Management System and configuration management system. Since your build tool will need to execute across all of your target platforms, the tool itself needs to be portable as well.

Lastly: the application development environment which you select should be designed to facilitate rapid, interactive program development, test and maintenance in a distributed network. The concept of working within a distributed network is key when considering development of portable applications.

A application development environment which facilitates development within a network can centralize many of the porting tasks, such as source control, but still provide for local compilation, test and execution on the target platforms. When looking for a productive application development environment, the key criteria for selecting a distributed development environment is the degree to which the network configuration is hidden from the programmer.

For example, data should be able to reside on any platform in the network. A developer working at a workstation should be able to easily access data on any of the target platforms. With large application development/porting projects, it is more effective to manage and administer data centrally than to have the data duplicated on each workstation in the network. For example, source control, tape backup and archiving as well as configuration management are easier when the project files are centralized.

The following diagram is an example of a possible distributed development environment for the porting task described earlier. In this development environment, a separate LAN is established for the application development and porting teams. Separate LANs ensure close team communication without an overload of inappropriate messages due to cross-team



MPE XL Development in a Multi-Platform Environment

message traffic. Although each team works independently through their established networks, they are all tied to the same timeshare network of HP3000s, HP9000s and IBM systems.

In this example, each member of the engineering team has a small, inexpensive X display machine (possibly a diskless HP workstation or an X terminal). The tool server would typically resides on the LAN hub machine. The new HP9000 Model 720 provides an excellent hub for distributed application development by providing the robust UNIX development environment with the processing power required to support a team of programmers. The HP9000 Model 720 is then tied to a series of timeshares, including the HP9000 Model 800 and an HP3000 Model 900. These timeshares are used as data storage facilities for the applications targeted for execution on that platform. A similar set of timeshare would need to be established for the IBM execution platforms as well.

As you can see a development environment which supports distributed application development provides many advantages to those who are focusing on the development of portable applications. The greatest benefit from this type of environment is in the standardization of the environment across all of the platforms. Since the same basic set of tools is supported through the same development environment, porting teams do not need to learn a new set of development tools for each of the target execution platforms. As a new port in undertaken, the target platform is merely added to the network as a data server. The development team is still utilizing the same tools. Only those tools which control that actual application construction need to be modified to support the new platform.

Other criteria to consider when selecting a application development environment which supports both application development and porting are:

-Leverage existing tools: You will want to integrated those tools which you develop internally to automate your internal porting process. The application development environment should easily accommodate integration of internal tools with those supplied with by the vendor.

-Support integrated tool sets: The tools should cooperate to present a task-oriented environment that lets users concentrate on what they want to do, not how to do it.

-Support interchangeable tools: Application development as mentioned before presents a very different problem to the programmer than porting.

Different CASE tools will be used for core application development and maintenance than for the porting process. However, all tools will need to communicate effectively with the data server. The interchangability of tools allows each team to select the tools which best meet their needs while ensuring effective cross team communication.

-Support application development teams: The tools and application development environment should support team coordination and the management of project files in a distributed development environment. Automated communication between team members is a definite requirement in the porting environment as identified earlier.

-Build on standards: You are not only concerned with the portability of your applications but also with the portability of your application development environment; you will be able to move your developent activities to new platforms without facing re-training your application development teams.

Selection of your application development environment can have a dramatic effect on the productivity of the application development team today and in the future. When selecting an application development environment, it is important to consider both the needs of the application developer and the porting specialist.

In summary, this paper has outlined some of the key requirements for ensuring application portability during initial development through the adoption of industry standards and the creation of internal, portability standards. The selection of which standards are required for your anticipated application execution platforms was also discussed prior to reviewing the porting process.

A standard porting process was reviewed. This process separated the development process from the porting process. In reviewing the porting process, the criticality of a good Source Management System and Configuration Management tools became evident. In addition to these two tools, the need for sophisticated application integration and build tools was also discussed. The selection of application development environment which supported distributed application development was seen as the last step in preparing for multi-platform development. The application development environment is the point in which all of the pieces of the problem of developing and porting applications come together. This environment determines the degree of productivity the application developers in your organization will achieve.

Paper # 3244

THE INS AND OUTS OF DATABASE DESIGN

by Lynn Barnes
Hewlett Packard Co.
(301) 258-2112

Introduction

In the mid 1960's database technology began replacing file systems and a new era of information management began. Over the years as the size of most organizations grew so did their databases. With this size increase has come an increasing realization of the need for good database design. With small databases, design is generally not complicated. However, with medium to large databases -- with 30 to hundreds of users, executing multiple application programs, doing hundreds of queries against, many megabytes of data -- database design becomes far more complex. Efficient and effective database design is essential to today's organizations which rely heavily on their information systems.

This paper will take the reader through the major steps of database design. It will discuss the goals of database design and its five phases: Requirements gathering, conceptual design, logical design, physical design, and implementation. After reading this paper, the reader should understand the importance and methodology of good database design.

Goals of a database designer are to satisfy the users' and applications' information requirements while providing a natural and easily understood information structure. Additionally, the designer must meet the processing requirements, storage requirements, and performance objectives. Unfortunately, these goals are hard to accomplish or even measure.

With the above design goals in mind, an effective database should be:

- Shareable among multiple applications.
- Flexible enough to support changes in process.

- Streamlined with minimal redundant data.
- Designed to accommodate the anticipated growth of the organization.
- Complete.
- Easily understood by both end users and data processing personnel. This includes database structure, naming conventions, and data definitions.

Major Stages

Over the years database design has slowly evolved from an art to a science. Structured database design is now a well-defined process with its major stages being:

- Requirements gathering and analysis
- Conceptual database design
- Logical database design
- Physical database design
- Database implementation

During these phases the designer must look at both the data content and structure as well as the database processing and software application. Traditionally, database design was attempted with a primary focus on one or the other of these design approaches but rarely were these two activities emphasized equally. It is now recognized that data content and structure design (known as data-driven design) and database processing and software application design (also known as process-driven design) must proceed together with very tight coupling to achieve a good overall database design.

Requirements Gathering and Analysis

Requirements gathering and analysis is very important to effective database design. The designer must understand the

expectations of the users and the intended use of the database. The first step to defining the requirements is to identify all users and application which will interact with the database and determe what their individual and global requirements are. The following activities can be performed during requirements gathering:

- o Review any existing documentation or previously written requirements analysis.
- o Identify users and applications that will use the database.
- o Identify processing requirements:
  - transaction type, frequency, and volumes
  - interaction between the transactions and the data object
  - information flow
  - data input and output
  - storage requirements
  - hardware and software platform
- o Interview key users from all groups to determine:
  - users' goals and expectations
  - users' priorities
  - performance, integrity, security, or administrative constraints
  - key applications and application interactions
  - application and user growth plans

Once the information is gathered, the designer should begin the analysis and generation of a global data model. The requirements and global data model is then usually transformed into a formal requirements specification with text, tables, diagrams, and/or charts. It is important that sufficient time be allocated to requirements gathering and analysis because it is crucial for the future success of the database system.

Conceptual Database Design

The second phase of database design is to develop the conceptual schema. During this phase the data requirements

determined in the first phase are used to produce a conceptual data model. Transaction design should begin in parallel with conceptual data modeling. Transaction information plays a crucial role in the physical design phase.

Conceptual data modeling

The conceptual data model is a high-level data model which is usually independent of the database management system (DBMS) to be used. The conceptual data model documents the users view of the data, incorporats the policies of the organization, and shows the relationships between the different data.

The conceptual model consists of three main components: Entities, attributes, and relationships. An **entity** is a "thing" about which an organization collects data. An **attribute** provides information about an entity. A **relationship** describes how each entity relates to another.

An entity is a noun. It is a person, place, thing, event, or concept; such as EMPLOYEE, STATE, PARTS, ORDER, and DEPARTMENT. Entities in the final conceptual data model should be fundamentally important to the entire organization. The database designer must be able to differentiate between what is important to the organization or an individual user. Usually an organization will already be collecting data about information that is important to it. Existing reports and forms give good leads to potential entities.

An attribute can be thought of as an adjective which describes or qualifies the entity. For example, the values for Name, Address, and HireDate could be attributes used to describe the entity EMPLOYEE. All entities have a set of attributes which describe the entity. Among this set of attributes, there must be an attribute (or combination of attributes) which uniquely identifies each occurrence of data within the entity. This attribute(s) is called the unique identifier for that entity.

A relationship is the verb which defines the association between two entities. Works_for might be a relationship between the entities EMPLOYEE and DEPARTMENT. Relationships are stated in terms of action and define the rules and policies of the organization.

Although it is generally easy to define entities, attributes, and relationships, it is sometimes difficult to distinguish their roles in the data model. Should City be an entity or an attribute? Should Orders be an entity or a relationship? The designer must decide whether the user organization needs to collect information about the item or if it is a piece of information about another entity. The following guidelines can be used by the designer to help differentiate between these constructs:

- Entities should contain descriptive information. If there is descriptive information about an object, it should be classified as an entity. If the object only requires one descriptor, it should be classified as an attribute.

- Classify multi-valued attributes as entities. If more than one value for a descriptor corresponds to one value of the unique identifier for an entity, the descriptor should be classified as a separate entity, even if it does not have its own descriptors.

- Attach attributes to the entities they most directly describe. For example, office_building should be an attribute of DEPARTMENT not EMPLOYEE.

- Avoid composite identifiers when possible. A composite identifier is a unique identifier which is made up of two or more attributes. If these attributes are all unique identifiers for other entities, then define this entity as a weak entity (an entity which relies on the keys from other entities to establish its uniqueness) or a relationship.

- Subtypes of entities should become entities. Some entities may contain attributes which are not common to all occurrences of the entity. For example, you may want to keep additional information about managers in the EMPLOYEE entity. This subset of information about managers should be kept in a separate entity.

Once the entities, attributes, and relationships have been identified, the entity-relationship (ER) diagram can be constructed. The ER diagram gives the designer a simplistic and readable view of the conceptual schema. ER modeling is a

method of representing data requirements using a set of semantic definitions. Currently no standard ER model exists; therefore, this paper will use the semantics and notations most widely used within Hewlett-Packard.

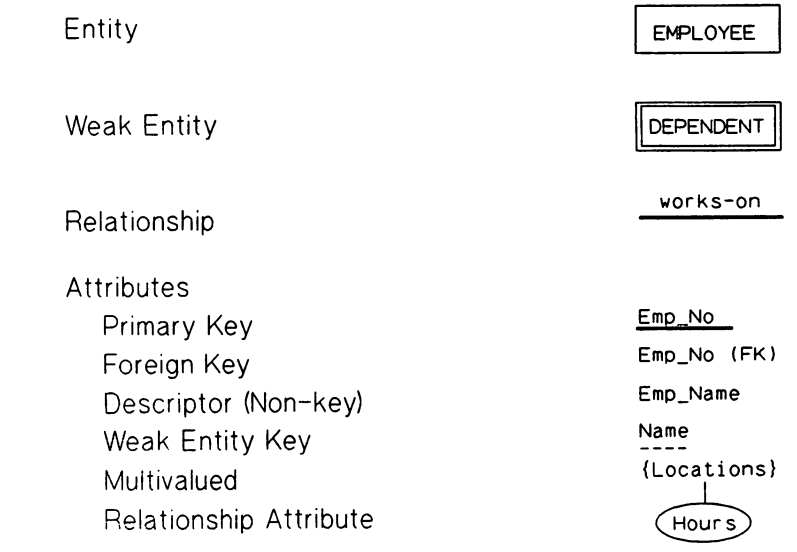


Figure 1. ER constructs for basic objects

Figure 1 illustrates the fundamental ER constructs used for basic objects while Figure 2 shows the fundamental constructs for relationship types. Using the symbols defined in these two figures, an ER diagram can be constructed which depict the conceptual data model in a pictorial form.

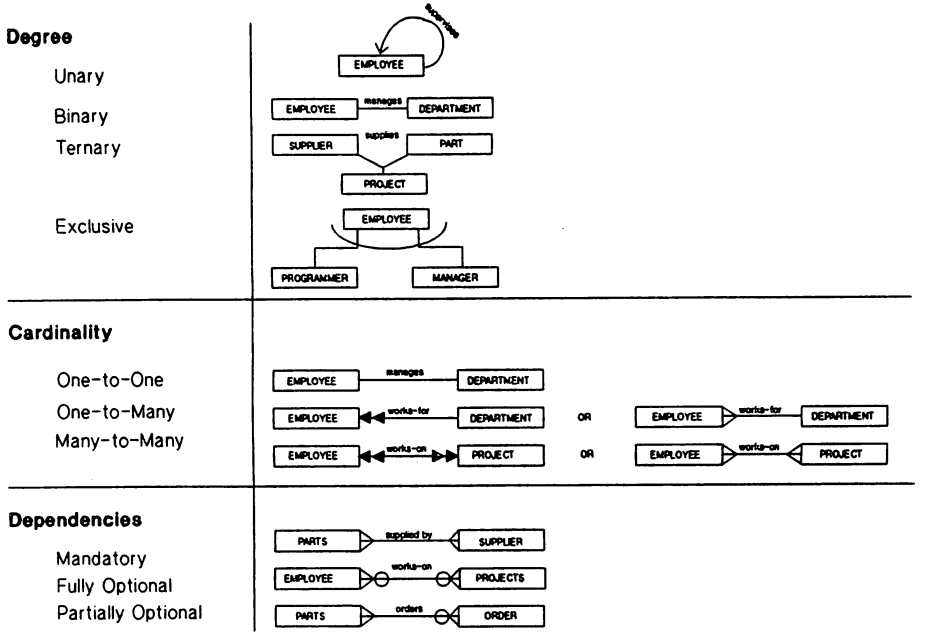


Figure 2. ER constructs for relationship types.

All of the basic objects in Figure 1 have been previously defined. We will now focus on defining the relationship types in Figure 2.

Degree - The degree of a relationship is the number of participating entities.

Unary - A unary relationship (also called an involuted relationship) exist when one occurrence of an entity has a relationship with another occurrence of the same entity.

Binary - A binary relationship is a relationship between two different entities.

Ternary - A ternary relationship relates three entities to each other in such a way that they cannot be decomposed into equivalent binary relationships.

Exclusive - A exclusive relationship is one in which an entity is shown to relate to two or more other entities but may have a relationship with only one of these entities at a time. The arc indicates an either/or relationship.

Cardinality - The cardinality ratio specifies the number of relationship instances that an entity can participate in.

One-to-one - A cardinality ratio of one-to-one (1:1) means that for each occurrence of an entity there can be only one occurrence of the other entity in the relationship.

One-to-many - A cardinality ratio of one-to-many (1:N) indicates that one entity occurrence is related to one or more occurrences of the other entity in the relationship.

Many-to-many - A cardinality ratio of many-to-many (M:N) means that many occurrences of one entity are related to many occurrences of the other entity in the relationship.

Dependencies - The dependency of a relationship refers to whether an entity must be present to support the relationship.

Mandatory - A mandatory relationship requires both entities to participate in the relationship. Neither entity may exist without the other. The participating entities are dependent on each other.

Fully optional - A fully optional relationship states that the participating entities in the relationship may exist independent of each other.

Partially optional - A partially optional relationship indicates that one entity is dependent on the other but the reverse is not true. The other entity may exist independently.

Figure 3 shows the information needs for a company's

database. This data will be used in further diagrams to illustrate the different data models.

DEPARTMENT

Dept-No, Dept-Name, {Location}, Manager, Mgr_Start

PROJECT

Proj-Name, Proj-No, Location, Control-Dept,
Completion-Code, Completion-Date

EMPLOYEE

Emp-Name, Emp-No, Address, Salary, SSN, Job_Code,
Dept, Supervisor, {Project, Hours}, Prim-Lang, Sec-Lang,
Mgmt-Level, Yr-Mgr

DEPENDENT

Name, Relationship, Sex, BDate, Employee

Figure 3. Company data.

Using the company data, figure 4 illustrates the first pass of the conceptual data model before refinement has taken place.

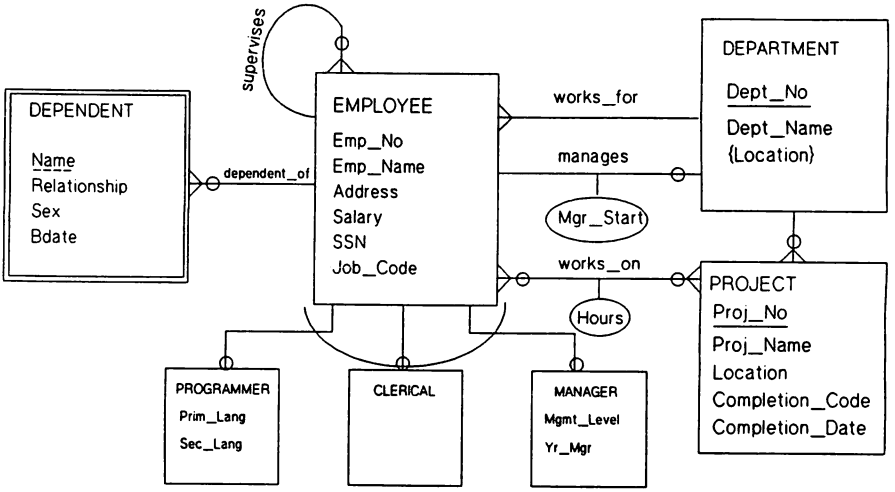


Figure 4. Company Database Conceptual Model before refinement.

Once the preliminary conceptual data model is developed, the designer must go through the refinement process. During this process primary keys must be identified, entities must be reviewed, and redundant entities must be removed. Many-to-many and involuted relationships must be examined for hidden entities. If implied relationships exist, remove any unnecessary direct relationships. Foreign keys need to be identified to support relationships.

Figure 5 shows the Company database final conceptual data model.

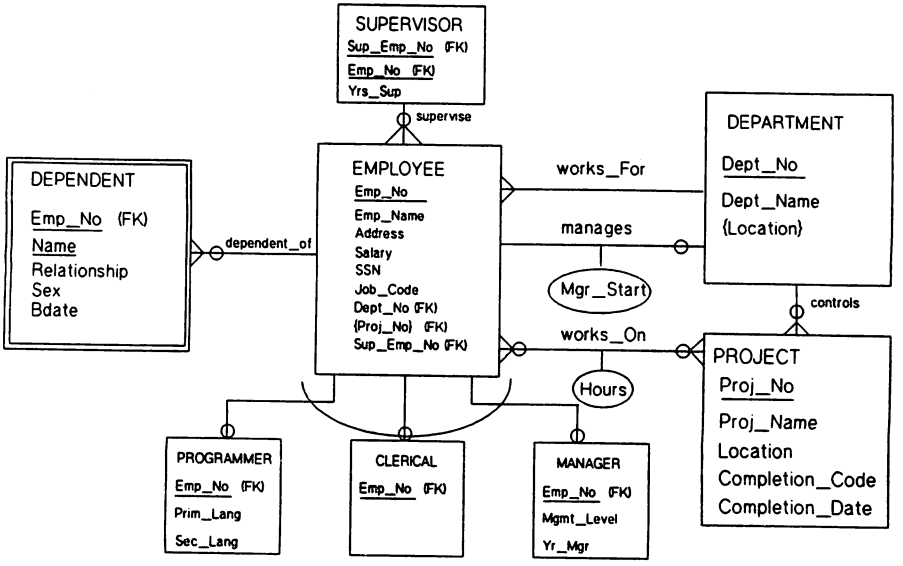


Figure 5. Company Database Conceptual Data Model after refinement

Defining and refining the conceptual data model is an iterative process. There should be several modeling sessions, each producing a more complete and detailed model. As previously mentioned, transaction design should be occurring in parallel with conceptual data modeling.

Transaction Design

When a database is being designed much is already known about the applications that will use the database. An important

part of database design is to understand the characteristics of the transactions that will be applied against the database early in the design process.

One technique for specifying transactions is to identify their input/output and functional behavior. Transactions are usually grouped as retrieval, update, or mixed transactions. Knowing these characteristics, as well as the relative importance of transactions and their rate of invocation is a crucial part in physical database design.

Logical Database Design

The third phase of database design is to develop of the logical data model. Although the logical model is still DBMS independent, the goal of logical data modeling is to come up with a record-based schema which will easily translate into a physical data model.

Logical data modeling includes the following activities:

- Examine attributes which may contain null values. Often a null-valued attribute is indicates of a hidden entity. Avoid keys that may have null values. Many database experts believe that nulls should always be avoided.

- Examine subtype entities. If a subtype entity has a different set of attributes, or if one of the subtype categories has attributes then make it an entity.

- Convert all many-to-many relationships to two one-to-many relationships with a connecting entity to link them together. The connecting entity should have the primary keys on the two new entities as its foreign keys.

- Examine fully optional relationships and partially optional one-to-many relationships where the optional part is the 'one' side. This is okay if the optional part is on the 'many' side. To avoid null foreign keys, consider placing a connecting entity between the two original entities.

- Normalize the data model to third normal form. During physical database design it may be desirable to denormalize parts of the model to maximize transaction performance.

The goal of logical data modeling is to address a limited number of implementation issues and prepare the model for physical database design. Figure 6 shows the ER diagram for the Company database logical data model.

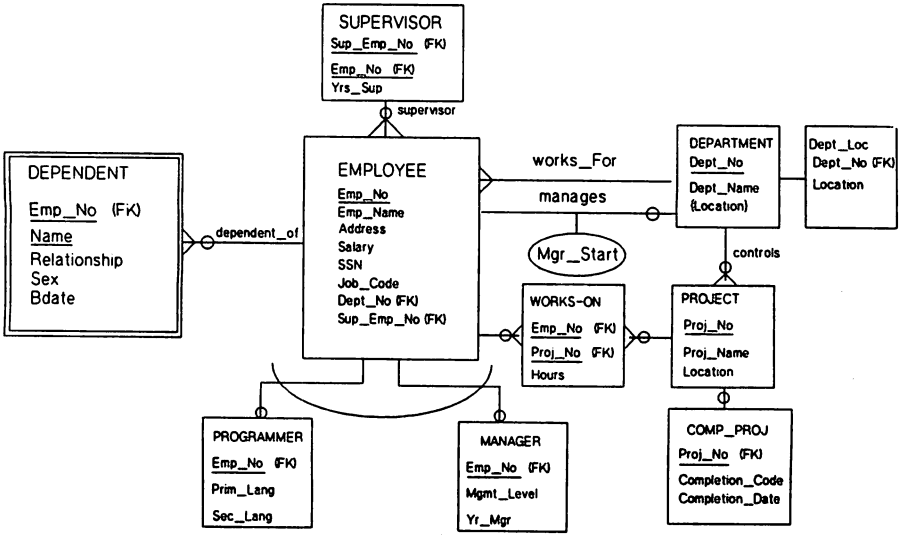


Figure 6. Company Database Logical Data Model

Physical Database Design

The next phase of database design entails converting the logical model into a physical database design. The physical design is a merging of the logical design, the transaction design, and the rules of the chosen DBMS.

The first step in physical design is to map the entities, attributes, and relationships into components of the DBMS. This can usually be done by following a few generalized steps.

Step 1: Each entity translates to a relation in the physical model. All attributes of the entity will become attributes of the relation. The primary key should be supported by a unique index and all foreign keys should be supported by non-unique indexes.

Step 2: For each weak entity type, create a relation with all the attributes of the weak entity plus the primary key attribute(s) of the owner entity. The primary key of the new relation will be the a composite of the primary key of the owner entity and the partial key of the weak entity.

Step 3: For each one-to-one relationship type, the primary key of at least one of the participating entities must be included as a foreign key of the other participating entity. It is best to choose an entity type with total participation for placing the foreign key.

Step 4: If any relationship types have attributes associated with them, the attributes should be moved to one of the participating entities. If the attribute is multivalued, the relationship should become an entity.

Step 5: For each one-to-many relationship type, the entity on the 'many' side should contain the primary key of the entity on the 'one' as a foreign key.

Step 6: If there are any many-to-many relationships, create a new connecting relation which has as its foreign keys the primary keys of the participating many-to-many entities.

Step 7: If there are any multivalued attributes, create a

new relation with the primary key of the original entity included as a foreign key in the new entity.

Step 8: For each n-ary relationship type (n is the number of participating entities) where n > 2, create a new relation. Include in the new relation as foreign keys the primary keys of the participating entities. The primary key of the new relation is the combination of all of the foreign keys.

After the physical data model is designed it must be tuned for performance, flexibility, updatebility, accessability and so on. During this step the information from the transaction design is heavily used.

When fine tuning the physical design the designer must understand how the database will be accessed. For each transaction, the designer should know:

- The files that will be accessed by queries or updates.
- The fields on which any selection conditions are specified.
- The fields on which any join condition are specified.
- The fields whose values will be retrieved by the query.
- The types of update operations on each file.
- The fields on which any selection conditions for a delete or modify operation are specified.
- The fields whose values will be changed by a modify operation.
- The frequency of invocation of the transactions.
- The time constraints of the transactions.

Relations may be denormalized, merged together, or divided to improve performance. Indexes need to be evaluated. Rules of the DBMS need to be applied to the data model to optimize its efficiency. Locking strategy, views, security, audit needs,

file placement, and database parameters all have to be considered. Once the physical database design is considered fully tuned and complete, it will need to be implemented.

Database Implementation

Implementing the database is usually done by the database administrator (DBA) using the guidelines defined by the database designer. Using the data definition language (DDL) of the DBMS, the DBA will create the database environment, tables, indexes, views, and security. The DBA will set the database parameters and define the backup strategy. After the designers specification have been fully carried out, the database will be ready to load and use.

Conclusion

For many years database design was an afterthought. Applications were written and databases where built without any real understanding of the relationship of the two. All too often it was after the database application was implemented that the creators realized the design errors (or lack of design errors). Once applications are written for a database changing the design of the database without extensive application program changes is virtually impossible. Most of the cost of an application is in its maintenance. This cost can be greatly reduced by good, thorough database and application design up front.

DBChange Plus: New and Improved

Mark Boronkay
Hewlett-Packard
19111 Pruneridge Avenue, Bldg. 44MA
Cupertino, California 95014
(408) 447-5009

Have you ever experienced database corruption? Hopefully not but if so, think about how the corruption was resolved. Some shops rebuild their database from scratch. Others might risk further corruption by attempting to patch the database themselves. In any case, it is certain that a significant amount of time will be involved. Now think about this. Wouldn't it be great if database corruption could be detected and resolved in a timely manner not requiring massive amounts of technical expertise? With **DBChange Plus** it is now possible to check a database for corruption and resolve the corruption quickly and easily. By issuing the CHECK BASE command, DBChange Plus will scan your entire database looking for corruption. Similarly, the FIX BASE command will direct DBChange Plus to resolve corruption within the database. DBChange Plus is an extended database administrative tool containing all the restructuring features provided by its predecessor DBCHANGE/V, as well as new features designed to decrease the effort required to maintain a database.

What is DBChange Plus?

DBChange Plus is a command-driven database maintenance tool which offers a simple way of performing database restructuring and maintenance tasks. DBChange Plus saves a copy of the database structure and uses a change file to keep track of requested changes. The changes are then applied when requested by the user. All changes are performed without the use of a database unload or load. The following functions are available with DBChange Plus:

- **Restructuring a database**
- **Capacity Management**
- **Checking a database for structural corruption**
- **Fixing structural corruption**
- **Erasing a dataset**
- **Repacking a dataset**
- **Copying a database**
- **Displaying database structural information**

How does DBChange Plus run?

DBChange Plus consists of 3 files. One of the files is a message catalog (DBCC000) containing the bulk of the messages generated by DBChange Plus. All DBChange Plus commands and options are also included in this message file for verification upon user input. Using a message catalog not only eliminates the need to hard code, but also allows future messages and commands to be added or localized more easily. The other two files are: **DBCPLUS** and **DBAPLUS**.

DBChange Plus is a two-step process consisting of a front end processor and a database generator. The front end program used by DBChange Plus is called DBCPLUS. DBCPLUS prompts the user for input and stores the requests in a change file. If the change file does not exist, DBCPLUS will build it. If the change file does exists, DBCPLUS will allow you to either write over or add to the existing data. The second

program, DBAPLUS, then reads the change file created by DBCPLUS and does the actual work requested by the user.

These two programs, DBCPLUS and DBAPLUS, can be executed as a single cohesive unit or they can be run independently from each other. Issuing the PERFORM COMMANDS command within DBCPLUS will launch a DBAPLUS process which will then read the change file and transforms the database accordingly. The following example illustrates how easy it is to use DBChange Plus interactively to increase the capacity of a dataset:

```
:DBCPLUS                          <<Begin the DBChange Plus process>>
>Base ORDERS                      <<Define the database>>
>Change Capacity INVENTORY 650    <<Increase capacity of INVENTORY dataset to 650>>
>Perform Commands                 <<Launch DBAPLUS>>
>Exit                             <<Terminate DBChange Plus>>
```

In the above example, entering DBCPLUS at the colon prompt initiates the DBChange Plus utility. The first command entered in the above example is the BASE command. This command defines the database for DBChange Plus. At this point, DBChange Plus builds (or modifies) the change file. Next, the command to increase the capacity of the INVENTORY dataset to 650 is entered. Finally the PERFORM COMMANDS command is issued. This will launch the DBAPLUS program as a son process. Control will return to the user when the DBAPLUS process is completed. The EXIT command causes DBChange Plus to terminate.

(When using the PERFORM COMMANDS command, DBAPLUS will automatically check the value of a special JCW set by DBCPLUS. DBAPLUS will not execute if the value is anything other than '0'. This built-in feature of PERFORM COMMANDS prevents DBAPLUS from executing if some type of error is detected in DBCPLUS.)

Requests could also be batched by the user in the change file only to be processed by DBAPLUS at a later time. To do this, the user simply runs the DBCPLUS program and does not indicate PERFORM COMMANDS before exiting DBCPLUS. At a later time, the user can simply run DBAPLUS in which case they will be prompted for the name of the database to transform.

File equates can be used in order to implement DBChange Plus as a batch process, UDC or command file. Special JCWs have also been employed by DBChange Plus to help the user control the flow of processing between DBCPLUS and DBAPLUS when used as separate programs. **DBCPLUSJCW** will be set to non-0 if an error occurs during DBCPLUS. Likewise, **DBAPLUSJCW** will be set to non-0 if an error occurs during DBAPLUS. The example on the following page illustrates a typical jobstream and its use of file equates and JCW checking:

```
!JOB CAPINCR,User.Acct;Outclass=LP,2
!DBCPLUS
Base ORDERS New                   <<Define database for DBCPLUS>>
Change Cap SALES 300              <<Change capacity of SALES to 300>>
Exit                              <<Exit DBCPLUS>>
!If DBCPLUSJCW = 0 THEN           <<Check DBCPLUS JCW>>
!   FILE DBCHGF=ORDERSCF          <<Define change file for DBAPLUS>>
!   RUN DBAPLUS.PUB.SYS;PARM=15   <<Launch DBAPLUS>>
!   IF DBAPLUSJCW = 0 THEN        <<Check DBAPLUS JCW>>
!      TELL User.Acct; Capacity Increase Successful!!!!
!      ENDIF
!ENDIF
!EOJ
```

How Does DBChange Plus Work?

The first thing to note about the command interface is that there are two distinct classes of commands used by DBChange Plus: **DEFERRED** and **IMMEDIATE**. Deferred commands are those that will change the structure of a database. They are considered deferred because DBChange Plus simply stores them in the change file until a user explicitly directs DBChange Plus to execute them. That is, by themselves they have no immediate effect on the database. The deferred commands are:

ADD, CHANGE, CHECK, DELETE, ERASE, FIX, RENAME, REORDER, RECOVER, REPACK

Immediate commands are those that do not alter the original database. They provide information about or help with the operation of the DBCPLUS program. Their effect is immediately known. The immediate commands are:

BASE, CANCEL, CONTROL, COPY, EXIT, HELP, OUTPUT, PERFORM COMMANDS, PRINT, RECOVER, REDO, REVIEW, XEQ

As noted earlier, DBCPLUS simply builds the change file containing the users requests. The database is actually transformed by DBAPLUS where a new schema is built based on the change file. That new schema is then file equated to DBSTEXT. DBSCHEMA next creates the new rootfile. Following the creation of the new rootfile, individual temporary files representing datasets are built based on those sets designated in the change file containing changes. Data is then transformed and/or copied where appropriate. Finally the current rootfile and datasets are purged and the new temporary rootfile and datasets are saved as permanent.

DBChange Plus will order all deferred commands intelligently for maximum data safety and throughput. For example, if a user entered a REPACK command prior to a FIX command, the FIX command will be executed first. Likewise, if two contradictory commands are entered, the most recent command entered takes precedence. An example of that would be if a user changes the capacity of a dataset to 20,000 and a moment later changes the same capacity to 10,000. The most recent request (capacity 10,000) will be the one used during database transformation time.

MPE commands can be issued within DBChange Plus. Simply input the MPE colon : and the MPE command after the DBCPLUS prompt > (for example, >:LISTF). The command interface also supports the MPE RUN command (for example, >:RUN QUERY.PUB.SYS).

What Does DBChange Plus Do That DBCHANGE/V Doesn't?

In addition to retaining the features of its predecessor DBCHANGE/V, DBChange Plus adds the ability to:

- o check a database for structural corruption,
- o fix many structural problems
- o monitor the capacity (fill rate) of a dataset and automatically increase or decrease its size
- o give optimal performance recommendations
- o change maximum block length (BLOCKMAX)
- o delete item/set security
- o erase datasets
- o repack detail data sets

Each of the above new features were designed to give the user added flexibility and power in the design and maintenance of TurboIMAGE databases. Let's take a closer look at each of these new features.

CHECK DATABASE

DBChange Plus has the ability to check a database (or parts of it) for various structural integrity problems. There are two different methods DBChange Plus can use to check a database: **QUICK** and **STANDARD**. The QUICK method does a check to determine whether a problem truly does exist or not. In order to do this check quickly, DBChange Plus uses checksum information derived from the database instead of following all the chains. This method will identify problem datasets but will not isolate the particular entry that needs to be fixed.

A more precise check can be done with STANDARD checking. This method takes a bit longer since it follows chains, but it does a better job in isolating where the corruption exists. If corruption is found in the database, the CHECK function will generate an analysis and a diagnostic file. This file can then be used by the FIX function to resolve the corruption.

CHECK can be performed on the rootfile, dataset(s), path(s), or the entire base. CHECK ROOT will look for inconsistencies in the rootfile. CHECK PATH will detect errors between datasets such as chain head and chain count inconsistencies. CHECK SET will detect inconsistencies within a dataset such as bitmap problems and forward/backward pointer on a chain. The CHECK SET analysis will contain the following information:

- **number of entries in the set**
- **capacity of the set**
- **percentage full**
- **high water mark (detail datasets only)**
- **delete chain count (detail datasets only)**
- **percentage of secondaries (master datasets only)**
- **longest cluster of blocks required for open slot (master datasets only)**
- **average cluster of blocks required for open slot (master datasets only)**

The following is an example of output from the CHECK SET @ (check all data sets) command:

```
CHECK SET (MASTER)

                                                 Pct  Pct  Longest  Average
Master Set Name  Type   Entries   Capacity      Full  Sec  Cluster  Cluster
-------------------------------------------------------------------------
CUSTOMER          M          10        221        5  0.0        0      0.0
DATE-MASTER       A           9        211        4  0.0        0      0.0

CHECK SET (DETAIL)

                                          Pct      Highest        Delete
Detail Set Name      Entries    Capacity  Full  Entry Used  Chain Count
-------------------------------------------------------------------------
SALES                    60         308    19          60            0

CHECK INFORMATION

Set Name           Type
  Search Item
    Message(s)
-------------------------------------------------------------------------
No problems were detected by CHECK.
```

The CHECK PATH function detects broken chains or incorrect pointer linkages. For a master set, CHECK PATH follows and examines synonym chains. For detail sets, CHECK PATH follows and examines detail chain pointers and the chain head pointers in the associated master data sets. The following data is reported by CHECK PATH:

- o search item name (will display SYNONYM CHAINS for masters)
- o set type
- o maximum number of entries in the longest synonym or detail chain
- o average number of entries per chain
- o standard deviation of average number of entries per chain
- o percentage of forward pointers that point outside the current block
- o average number of blocks per chain
- o packing ratio (efficiency of path)

The following is an example of what the CHECK PATH @ (check all data paths) command will produce as a report:

```
CHECK PATH

Set Name                Type      Max      Avg      Std   Pct Far      Avg      Packing
  Search Item           (PS)    Chain    Chain      Dev      Ptrs   Blocks        Ratio
-------------------------------------------------------------------------------------
CUSTOMER                 M
  SYNONYM CHAINS                   1     1.00     0.00         0     1.00          N/A
DATE-MASTER              A
  SYNONYM CHAINS                   1     1.00     0.00         0     1.00          N/A
PRODUCT                  M
  SYNONYM CHAINS                   1     1.00     0.00         0     1.00          N/A
SALES                    D
  ACCOUNT                (S)      47    20.00    23.81        30     7.00         0.14
  STOCK#                 (P)      45    20.00    22.61        12     3.33         0.30
  PURCH-DATE                      23     7.50     8.85        13     2.00         0.50
  DELIV-DATE                      29    15.00    11.22        12     2.75         0.36
INVENTORY                D
  STOCK#                           0     0.00     0.00         0     0.00         0.00
  SUPPLIER                         0     0.00     0.00         0     0.00         0.00
  LASTSHIPDATE                     0     0.00     0.00         0     0.00         0.00

CHECK INFORMATION

Set Name                Type
  Search Item
    Message(s)
-------------------------------------------------------------------------------------
No problems were detected by CHECK.
```

The CHECK BASE function does an implicit CHECK ROOT, CHECK PATH @, and CHECK SET @. The report produced is a concatenation of the CHECK PATH and CHECK SET analysis. The following is an example of a job that can be streamed at desired intervals. It does a full check against a database called ORDERS. By simply changing the base name, the same job can be run against other databases.

```
!JOB CHECKER,User.Acct;Outclass=LP,2
!DBCPLUS
Base ORDERS New              <<Define database for DBCPLUS>>
Check Base                   <<Standard database Check>>
Perform Commands 15          <<Do Check Now!!!>>
Exit                         <<Exit DBCPLUS>>
!EOJ
```

FIX DATABASE

The FIX function attempts to resolve the corruption uncovered based on the data generated by CHECK. If FIX is indicated without a CHECK, it will generate its own diagnostic file by running CHECK prior to the FIX routines. FIX can recognize and resolve the following problems:

- o **User Label Inconsistencies**
- o **Delete Chain Errors**
- o **Broken Chains**
- o **Bit Map Errors**
- o **Unlinked Entries**
- o **Root File Path Sequence Corruption (also corrected during restructure)**

Consider the following seven entry chain in a detail data set called ACCOUNTS (record numbers are pictured as decimal values) where record #26 is the head of the chain and record #19 is the tail of the chain:

```
26<---->27<---->28<---->7<---->8<---->16<---->19
```

Now suppose that for some reason, the forward pointer for record #27 pointed to #100 rather than #28 as it should. Furthermore, suppose that the backward pointer for record #8 pointed to #200 rather than #7 as it should. The following illustration depicts the broken chain environment:

```
          ------>100
          |
26<----->27<---- 28<----->7 ----->8<----->16<----->19
                                  |
                          200<-----
```

The example on the following page illustrates the output generated by FIX BASE on this corrupted database. Note that the report prior to the error messages is again a concatenation of CHECK SET @ and CHECK PATH @.

```
CHECK SET (MASTER)

                                                    Pct   Pct   Longest  Average
Master Set Name   Type    Entries    Capacity      Full  Sec   Cluster  Cluster
-------------------------------------------------------------------------------
CUSTOMER          M            10         221       5  0.0         0      0.0
DATE-MASTER       A             9         211       4  0.0         0      0.0
PRODUCT           M            10         307       3  0.0         0      0.0

CHECK SET (DETAIL)

                                          Pct      Highest        Delete
Detail Set Name      Entries    Capacity  Full  Entry Used  Chain Count
-------------------------------------------------------------------------
SALES                     60         308    19          60           0
INVENTORY                  0         450     0           0           0

CHECK PATH

                                                          Pct
Set Name            Type     Max      Avg      Std      Far      Avg  Packing
  Search Item       (PS)   Chain    Chain      Dev    Ptrs   Blocks    Ratio
-------------------------------------------------------------------------------
CUSTOMER            M
  SYNONYM CHAINS                1     1.00     0.00       0     1.00      N/A
DATE-MASTER         A
  SYNONYM CHAINS                1     1.00     0.00       0     1.00      N/A
PRODUCT             M
  SYNONYM CHAINS                1     1.00     0.00       0     1.00      N/A
SALES               D
  ACCOUNT           ( S)       47    20.00    23.81      32     7.33     0.14
  STOCK#            (P )       45    20.00    22.61      12     3.33     0.30
  PURCH-DATE                   23     7.50     8.85      13     2.00     0.50
  DELIV-DATE                   29    15.00    11.22      12     2.75     0.36
INVENTORY           D
  STOCK#                        0     0.00     0.00       0     0.00     0.00
  SUPPLIER                      0     0.00     0.00       0     0.00     0.00
  LASTSHIPDATE                  0     0.00     0.00       0     0.00     0.00

CHECK INFORMATION

Set Name            Type
  Search Item
    Message(s)
-------------------------------------------------------------------------------
SALES               D
  ACCOUNT           (Path 1; linked to master CUSTOMER, path 1)
    Path Chain inconsistencies detected (CHK 400).
    Chainhead record 15 chain count mismatches entries on chain (CHK 450).
    Record 7 is not linked into the proper chain (CHK 460).
    Record 8 contains bad backward pointer (CHK 440).
    Record 27 contains bad forward pointer (CHK 430).
    Record 28 is not linked into the proper chain (CHK 460).

FIX INFORMATION
-------------------------------------------------------------------------------
All requested fixes have been successfully applied.
```

Note that the corruption is characterized at the end of the report and that there is a FIX INFORMATION section verifying that the fixes were applied.

CAPACITY MANAGEMENT

The ability to monitor the growth of particular data sets is very useful. How many times have you had to rerun a job after finding out that a data set filled up and the program aborted? Sound familiar? DBChange Plus has the ability to control the growth of data sets by previewing the capacity versus the entry count of data sets. If the percentage differs from what you tell DBChange Plus it should be, then that set is either flagged as needing a capacity change OR the new capacity is automatically inserted in the change file. With the CONTROL PERCENTFULL command, the user determines whether to CHANGE CAPACITY manually or let DBChange Plus change the capacity. CONTROL PERCENTFULL has the following command syntax:

```
CONTROL PERCENTFULL SETNAME MINFULL MAXFULL [NEWFULL]
```

SETNAME can be for one set or one of the following:

@ to indicate that CONTROL PERCENTFULL applies to all data sets
@MASTERS to indicate that the CONTROL PERCENTFULL applies to all master data sets
@DETAILS to indicate that the CONTROL PERCENTFULL applies to all detail data sets

MINFULL is the minimum percentage full desired for the given data set. If the set is less than the MINFULL, DBChange Plus will print a warning indicating so. Zero may be used for this parameter.

MAXFULL is the maximum percentage full desired for the given data set. If the set is more than the MAXFULL, DBChange Plus will print a warning indicating so.

NEWFULL is the desired percentage full for the given data set. If the set is less than MINFULL or greater than MAXFULL, the NEWFULL option will AUTOMATICALLY COMPUTE and ADJUST the capacity within the change file.

If the capacity is to be expanded, the data set file will be enlarged. If the detail set capacity is to be reduced, a REPACK SET SERIAL will be used to reduce the file. The new capacity will be based on the desired percentage full (NEWFULL) and will be adjusted: to avoid 2's complement numbers for master sets, and to round to the nearest block for detail sets. If a CAPACITY CHANGE or REPACK SET is already pending for a data set, CONTROL PERCENTFULL may print a warning and cancel the previous change. Some examples of CONTROL PERCENTFULL are:

CONTROL PERCENTFULL @ 0 70: If any of the data sets are more than 70% full, print a message indicating so.

CONTROL PERCENTFULL @DETAIL 55 85 65: If any of the detail data sets are less than 55% full or greater than 85% full, change the capacity so that it becomes no more than 65% full.

CONTROL PERCENTFULL INVENTORY 60 80: If the INVENTORY data set is less than 60% full or greater than 80% full, print a message indicating so.

OPTIMIZE PERFORMANCE

Performance recommendations are made indirectly through the reports produced by CHECK SET and CHECK PATH commands. Regular checks of the database can prevent performance problems caused by large clusters in master data sets, poor packing on heavily used chains, and large gaps caused by deleted records in detail data sets. Let's take a look at how the statistics in the CHECK PATH report can offer us help in performance tuning.

MAX CHAIN and AVERAGE CHAIN: While long chains are not necessarily harmful to the database, they can have a considerable effect on performance. Consider the case of a master set with an inefficient capacity. This situation can lead to an inordinate amount of synonym chains which will effect performance during DBPUTs to their related detail datasets. Another example is the use of sorted chains. The longer the chain becomes, the more time it may take to add items to that chain. You may want to consider the following guidelines when evaluating chain lengths based on the CHECK PATH report:

o choosing an efficient capacity will keep synonym chains to a minimum.
o the average chain should not exceed 40% of the capacity.
o data items most heavily used should be specified as a search item for a detail data set repack.
o sorted chains should be less than 50 records unless key values are added in ascending order over time.

In addition to chain length evaluation, other inferences can be made from the statistics offered by CHECK PATH:

STD DEV (standard deviation) is an indication of the accuracy of the Avg Chain statistic. The closer to 0.00 this number is, the more accurate the Avg Chain statistic is.

PCT FAR PTRS is the percentage of forward pointers that point outside the current block. This statistic can be used to enhance the packing ratio described below.

PACKING RATIO is the efficiency of the path. It is the optimal average number of blocks per chain divided by the actual number of blocks per chain. A value of 100% means that every individual chain for the specified path occupies the minimum number of blocks possible. Although packing ratio will vary from application to application, try to maintain a packing ratio of at least 60% on primary paths.

The CHECK SET can also offer us statistics which can help fine tune performance. In the area of master data sets, DBChange Plus offers these statistics:

PCT FULL is the percentage of the data set capacity currently in use. The recommended percent full for a master data set is between 60% and 80%. If a master data set capacity is significantly less than 60% full, a serial read becomes slower. If a master data set is greater than 80% full, DBPUT intrinsics involving the master data set can slow down.

PCT SEC is the percentage of secondary entries. In general, the lower the percentage. the better. A high percentage indicates that the hashing algorithm is creating many synonyms. To decrease the percentage of secondaries, increase the data set capacity to a larger number that is not a power of two.

LONGEST CLUSTER and AVERAGE CLUSTER are the longest and average number of TurboIMAGE blocks that must be read to find an open slot to place a synonym. If two records with the same hash value are added to a master set, one of them must be placed in another slot. Acceptable guidelines range from a cluster of 10 on a heavily loaded transaction processing system to as much as 200 on a very lightly loaded system. A collection of historical statistics about the database can help you evaluate if the longest and average clusters are within a reasonable range. Your database users can help identify clustering problems by reporting slow response time when adding records to a particular set. Once you have determined if the range should be changed, increase or decrease the data set capacity accordingly.

In the area of detail data sets, DBChange Plus offers us these statistics:

PCT FULL is the percentage of the data set capacity that is not available for use. This statistic is useful for capacity planning.

HIGHEST ENTRY USED is the record number of the highest entry ever used. When reducing data

set capacity, do not reduce it below the highest entry used; otherwise you must repack the data set to recover the unused space in the middle of the data set.

DELETE CHAIN COUNT is the number of records in the delete chain. This number should be as close to 0 as possible. A high delete chain count may mean a problem with a large quantity of deletes. For example, if you have a program that performs a large number of deletes, gaps may be left in your detail data set. Subsequent record additions may disburse data randomly within the data set. As a result, chain reads may be slow. To correct a high delete chain count, repack the data set.

CHANGE BLOCK LENGTH (BLOCKMAX)

There are instances where the default block size (512 half-words) is not sufficient to hold an optimal number of TurboIMAGE entries. And without optimizing the block lengths, it is possible that disc space could be wasted. Let's say for instance that you have a particular dataset whose media length is 100 half-words and the blocking factor is 10. In this case, the block size must be at least 1001 half-words in order to hold all 10 entries. Obviously, a block size of 512 half-words is not sufficient for this dataset. The CHANGE BLOCKMAX feature within DBChange Plus will allow the user to modify the block length for this particular dataset so that all 10 entries will fit properly.

The CHANGE BLOCKMAX command can be specified for all datasets, for a range of datasets or for particular datasets. To apply the new BLOCKMAX to all datasets, the user specifies the '@' as one of the options in CHANGE BLOCKMAX command. To apply the new BLOCKMAX to a range of datasets, the user specifies the beginning dataset and ending dataset as options in the CHANGE BLOCKMAX command. To apply the new BLOCKMAX to a particular dataset, the user specifies the single dataset. An example to set the BLOCKMAX to 2048 for the whole database would look like this:

```
>CHANGE BLOCKMAX @ 2048
```

This command is similar to the $CONTROL BLOCKMAX command used in DBSCHEMA. In fact, DBChange Plus essentially inserts a $CONTROL BLOCKMAX command in the schema at the appropriate spots prior to rebuilding the database.

DELETE ITEM/SET SECURITY

As an application changes, it sometimes becomes necessary to change the type of availability to data the end user has. One way this can be accomplished is to modify the application itself to prevent the user from accessing the data. Another method is to change the read/write classes at the item or set level.

The following example illustrates how easy it is to delete the set security for a dataset resulting in a read-only dataset to all except the database creator:

```
>DELETE SETSECURITY orders
```

ERASE DATASET

Unlike DBUTIL ERASE which erases the entire database, the DBChange Plus function deletes all entries in a given dataset. There are three cases which apply to the ERASE command: erasing details, erasing manual masters, erasing automatic masters.

Details:
All entries in the detail set are erased and corresponding master sets are updated. If the deletion of a detail entry results in an manual master entry not being linked to any detail entries, then the master entry is left in the set.

Manual Masters:
In a manual master set, only those entries with no corresponding detail entries are erased. If the set is linked to one or more detail sets whose path count is not zero, a message is displayed stating that the master cannot be erased due to existing detail set entries.

Automatic Masters:
Automatic master datasets cannot be explicitly erased.

In order to prevent accidental deletion of data, DBChange Plus asks the user to confirm the ERASE command in session mode. The confirmation is bypassed during batch processing. The CANCEL ERASE command allows the user to cancel an ERASE command that has not yet been executed. It is important to note that once DBAPLUS has been run, the ERASE COMMAND cannot be cancelled.

REPACK DATASET
The REPACK SET function comes in two flavors: **serial** and **chained**. The chained version will repack your set along a specified path. If no path is specified for the chained repack, then the primary path is assumed. Repacking a dataset along a specified path is an excellent way to improve performance of a chained read. A serial repack is useful when reducing the capacity in a detail data set which is highly fragmented (i.e., a dataset which has many chunks of data intermixed with many chunks of free space). The serial version of repack does not require a path name. The REPACK SET command is valid for detail data sets only. A CANCEL REPACK command will cancel a previous REPACK SET command. Again it is important to note that once DBAPLUS has been run, the REPACK SET command cannot be cancelled.

The repack works by rebuilding the set, eliminating gaps left by deleted entries and adjusting the pointers. In order to accomplish this, DBChange Plus uses an internal mapping file. Once the mapping file is built and loaded with data DBChange Plus proceeds to rebuild according to whether the repack is serial or chained.

Summary.....

As you can see, DBChange Plus provides database users with a simple solution for database restructuring and maintenance tasks. In addition, it also provides features to help the user make performance decisions based on data analysis. But most of all, it is designed to be easy and simple to use.

Paper 3247

Develop Software Using a Synthesis Approach

Phil Nguyen
Wayne McKinney
Lockheed Engineering and Sciences Co.
2400 NASA Road 1, Houston, TX 77058
(713) 333-7177

I. Software Reuse Is Here

Traditional software development methodologies are inadequate. They often have limited perspective and never address issues such as program integration or software reuse. On the other hand, one characteristic of rapid software development is that it emphasizes code reuse throughout the project's life cycle.

Recently, our financial analysis group requested that an application be developed for them that could compile data from time cards. They wanted it to acquire data for a month and then create a summary report of labor grades by skill code within division and branch.

From this project, we learned how to orient ourselves toward developing reusable software. The application was constructed using a number of modular subroutines, where each subroutine was independent, compact, and handled one specific function. Because the code was highly modular, we could easily retrofit it for use in future applications.

Several other good results came out of our work with that application. We have reduced thousands of lines of code to hundreds, thus making the task of software maintenance easier and less costly. Moreover, the application's code was structured so that it was easier to understand and easier to test.

By using data tables in the application, we derived a second benefit. The subsystem was made more reliable and more flexible. It can now handle many changes in business entities with a minimum of effort by mis production personnel, and it can handle those changes in real time.

Our work has shown us that when a project is completed, an effort should be made to identify what software components could be added to a software reuse library. Once the reusable code has been identified, we should next find efficient pieces of code and store these in the library as well. By having these examples and reusable subroutines in one place, all programmers can inspect them and exploit their capabilities. It makes sense to create such a library, for it lets us capitalize on both existing software resources and the expertise of good programmers.

Software reuse does not have to start at the beginning of a project - it can even start after a project has completed. The only prerequisite is the availability of a database that can serve as a common ground for both software developers and programmers. Once this system is in place, it can serve as a source code "textbook" that can be continually updated and annotated by the people who use it (Figure 1-1).

II. Members of a Program Family

The evolution of factory industry has reorganized the way that products are manufactured. Today, products often evolve into a family of products rather than a single product system. The main reason for this progression is that of economy and maintainability. Building products in a setting of mass production almost always costs us less. It does not matter whether we are talking about a line of Honda Accords or a family of medical equipment. Each model is slightly different from the rest. Software development should follow a similar approach.

"Synthesis" is used in "Synthesis Approach" to represent a systematic process for rapidly creating different members of a program family. Stepping back from our limited perspective as programmers, we were able to see the coherency of the functions that our programs must perform. Now, when new applications are being developed or programs are being modified, we are able to transfer code. By doing this, we are creating a new member of a software family.

Lockheed has an application that tracks the procurement of items critical to space shuttle flights. Throughout its use by our Purchasing department, it has proven to be a reliable application that assures both the timely acquisition of flight hardware and the avoidance of any procurement problems.

Specifically, this application scans all our purchasing records and computes the actual time elapsed for the various phases of procurement. It then compares elapsed time with the "standard" amount of time that these phases are supposed to take. Lastly, it prints a report detailing variances from these standards.

Later, we created another application that looks at a two month spread of fiscal year costing data, calculates deltas, and compares them with preset tolerances. Any values found to exceed the "standard" tolerances will be printed in a report. In actual use, this application can detect problems at an early stage to one of any 1,500 job accounts.

Do you see a similarity between these two applications? You should, because they are members of the same program family. Code that was developed to run reliably in a procurement environment, can now be modified to handle a data processing need in an entirely different setting. Developing software using the Synthesis Approach lets us map variations in requirements to variations on a standard design. Because this approach enables us to easily generate deliverable products, our software development can achieve a high level of productivity and a high level of quality.

Look at the applications that your data processing shop creates and maintains. Regardless of what language or product your applications are written in, you will find that functionally they are quite similar to each other. Do you see the makings of a family here? If you do, then you should be working at creating a software development environment that fosters the computer assisted generation of program family members.

We have in our shop a system that handles employee termination activities. Its primary task is to generate reports for management showing a summary count of employee terminations by job code, by branch, or by department. It also generates other pieces of information such as turnover rates. Code for this application was developed and tested to such an extent that it does its work flawlessly.

Now step back from the previous explanation and consider what this termination application does. It creates reports that summarize by different classes, and it calculates information based on these summaries. Later, our Purchasing department came back to us asking for an application that could perform vendor ratings. For each vendor, Purchasing wanted a summarized report that presented information such as how good the service was, pricing, delivery time, average days late, and so on. Do you see a similarity here? The employee termination application was modified so that it could provide the exact information that Purchasing needed. Development was quick and this application's performance has been quite good.

Was it by luck that we happened upon two similar applications? No. We were able to see similar requirements between these applications and take appropriate action. Using the Synthesis Approach, it's possible to generate program family members easily, flawlessly, and meet project deadlines. Can programmers accomplish this feat alone? No. User involvement is very much needed. To aid us in bringing users into the project, we need to use another tool of the Synthesis Approach - that of rapid prototyping.

III. User Involvement Is Needed

Thanks to rapid prototyping, users can now be brought into the development loop quite early in the software life cycle. An analogy to this kind of prototyping is that of producing a play. Think of the users of the application as the play's audience. You have the multifaceted job of being the writer, producer and director of the play. Just as in some modern plays, software development is often easier when you, the programmer, call for audience participation.

One example of a project where we involved users from the beginning was an application that rates the performance of vendors. Purchasing needed a system that could gather statistical information on vendors. By looking at a vendor's past performance, Purchasing wanted to determine which vendors were reliable and which ones could be depended upon to deliver on time.

As with any other development project, we held various meetings with Purchasing to determine what the specific requirements were. Once we reached an agreement upon the requirements, we decided to test how reliable they were by prototyping a demo of the vendor rating system. After system installation, output was constantly being reviewed to check the validity of the different rating categories.

Because the level of user participation was high, we were able to make a number of improvements to the prototype in an effort to deliver the final system. One area needing improvement was that of the reports. A detail report was difficult to read and contained redundant information. A summary report containing rating data confused the users. Even the information found in one report did not support the information found in another. It looked as though the reports were in a hopeless state. However, by getting the users to wade into the quagmire with us, we were able to correct each of these problems.

The users, clearly, did not want to be handicapped by their own system. As a result, they were motivated to meet with us so that the reports and presentation of the data could be changed to meet their needs. With our software environment, we were able to provide an immediate response to each change in requirements. Though our modifications were extensive, we kept a skeleton of the old prototype intact. We simply replaced old blocks of code with new blocks. Our case tool helped ensure that the new blocks could interface with the rest of the application.

What was created in the end was a very useful application that ensured the company success in locating vendors who provided quality services. The reports (Figure 3-1) look well organized, are easy to understand, and show at glance how well a vendor is performing. Even now, the application's users are constantly testing its functionality. They want to be sure that we left nothing out of this application. Could such a successful application have been created without the user's continual involvement?

Under traditional methodologies, software development can start only when needs are well defined and requirements are stable. Yet in a dynamic business environment, stable requirements will usually not happen. Even if stable requirements are present, users are often not able to present them in a formal written format. A better approach asks for rapid prototyping that can generate a test system that we can take to the user and let him evaluate. As we saw in the previous example, the user can often clarify the requirements if he has an example that he can evaluate.

IV. Benefits of Rapid Prototyping

Though the Synthesis Approach stresses the use of existing to aid in building new program family members, there will come a time when you don't have a prior model to work from. Does our approach break down in such a situation? Not at all, for with the aid of rapid prototyping, we can start building the "alpha" program and then expand from there. We encountered this "alpha" program when we built a system that aided Purchasing in the acquisition and tracking of items purchased for use on space shuttle flights.

Prior to the implementation of our system, Purchasing tracked flight critical hardware on a Lotus 1-2-3 spreadsheet. For small volumes of purchases, this method worked well, but when volume increased, they quickly needed a system that was more capable and more reliable than lotus. Our initial discussions with Purchasing showed that their requirements were vague.

We decided to go straight to the person who had previously handled the data on flight critical purchases. Discussions with this person gave us enough information that we could rapidly prototype a data base around which the application would be centered. From this data base, we created a series of screens into which purchasing data could be entered. We also prototyped an initial set of reports that analyzed the data that was entered.

After this prototype was used for a week, a meeting with the actual users of the system showed that they could now more clearly state their requirements. They wanted data sorted and categorized by dollar amount, a record of time intervals between procurement phases, and reports on any purchases that exceed the "standard" time intervals between phases. Purchasing wanted to be able to recognize any troublesome purchases and take steps to resolve those problems. We also desired a set of charts that could visually summarize for management the application's data.

At this stage of the project, we still didn't have any formalized written requirements for the application. However, we did have enough information from which we could create another generation of the prototype. In this next prototype, we included all of what the user had previously required. We also added more business oriented functions that made the user's job a lot easier (Figure 4-1).

Even after all of our requirements were formally stated and the final application delivered, the application's users often came back month after month requesting more changes. By prototyping each change and by borrowing code from existing applications, we were able to quickly deliver a new function that not only met the user's requirements, but also fulfilled his need for accurate and timely information. Business functions may change, but our application will be able to quickly change with it because we developed it using the software Synthesis Approach.

V. Conclusion

Often times, the software developer must assume a role similar to that of a chief cook who oversees a big restaurant. He has available to him a wealth of materials and tools with which he can make a wonderful creation. The only trick involved is knowing how to quickly integrate these tools to come up with a product that the users can view, touch, and evaluate.

Every restaurant has difficult-to-please diners. The users will often come back every month or every week to request change or enhancement to your system. However, like a good cook, you will not get upset because you know that by following the software Synthesis Approach, you can vary the ingredients to create a new and better dish that will surely please them.

By paying attention to the customers' palate, a big restaurant can cater to changing desires. For the software developer, this means you must pay attention to what the user asks for. Communication is essential. By using rapid prototyping and case tools, you will be able to quickly please your users and ensure that your application remains a highly functional part of its program family.

What's great about the Synthesis Approach is that it's an approach that can change with the times. A next generation component to this approach will most surely be that of automated application development. Could it be that cooks will no longer be needed in the kitchen? Perhaps. However, until that time comes, we encourage you to create an environment that fosters the automation of code development and that follows the software Synthesis Approach. Both you and your users will be happy that you did.

The two most obvious benefits are lower development costs and lower maintenance cost. Another benefit of not lesser magnitude is in faster delivery of final products. As such, development using this approach grants us the ability to handle the enormous user backlog that MIS is facing today.

As always, MIS departments are under tremendous pressure to respond to rapid changes in the business environment. By embracing the Synthesis Approach, you will be able to meet these challenges by delivering products that are high in quality and meet your needs. Though the Synthesis Approach is new, it's possible that it could help solve some long standing problems such as integrating dissimilar types of data and assimilate new technology into old systems.

References

1. Raymond T. Yeh "Case for Rapid Application Development" CASE World Conference Proceedings. Spring 1991

2. William M. Ulrich "Re-Development Engineering : Formulating an Information Blueprint for the 1990's" CASE Outlook 90, No 2 1990

3. David M. Weiss "Synthesis : Integrating Product and Process" Software Engineering RICIS Symposium, 1990

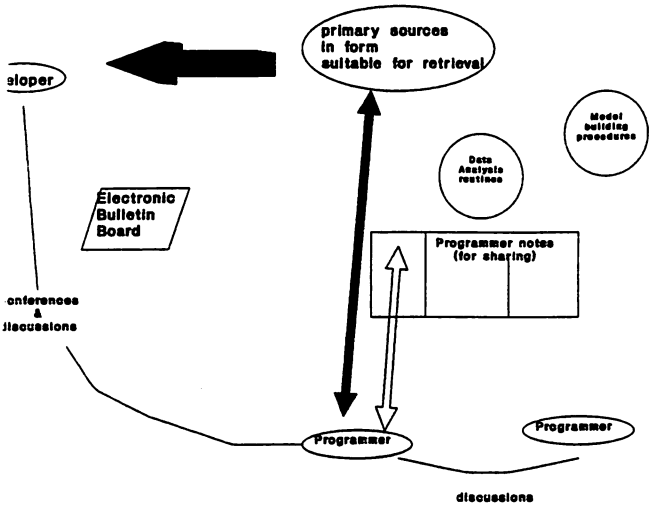

primary sources
in form
suitable for retrieval
eloper
Model
building
procedures
Data
Analysis
routines
Electronic
Bulletin
Board
Programmer notes
(for sharing)
onferences
&
iscussions
Programmer
Programmer
discussions

Fig 1-1

12/24/90 VENDOR PERFORMANCE BY PURCHASE ORDER FOR PERIOD 11/01 THRU 11/30/1990

PO NUMBER	PO DOLLARS	CLOSE DATE	COMPET PRICE?	% ACCEPT	ON TIME	DAYS LATE	SERVICE	C
3 COM CORP.								
02A0146266	9,563.00	11/02/90	N	100.00	Y	0	E	
VENDOR: 400530606 ----------		RATING ---						
1 PO'S :	9,563.00	100						
BICC-VERO ELECTRONICS INC.								
02C0149754	3,709.10	11/07/90	Y	100.00	Y	0	E	
VENDOR: 000072199 ----------		RATING ---						
1 PO'S :	3,709.10	100						
COMPUADD								
02C0144891	2,655.00	11/06/90	Y	100.00	Y	0	G	
VENDOR: 001263003 ----------		RATING ---						
1 PO'S :	2,655.00	100						
COMPURIZE								
02C0148920	2,677.00	11/01/90	Y	100.00	Y	0	E	
02C0149268	5,264.00	11/01/90	Y	100.00	Y	0	E	
02C0149269	7.00	11/01/90	Y	100.00	Y	0	E	
02C0149670	4,836.00	11/01/90	Y	100.00	N	13	G	
02C0149671	5,542.00	11/01/90	Y	100.00	Y	0	E	

Fig 3-1

FLIGHT DATABASE
MENU TREE

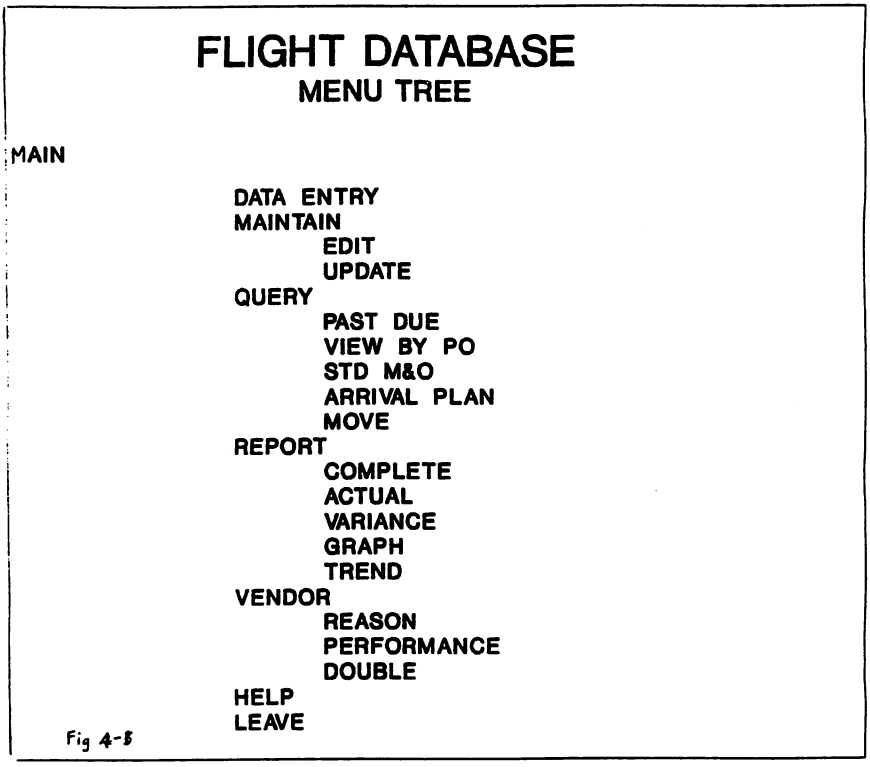


Fig 4-5

Paper #3248
HP 3000 Open CASE
Phiroze Petigura
Hewlett-Packard Company
19111 Pruneridge Avenue
Cupertino, CA 95014
(408) 447-6122

Introduction

Computer-Aided Software Engineering, or CASE, has been used by HP 3000 developers for many years to streamline development and maintenance of commercial applications. CASE addresses one of the most critical business needs today - the ability to quickly, and effectively manage a business' information assets. The HP 3000 open CASE program provides developers and value-added software businesses with CASE tools for developing and maintaining application for HP 3000 systems. The HP 3000 open CASE program provides support for methods and advanced CASE tools for the HP 3000 commercial MIS and value-added software developer communities.

The subject of this white paper is the HP 3000 commercial CASE program. It presents HP's strategy to extend CASE to meet the needs of new applications on HP 3000 systems. Although this paper primarily addresses the needs of in-house MIS departments, it applies equally to the needs of value-added software suppliers.

Objectives of HP 3000 Open CASE

The objectives of the HP 3000 open CASE program are to deliver to HP 3000 MIS and value-added application developers:

- the best-in-class CASE tools from leading CASE vendors,
- a multi-vendor CASE solution,
- CASE tools ranging from standalone to complete integrated-CASE solutions,
- CASE tools for both mainframe class and open systems, client/server applications.

HP recognizes that as the capabilities of the HP 3000 grows to meet mainframe and open systems standards, the breadth of developer requirements must also increase. Besides enhancing the current set of HP 3000 application development tools and protecting its customers' investment in existing development tools, these objectives broaden the HP 3000 CASE tools offering and addresses the application development needs of new, open systems and mainframe class applications.

Refer to Appendix A for a profile on HP 3000 developers.

The Benefits of CASE

To Corporate MIS Departments

As businesses become increasingly driven by information, the ability to easily deploy and modify its information systems can be a significant competitive advantage. Rapid deployment and modification of information systems give businesses the ability to quickly focus on new opportunities, offer a better level of service, expand, consolidate with other information systems, and keep pace with changing information management technologies. In today's competitive business environment, strength in information management is as crucial to an enterprise's success as strength in areas such as engineering, manufacturing, finance, or sales.

CASE provides this advantage by reducing the investment required to build and maintain new applications. CASE accomplishes this by providing methods and tools that streamline application development. Methods provide discipline to the application development process. For example, methods improve the quality, usability, and maintainability of applications by ensuring that adequate analysis and design are conducted in the early stages of a project and by ensuring the production of accurate project and product documentation. CASE tools assist in the implementation and maintenance of applications. CASE tools can, for example, alleviate much of the actual coding burden of programmers thereby increasing the speed of implementation and maintenance, and decreasing the likelihood of introducing coding errors.

CASE helps address the growing maintenance backlogs experienced by MIS departments. Research shows that today up to 75% of MIS resources are devoted to the maintenance of existing applications. Diverting MIS resources towards maintenance not only inhibits the ability for businesses to develop new information systems but also increases the cost of operations and decreases the morale of the programming staff. CASE tools and techniques can reduce the maintenance backlog by giving developers the ability to reverse engineer existing application code into new code that is easier to maintain or more compatible with new technologies.

To Value-Added Software Suppliers

CASE plays an important role for value-added software businesses. These companies, whose business is to develop, market, and support software business solutions, experience many of the same challenges as conventional MIS departments. In order to remain competitive, they must quickly adapt their products to new business practices, regulations, and standards. They must be capable of delivering enhancements and fixes in a timely manner. Value-added software suppliers must be able to adapt their product to the quickly changing technologies used by their customers. Today, for example, many software businesses are considering client/server computing technologies. For value- added software businesses, CASE provides a quick and inexpensive means to keep up with business trends and technologies.

CASE in the 1990's has become a major topic of interest and a pressing need among forward-looking developers from both MIS organizations and value-added software businesses. CASE has become as important to decision makers in the application development arena today as traditional system attributes such as price/performance, reliability, mass storage capacity, or support.

Refer to Appendix B for a discussion of CASE terms and concepts.

HP 3000 Open CASE Product Strategy

HP's strategy to achieve the objectives of HP 3000 open CASE is to:

- strengthen the current set of HP 3000 application development tools,
- deliver new, best-in-class, CASE tools from leading third-parties and HP,
- offer state-of-the art integrated-CASE tools.

Strengthen Current HP 3000 CASE Tool

The HP 3000 currently provides developers with a comprehensive set of high quality CASE and decision support tools that meet the needs of almost all HP 3000 applications. These tools are supplied not only by HP but also by leading third-party CASE tools vendors. Current CASE tools for developing applications for HP 3000 systems include analysis and design tools, construction tools (industry standard 3GLs, de facto standard 4GLs, report writers, decision support tools, industry standard database management systems, forms management systems), testing tools, and maintenance tools (symbolic debuggers, impact analysis tools, version control tools).

These CASE tools address all the major phases of the applications development lifecycle. Figure 1 shows each of these phases and the activities for each phase and figure 2 lists some of the major CASE tools available for each phase.

Figure 1. Major Activities for Each Phase of Development

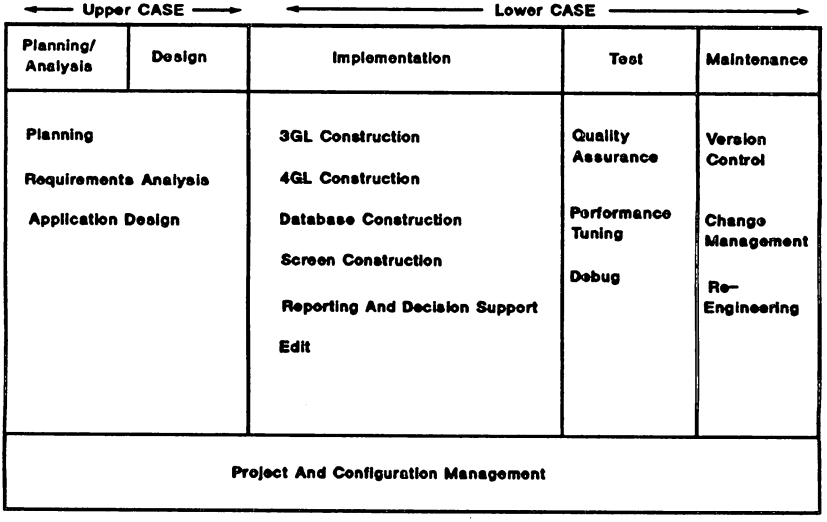
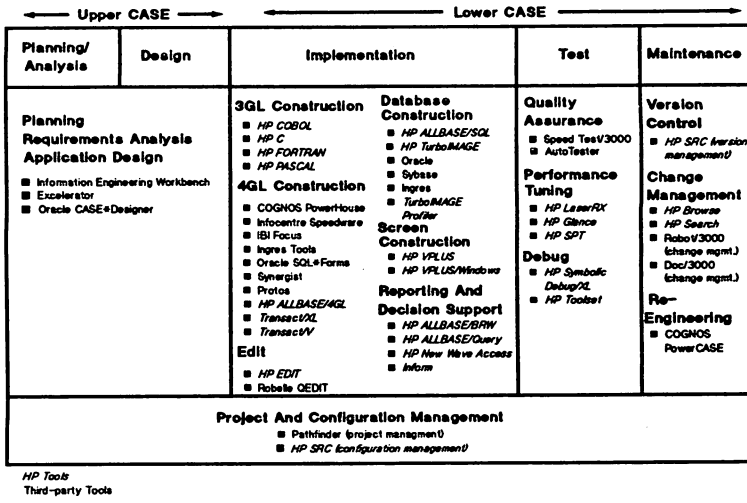


Figure 2. Major HP 3000 Tools Available Today



The tools listed in figure 2 form the core of the HP 3000 CASE offering. There are, in total, over 200 CASE tools from more than 120 vendors available for HP 3000 application development and maintenance. These tools will continue to meet the needs of the majority of HP 3000 application developers. HP will protect its customers' investment in these tools and continue to enhance the functionality and scope of these tools.

Refer to Appendix C for a description of some of these CASE tools.

HP's information management strategy is based around support for both of HP's strategic database management systems: HP TurboIMAGE and HP ALLBASE/SQL.

Enhancements To TurboIMAGE

TurboIMAGE is the highest performance, network model database management system on the HP 3000 and is widely used by MIS and value-added software developers to support performance critical OLTP applications. HP will protect its customers' investment in TurboIMAGE applications by continuing to improve TurboIMAGE performance to scale with the high performance HP 3000 systems.

Enhancements To ALLBASE/SQL

ALLBASE/SQL, is HP's ANSI standard, price/performance leading relational database management system. ALLBASE/SQL has gained wide acceptance for new, standards-based, OLTP applications. Recently, HP enriched its ALLBASE/SQL offering by announcing the availability of several leading third-party CASE tools for ALLBASE/SQL:

- Powerhouse (Cognos Corp.)
- Ingres Tools (Ingres Corp.)
- Focus (Information Builders Inc.), and
- Speedware (Infocentre Corp).

These leading third-party CASE tool vendors will support ALLBASE/SQL. This not only gives HP 3000 developers the choice of a wide range of tools for ALLBASE/SQL applications, but also lets them easily port between ALLBASE/SQL and other multi-vendor relational databases. For example, developers concerned with performance or data integrity would choose ALLBASE/SQL, while those needing access to multi-vendor systems would choose one of the leading multi-vendor databases. This flexibility gives an important advantage to value-added software businesses and MIS developers wishing to support a diversified installed base of systems.

In addition to more tools, HP has made enhancements to ALLBASE/SQL price/performance and functionality. ALLBASE/SQL has achieved tight integration with the PA-RISC platform to deliver the leading price/performance OLTP solutions for HP 3000 systems. Because of the tight integration, this price/performance leadership will scale with releases of high performance HP 3000 systems. HP has also enhanced the functionality of ALLBASE/SQL by providing connectivity to TurboIMAGE and DB2 databases through the ALLBASE/Turbo Connect and ALLBASE/DB2 Connect products, and connectivity to other ALLBASE/SQL databases through ALLBASE/NET.

HP has also enhanced ALLBASE/SQL to support the construction of windows-based, client-server applications. At present, HP supports an application programming interface (API) for HP 3000 ALLBASE/SQL servers and MS-Windows 3.0-based clients. HP will also support the industry-standard, client-server API for ALLBASE/SQL based on the standards set by the SQL Access Group.

Database Conversion Tools

HP will provide re-engineering tools that allow developers to convert TurboIMAGE applications into ALLBASE/SQL applications. CASE tools such as PowerCASE from Cognos will be able to convert TurboIMAGE applications to ALLBASE/SQL applications.

Best-in-Class CASE Tools Through Leading Third-Parties

The HP 3000 has become one of the most open commercial application development platforms available today by supporting a very wide variety of standalone CASE tools from leading third-party CASE vendors. As the quality and sophistication of the CASE tools increase, HP will broaden its focus to include more tools from third-party CASE vendors.

Third-party CASE tool vendors have led the development of new standards and tools for implementing client/server applications. In keeping with HP's New Wave Computing thrust to become the leading distributed computing vendor, HP will deliver the best open systems, client/server application development tools for HP 3000 systems as part of the HP 3000 open CASE program. Through its own investment, and through third-parties, HP will also deliver a broad offering of high quality CASE tools for open systems, host/terminal and mainframe class applications. The major new and existing CASE tools for HP 3000 systems are described below:

FOCUS

FOCUS is a leading 4GL and reporting tool from Information Builders Inc. and is one of the most widely used mainframe 4GLs. FOCUS runs on a large number of mainframe, midrange, and personal computer platforms and supports a variety of database management systems (DB2, IMS, ORACLE, Rdb, dBASE, etc.). FOCUS will be enhanced to support ALLBASE/SQL in the second quarter of 1991. FOCUS is suitable for new, mainframe class applications and porting of mainframe applications to HP 3000 systems.

Ingres/Windows 4GL

Ingres/Windows 4GL is a client/server 4GL that allows users to develop applications that use the HP 3000 as a database server and personal computers or workstation as clients. Ingres/Windows 4GL supports a variety of windowing interfaces such as MS- Windows, OSF/Motif, and Presentation Manager. Ingres/Window 4GL is suitable for open systems client/server applications. It will be available for HP 3000 systems and ALLBASE/SQL by the second quarter of 1991.

SQL/Windows

SQL/Windows from Gupta is a client/server application development tool. SQL/Windows runs under MS-Windows 3.0 and allows developers to build applications that use the HP 3000 as an ALLBASE/SQL server. SQL/Windows will give developers the ability to connect to DB2 and Oracle databases. SQL/Windows is ideal for OLTP, open systems client/server applications. It will be available in the second half of 1991.

PowerBuilder

PowerBuilder from Powersoft Corporation is a design and construction tool for client/server applications. Applications constructed with PowerBuilder will run in MS-Windows 3.0 and OS/2 Presentation Manager (PM) environments and will use the HP 3000 as an ALLBASE/SQL server. PowerBuilder will be available for HP 3000 systems in the second half of 1991.

Powerhouse 4GL

Powerhouse 4GL from Cognos Inc. is the leading high performance 4GL on HP 3000 systems. Powerhouse 4GL currently supports TurboIMAGE databases and will support ALLBASE/SQL databases in the second quarter of 1991. Powerhouse 4GL will be integrated with the Cognos' PowerCASE product in the second quarter of 1991 to provide design to code capabilities for HP 3000 systems.

SpeedWare

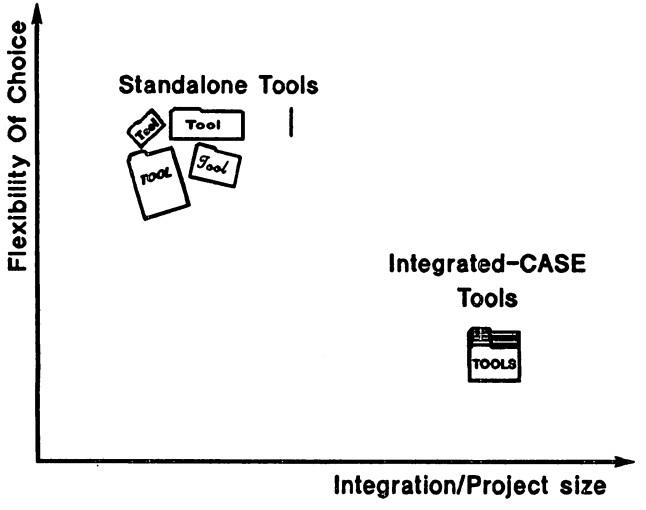
SpeedWare from Infocentre Corporation is a leading high performance client/server 4GL for HP 3000 systems. Speedware currently supports application development against TurboIMAGE databases and will support ALLBASE/SQL in the second quarter of 1991. Infocentre provides a high level application specification facility that supports rapid implementation of Speedware applications.

Integrated-CASE Tools for the HP 3000

Integrated-CASE, or I-CASE, tools consist of sets of individual tools, from single vendors, that aid applications development and support all phases of the application development lifecycle. I- CASE tools provide developers with the capability to automatically generate 3GL or 4GL applications from high level designs and to maintain applications at the design level. I-CASE tools are a single vendor solution built around a repository through which individual tools within the I-CASE toolset can share information about an application. This allows the I-CASE tool to propagate changes through an entire set of programs, databases, and screens from a single change to application design.

The difference between I-CASE tools and traditional, standalone CASE tools lie in overall functionality and flexibility of choice. Because of its tight integration and lifecycle orientation, I-CASE offers greater overall functionality and power to developers. However, I-CASE users can only choose from the limited number of tools available within the I-CASE toolset. Figure 3 illustrates the broad tradeoff between functionality and flexibility when moving from standalone CASE to I-CASE. Today, I-CASE tools form among the most powerful CASE tools available. I-CASE tools not only support entire development lifecycle methodologies but can also provide developers with the capability to manage large application development teams. I-CASE tools are used primarily for large, mainframe class applications.

Figure 3. CASE Tool Categories



Because of the complete lifecycle and methodology orientation of I-CASE, developers adopting I-CASE tools should have a long term commitment to the tool. The development organization, including users, programmers, and managers need to be trained in the new development methodologies, processes and tools supported by the I-CASE tool. Additional hardware and software such as personal computers and LAN-based servers may also have to be purchased to support the I-CASE tool.

HP recently announced the following leading I-CASE tools for HP 3000 systems:

PACLAN and PACLAN/X

The I-CASE tools from the French-based CGI Informatique includes PACBASE, PACLAN, and PACLAN/X. CGI is the world's largest I-CASE vendor. The CGI I-CASE tools operate in a client/server environment of personal computers and workstations on a LAN to a server-resident repository. The CGI products support application development for a large number of platforms and supports a variety of methodologies. The CGI products currently generate COBOL and TurboIMAGE applications for HP 3000 batch mode environments and will generate ALLBASE/SQL and VPLUS applications for HP 3000 systems by mid-1991. CGI also provides a reverse engineering facility called PACREVERSE to aid in the maintenance of existing COBOL code.

The CGI products are ideal for moderate to large COBOL development efforts ranging in size from about 4 to about 50 developers. CGI products are ideally suited for mainframe class developers and those wishing to downsize from mainframes to HP 3000 systems.

Maestro II

Maestro II is an I-CASE tool from Softlab Inc. of Germany. Maestro II runs in a client/server environment consisting of personal computers on a LAN to an HP 9000 server. Maestro II supports the connectivity and terminal emulation capabilities for connecting to, compiling, and testing applications on an HP 3000. It is customizable and will support any standard or custom development methodology. Maestro II is rich in project management and configuration management functionality, and supports application development for many platforms.

Maestro II is a high-end CASE environment that is ideal for very large projects with very large programming staffs. Maestro II will be available for the HP 3000 and will support COBOL, ALLBASE/SQL, and VPLUS in the second half of 1991.

PowerCASE

PowerCASE is an I-CASE tool from Cognos Corp. for developers of Powerhouse 4GL applications. PowerCASE is a PC-based graphical design tool that allows developers to design and generate Powerhouse 4GL, ALLBASE/SQL, and TurboIMAGE applications for HP 3000 systems. PowerCASE uses Entity-Relationship modeling and Data Flow Diagramming techniques to design applications, and supports the Cognos development methodology. PowerCASE can be used to migrate TurboIMAGE Powerhouse 4GL applications to ALLBASE/SQL Powerhouse 4GL applications

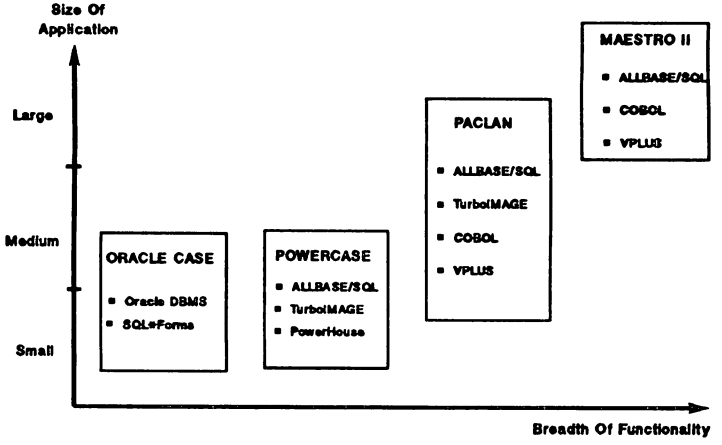
PowerCASE is suitable for small to medium Powerhouse 4GL developers. PowerCASE will be available for HP 3000 systems in the second quarter of 1991.

Oracle CASE

Oracle's CASE toolset includes CASE*Method, CASE*Dictionary, CASE*Designer, and CASE*Generator. This toolset only supports the ORACLE database, and generates SQL*Forms code from applications designed using CASE*Designer and CASE*Method. Oracle's CASE toolset runs on a large number of PC and UNIX workstation platforms and can target HP 3000 systems. Oracle's CASE toolset is available now.

Figure 4 shows how well the new HP 3000 I-CASE tools are suited to different application development situations.

Figure 4. New HP 3000 I-CASE Tools



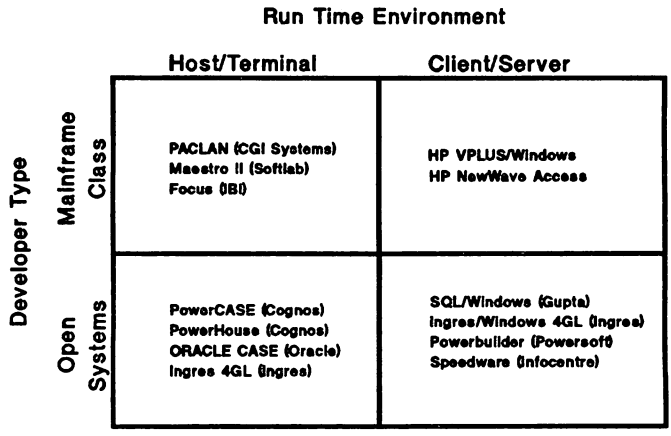
Summary

The HP 3000 open CASE program addresses the needs of both MIS and value-added software developers. The HP 3000 open CASE program has been successful in protecting the existing investment in HP 3000 CASE tools and in providing powerful, new standalone CASE, and integrated-CASE, tools for new applications development. This means application developers not only have the ability to maintain their existing applications using their tools of choice but also have the option to use new, best-in-class CASE tools to implement new, mission critical applications.

These new CASE tools not only meet the needs of today's applications but also provide the means to develop future applications for open and high-end HP 3000 systems. Figure 5 shows that these new CASE tools not only meet the needs of mainframe class and open systems developers but also the need for tools to develop host/terminal and client/server applications. From the preceding discussion on CASE tools, and analysis of the requirements of different application developers, it is clear there is no single superior CASE solution for all developers. Developer organizations must choose their CASE strategy in light of the size and complexity of their

application development projects and the ability and willingness of the organization to adopt certain CASE tools and practices. HP has therefore chosen to offer all HP 3000 developers a broad choice of best-in-class standalone CASE and I-CASE tools through the HP 3000 open CASE program.

Figure 5. New HP 3000 CASE Tools



Appendix A - Changing HP 3000 Developer Profiles

Although well-entrenched as a leading commercial midrange system, the HP 3000 is now positioned as both a mainframe and an open system. With the advent of the PA-RISC architecture, the HP 3000 now scales from the very low-end systems to the high-end multiprocessor mainframe class systems. HP's early thrust into distributed computing through its New Wave computing efforts has also made HP 3000 systems a leading open system.

In order to understand the need for CASE for HP 3000 systems it is necessary to understand the makeup of HP 3000 applications developers and how this has changed as the HP 3000 has grown into a mainframe class and open system. In addition to the traditional group of HP 3000 application developers, there is a growing need among both existing and new HP 3000 customers to address the development of mainframe class, open systems host/terminal, and open systems client/server applications.

Mainframe Application Developer

This group consists of developers whose size has made their applications development requirements characteristic of those of developers in the mainframe class environment. Applications developed by this group are typically large, business critical systems involving multi-person efforts which last several years. Once deployed, these systems require periodic rework and generally have a dedicated maintenance staff assigned to it. The main application development requirements of mainframe class developers are:

3GL Development

The high-end developer has relied primarily on traditional 3GL development tools for implementing systems. This toolset (COBOL, TurboIMAGE, SQL, and VPLUS) has typically yielded better performing applications than equivalent 4GL implementations. This performance advantage is important to this group because the business critical nature of their applications. Although some high-end developers are moving development to 4GLs, the greater part of high-end developers will continue to use 3GLs. CASE tools for these developers must therefore support the traditional development toolset.

Project Management

These developers have a need for CASE tools to control and manage the applications development process. As high-end developers typically develop and maintain large projects involving teams of programmers their requirement of CASE tools extend beyond simply reducing the investment necessary to construct and maintain applications.

The high-end developer needs CASE tools that assist in the management of the development and maintenance processes. This ranges from the enforcement of methodologies to the close tracking of resources expended on constructing an application.

Configuration Management

Large project often require configuration management tools. These tools facilitate the building and distribution of the application. Large projects are typically divided and given to different teams of programmers to implement. Configuration management tools to ensure that all the correct versions of the software are included in any given version of the application.

Platform-Independent Tools

The high-end developer often needs to contend with a multi-vendor computing environment. Although applications may originally have been developed and deployed on specific systems, as these accounts move forward towards enterprise wide computing there is a growing need to deploy the same application on the different systems. High-end developers therefore need access to the same CASE tools for the HP 3000 as are available for their IBM compatible mainframes to simplify migration to HP 3000 systems.

Application Generation

Application generation is the ability to automatically produce the code for an application from high level specifications. It eliminates the need to write the code, build the databases, or implement the screens necessary for the application. Application generation improves the quality and speed of implementation, and speeds up maintenance by giving programmers the capability to make changes to the application at the design level and re-generate the application. Application generation also facilitates reuse of designs and code.

Client/Server Migration

As personal computers become more prevalent in the mainframe class environment, there will be a growing need to take advantage of their power and user friendliness. As the availability of personal computers among users increases, mainframe class application developers will require tools to migrate existing applications into client/server applications to better utilize their personal computers. HP VPLUS/Windows is an example of a tool for migrating terminal-based VPLUS applications to a MS- Windows-based, client/server environment.

Open Systems Developer

Unlike the high-end developer, the open systems developer consists of smaller developers concerned with developing and deploying portable applications for an open, standards-based environment. Applications developed by this group typically make use of industry standard 3GL's and 4GL's, database management systems, forms management systems, and operating system services.

The majority of HP's value-added software businesses are open systems developers. The main requirements of the open systems applications developer are:

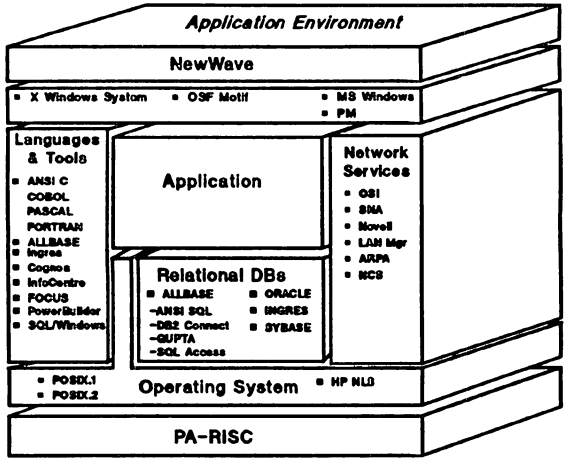
Support For Standards

Open systems CASE tools must support coding to these standards. Applications developed by the open systems developer must be platform portable by virtue of adherence to official or de facto standards. Figure 6 shows the standards for open systems applications.

Client/Server 4GL

Besides the ability to develop open systems host/terminal applications, the open systems developer needs the capability to develop client/server applications. Although client/server is not a new concept, implementing client/server applications has been difficult. In order to construct client/server transaction processing applications, developers not only need to be expert on both the client and server platforms but also need to know how to manage the networking, and the graphical user interfaces of the client. High level languages, such as 4GL's, facilitate the construction of client/server application by shielding the developer from many of the details of construction.

Figure 6. Standards for Open Systems Applications

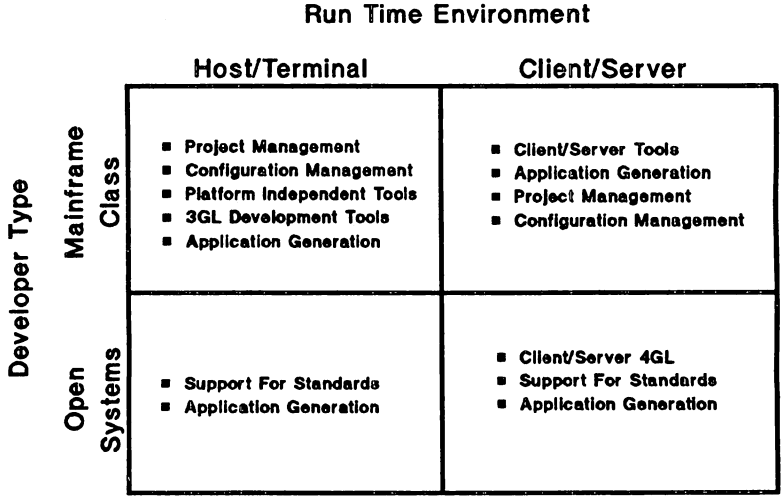


Application Generation

Like the mainframe class application developers, open systems application developers will want tools that automate implementation.

The CASE requirements for each group are shown in figure 7.

Figure 7. Application Development Requirements



Appendix B - What is CASE

CASE refers to the tools and methods that increase the quality of applications and decrease the investment necessary to develop and maintain applications. CASE applies equally to commercial applications development or technical software engineering projects. The differences between commercial and technical CASE lie primarily in the methods and tools used. This discussion is confined to commercial CASE.

CASE Tools

CASE tools aid and automate the design, implementation, and maintenance of applications. CASE tools support every phase of applications development including planning and analysis, design, implementation, test, and maintenance. CASE tools range from PC- based graphical design tools to server-based version control and archiving tools.

CASE tools are not new to applications development. Tools such as 4GLs, application generators, version management systems, symbolic debuggers, etc. have been used for many years. Many of these tools were originally developed as in-house productivity tools and found initial acceptance by small projects and small MIS departments that needed to leverage limited programming resources. However, as a result of pressures to trim costs and improve the responsiveness of MIS to changing business needs and technologies, more and more mainstream MIS departments have begun to use CASE tools as the means to achieve this.

The development of CASE tools has made rapid progress in recent years. With wider acceptance and usage of tools, the CASE vendor community has seen the emergence of standards in areas such as import and export specifications and diagramming conventions. The CASE industry has seen the emergence of CASE companies that have taken leadership positions in the market. Canadian and US companies such as Cognos, Infocentre, and Knowledgeware, and European companies such as CGI Systems, and Softlab have become leading CASE vendors.

Upper and Lower CASE

CASE tools are frequently referred to as either upper CASE or lower CASE tools. This distinction refers to the phases of the lifecycle that different tools address. Upper CASE addresses the planning, analysis, and design phases while lower CASE addresses the implementation, testing, and maintenance phases. In general, upper CASE tools are graphical in nature, employ diagramming or charting techniques, and often run on personal computers or workstations. The output of upper CASE tools are application designs applicable to most target system. Lower CASE tools do not require graphics capabilities and are often deployed on a server or multiuser system.

Methods

CASE methodologies refers to the discipline adhered to during the process of developing applications. Methodologies help ensure the quality of the final application by identifying each step in the development of an application and formalizing the activities, standards, and checkpoints that need to be adhered to. Formal methodologies such as Information Engineering, Yourdon, SSADM, and others have been used for many years with large application development

projects and by mainstream MIS organizations. These methodologies have not, however, seen widespread acceptance among smaller developers because of the high cost of implementation and training associated with their use.

In practice, CASE tools and methodologies are highly interdependent. The combined use of methodologies and CASE tools gives application developers the powerful capability to not only automate implementation but also enforce the process. Recent developments in CASE has seen the emergence of sophisticated CASE tools that can be configured to support different methodologies. These integrated-CASE tools not only represent a significant step towards making CASE address mainstream MIS needs but also reduces the cost of entry for smaller MIS organizations to use CASE.

Target and Development Systems

Descriptions of CASE tools often refer to target and development systems. Target systems are where the applications will run in production. Development systems are where applications are developed. In many instances, target and development systems are the same. Applications are developed and deployed on the same system because it gives developers the ability to custom build the application for optimal performance on the production system.

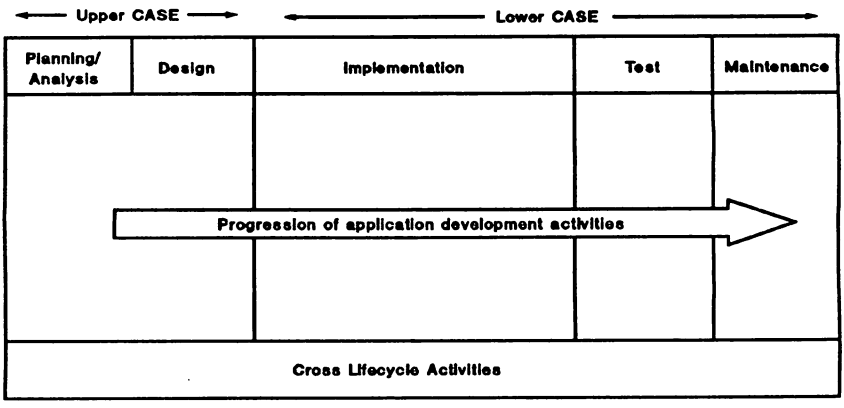
Recently target and development systems have begun to diverge. As the price/performance of production systems decreases and the specialized computing needs of CASE tools increases, the choice of development and target systems becomes driven by different constraints. Development systems are often selected for features such as the ability to support graphics and for individual productivity. For example, DOS-based personal computers and UNIX workstations are often selected as development platforms because their sophisticated graphical displays and dedicated computing power makes them ideal for today's advanced CASE tools. On he other hand, target systems are often selected because of performance, reliability, data integrity, system security, systems management, and other production related features.

The division between target and development systems have brought about a new class of target system independent CASE tools which permit the development of applications for a variety of systems from single application specifications.

Lifecycle Framework

CASE tools are often referred to in the context of how they address different phases of the application development lifecycle. Each phase corresponds to a set of activities that are undertaken for applications development. These phases are planning and analysis, design, implementation, test, and production and maintenance. These are often depicted in a lifecycle framework like the one shown in figure 8.

Figure 8. Application Development Lifecycle Framework



Planning and analysis

These activities occur at the very beginning of the application development process and involves the analysis and mapping of a company's business goals to its information systems plans.

Design

This phase translate the information system needs identified in the previous phase into high level data and process models. During this phase, the data and process needs of the information system under design can be evaluated at an abstract level before construction of the application occurs.

Implementation

This phase involves the building of the application. Tools used at this phase are 3GL's, 4GL's, database management systems, and forms management systems. More recently, report writers and code generators are also being used at this phase.

Test

This is the phase during which the application is tested for compliance to specifications and debugged. The tools employed during this activity are code analyzers, debuggers, and testing tools.

Production/Maintenance

This phase comprises activities performed after an application is deployed into production. This includes such tasks as implementing enhancements and bug fixes. Another, less well understood, task performed during this phase is software distribution to remote operations. For

centralized MIS departments, the task of ensuring that each remote production system has the latest software revision is often complex, expensive, and prone to error. The complexity increases for client/server applications where it is necessary to distribute to each client system. The tools most often utilized during this phase are source management, source analysis, and reverse engineering tools.

Cross Lifecycle

These are the activities, such as project management and configuration management, that occur across all the phases of application development.

Appendix C - Major Hewlett-Packard CASE Tools for the HP 3000

HP COBOL II

HP COBOL II is an optimizing compiler for 1985 ANSI Standard COBOL code. COBOL is an ideal 3GL for batch and OLTP applications.

HP VPLUS

HP VPLUS is a forms management system for all HP block mode terminals. HP VPLUS intrinsics are callable from most programming languages. HP VPLUS is optimized for performance on HP 3000 systems.

HP VPLUS/Windows

HP's VPLUS/Windows is a client/server application development tool, and migration tool for existing VPLUS applications. VPLUS/Windows permits host/terminal VPLUS applications to run as client/server applications under MS-Windows 3.0 and New Wave. VPLUS/Windows runs in a PC-LAN environment connected to one or more HP 3000 systems or servers.

HP New Wave Access

New Wave Access is a client/server decision support tool that runs under MS-Windows 3.0 and New Wave. New Wave Access allows end users to graphically extract and manipulate data from many sources (ALLBASE/SQL, TurboIMAGE, DB2 and Oracle). New Wave Access lets end user offload programmers by defining and generating their own decision support applications.

HP Transact/XL

Transact/XL is a high-level programming language for developing OLTP applications for TurboIMAGE databases and VPLUS forms. The Transact/XL compiler is available for all HP 3000 Series 900 systems. Transact/XL applications require fewer lines of code to implement and results in reduced development and maintenance costs.

HP ALLBASE/4GL

ALLBASE/4GL is an advanced 4GL for developing OLTP applications for ALLBASE/SQL and TurboIMAGE databases. ALLBASE/4GL gives developers the ability to rapidly prototype applications using the ALLBASE/4GL screen painter.

HP ALLBASE/BRW

HP ALLBASE/BRW is a high performance reporting tool for MIS professionals and sophisticated end users. It supports ALLBASE/SQL and TurboIMAGE databases.

HP ALLBASE/Query

HP ALLBASE/Query is an easy-to-use end user decision support and reporting tool. It gives end users decision makers the ability to extract, format, and report data resident in ALLBASE/SQL and TurboIMAGE databases.

HP Information Access

HP Information Access is a PC-based decision support tool that gives decision makers the ability to extract and report data resident in ALLBASE/SQL, TurboIMAGE, and DB2 databases.

HP GlancePlus/XL

HP GlancePlus/XL is a diagnostic and performance monitoring tool that gives programmers the ability to tune the performance of their application by identifying its performance bottlenecks.

HP Software Performance Tuner/XL (SPT)

HP SPT is a tool that provides information on the efficiency of an application's algorithms and code.

HP Symbolic Debug/XL

HP Symbolic Debug/XL is a on-line debugger for all HP 3000 applications. It gives programmers the ability to track memory- resident values using symbolic names instead of relative memory addresses. It also allows programmers to track and debug code paths.

HP EDIT/XL

HP EDIT/XL is a full function, screen oriented editor for application developers. It supports standard and COBOL line numbering and has features such as automatic indentation and line shifting. HP EDIT/XL has comprehensive on-line help.

HP Software Revision Controller (SRC)

HP SRC is a sophisticated version management and configuration management system design to control changes to program code and other files. HP SRC offers check-in/check-out facilities to guarantee the integrity of the files it manages. HP SRC also maintains an audit trail of changes made to a file.

HP Search/XL

HP Search/XL is a general-purpose search utility for quickly finding the occurrence of words or patterns in files or groups of files. HP Search/XL supports searches against wildcard characters and patterns stored in other files. HP Search/XL can search up to 65,000 files at a single time.

HP Browse/XL

HP Browse/XL is a full-screen, interactive utility for programmers to quickly examine and print the contents of files or search for patterns without having to use a standard editor or reporting tool. HP Browse/XL supports windowing to access two files simultaneously.

Paper #: 3902

The Pros and Cons of Prototyping

George Federman
George Federman & Associates
6236 Parkhurst Drive
Goleta, CA 93117
(805) 683-3037

In many disciplines, paper plans and working models go hand-in-hand in designing new structures and systems. In systems analysis and design, prototyping offers us the same use of working models. However, in many data processing environments, the prototyping approach is rejected out of hand.

The reasons are understandable. After many years of fighting for structured analysis and structured design, few data processing managers or project leaders would welcome a code-it-now, fix-it-later philosophy. Prototyping looks like that philosophy under a new name.

However, if prototyping is seen as augmenting rather than replacing a structured systems development life cycle, and seen as a tool for inquiry and modelling rather than seat-of-the-pants programming, its virtues become much clearer.

In the requirements phase of systems analysis, prototyping can elicit user needs, and confirm our understanding of their needs, better than any other technique.

In the physical design phase, it offers users a hands-on model of the system, a model they can manipulate and question. With prototyping, we get immediate user feedback, and the chance to quickly correct or improve designs based on that feedback.

There are drawbacks. Prototyping only works in some situations, and not in others, and we have to know when to prototype. Prototyping may lead us to select a design and begin coding sooner than we should. Prototyping encourages user response and iterative modification; sometimes we end up in an infinite loop, forever tweaking the model. Prototyping, like all methodologies, can be

carried to extremes, and sometimes the working model is delivered as the final system, with critical edits, controls, and procedures still missing.

Any design technique has its benefits and drawbacks, and prototyping is no exception. But properly used, it offers us remarkable opportunities for improving and accelerating system design.

Paper # 3905
Using HP's "F" words and gain control of your sequential files

Rick Roberts
Standard-Thomson Corp.
152 Grove Street
Waltham, MA. 02154
(617) 894-7310

What are HP's "F" words ?

In the context of this presentation, they are the intrinsic calls for managing files. Some examples would be:

FOPEN FREAD FWRITE FDELETE FSPACE FCLOSE

They all begin with the letter "F".

Why use intrinsic file calls ?

When I first started working on an HP I was surprised by how ineffectively they handled sequential files. Maybe HP felt sequential files were a thing of the past, with KSAM files and Image database files being the future. Sometimes simplicity can be the elegant and effective solution.

What can you possible do with intrinsics ?

Lots ! I was surprised once I started researching intrinsics by what could be done and how effective they were. We will discuss:

Creating/Finding variably named files
Sizing the file (other than the default size)
Checking if a file exists; then delete or add
Deleting a file
Renaming a file
Delete logical records (and they said it couldn't be done)

All this can be done from within a program, where the logical flow of a system can best be managed.

Before I describe the actual code to do this, I will explain some practical instances where I have used file intrinsics, when nothing else would do.

[CASE 1] My first experience with file intrinsics came when our company had a need to create order files quickly. We felt that using image datasets directly would slow the process. Most of our sales were taken over the phone and during the hectic ordering seasons we wanted to provide the fastest possible way to serve our customers.

We felt that if we created sequential files with variable names, we could then have a batch program running in a lower priority queue to add them to the Image order sets. We used an Image master set to keep a control number for file names and added that number on to a prefix to generate a file name. The batch program then read through a range of file names to generate orders. We used a prefix of "WO" to indicate a work order. We discovered that by doing this we also could use "BO" as a prefix to indicate a backorder on the original order and "HO" to indicate an order on hold.

Once we began using the system we found that our telephone sales people could create orders faster than the batch file could add them. Our customer service was improved and the system worked well.

[CASE 2] In another instance we needed to create requisitions for parts. The planners were thumbing through large MRP reports to determine what needed to be ordered. What we wanted to provide the planners, was a file of recommended parts and the dates they needed. The generation of requisitions took place over the period of a week and we wanted to be able to delete records that had already been processed so the planners only viewed what hadn't been completed yet.

We used intrinsics to name the files according to the part's class code which was unique to a planner. We made the sequential file an RIO (Relative I/O) file and were able to "FDELETE" logical records. These were large volatile files that would wreak havoc on an Image dataset but were handled quite easily by using sequential files.

[CASE 3] Another situation involved files that we use for our executive information system. These were small files containing summary information on the status of different aspects of our business. Daily Sales, Daily Shipping Statistics, Inventory Levels, Status of our Production Plan.

These files were all snapshots of the status on specific days, so we incorporated the date into the file name. In some cases we accumulated month-to-date and year-to-date information. By using file intrinsics we could "step back" from today's date to find the previous days file for accumulation information. If anyone viewing the information wanted to see the status for a previous date, they entered the date and the information was displayed quickly and easily.

CODE FOR USING INTRINSICS

```
CONFIGURATION SECTION
SPECIAL-NAMES.
     CONDITION-CODE IS ERROR-FIELD.

WORKING STORAGE SECTION

01  DEVICE          PIC X(08) VALUE "DISC".
01  FNAME1.
     05  F1-PRE     PIC X(02) VALUE "WO".
     05  F1-NUM     PIC 9(06).
     05  F1-LOC     PIC X(17) VALUE "group.acct".
01  FNAME2.
     05  F2-PRE     PIC X(03) VALUE "REQ".
     05  F2-CODE    PIC 9(04)B.
01  FNO1            PIC S9(4) COMP.
01  FNO2            PIC S9(4) COMP.
01  FSIZE1          PIC S9(9) COMP VALUE 20000.
01  RECSIZE1        PIC S9(4) COMP VALUE -94.
01  RECSIZE2        PIC S9(4) COMP VALUE -48.
01  ECODE           PIC S9(4) COMP VALUE 0.
01  SECCODE         PIC S9(4) COMP VALUE 0.
01  CCNTRL          PIC S9(4) COMP VALUE 0.
01  DISP-0          PIC S9(4) COMP VALUE 0.
01  DISP-1          PIC S9(4) COMP VALUE 1.
01  DISP-2          PIC S9(4) COMP VALUE 2.
01  DISP-3          PIC S9(4) COMP VALUE 3.
01  DISP-4          PIC S9(4) COMP VALUE 4.

01  IN-REC1.
01  IN-REC2.

     this area would contain the layout for your files.
```

FIGURE 1

Figure 1 shows the necessary data fields for this processing. Some of these fields will be discussed later when I cover the processing code.

CONDITION-CODE must be defined under special-names and is returned after every intrinsic call. Error-field is not defined anywhere else in the program and is the data name referred to for error checking.

DEVICE will always be DISC since we are dealing with sequential files.

FNAME1 is the parameter that tells the call what file to "FOPEN". In one example we have set the prefix to "WO". The file we are creating or finding will begin with WO. The second part of the file is defined to have six digits. This is the variable "name" part of the file. The last field contains the group and account. It should be noted that the filename parameter must be terminated by a nonalphanumeric character other than a slash (/) or period (.). I use a blank.

FNO1 is the parameter that is passed back by the "FOPEN". After the file is opened any further references to the file will use this system assigned file number.

FSIZE1 is the file size (number of records). If this parameter is not used the default is 1024. You can make this parameter larger than necessary and then "shrink" the file to exact size with an "FCLOSE" parameter.

RECSIZE1 is the length of each record. A negative number indicates the number of bytes; if positive, it is the number of words.

IN-REC1 is the area where the file record is placed when it is "FREAD" and should contain the file layout.

The rest of the parameters will be explained later.

This is easy!! Let's go on to the processing involved.

The first step is to open the file. FIGURE 2 shows some examples of "FOPEN".

Most of the parameters have already been discussed. The end of the call has the statement, GIVING FNO1. This is the system assigned file number that is used in all subsequent calls to this file.

There are two new parameters in the call; both represented by octal numbers (preceded by a % sign). These are the "F" (file) options and the "A" (access) options. These two options are key in describing the file and the source of most errors when using file intrinsics.

EXAMPLES OF "FOPEN"s

```
CALL INTRINSIC "FOPEN" USING FNAME1, %4, %101,
     RECSIZE1, DEVICE, GIVING FNO1.

CALL INTRINSIC "FOPEN" USING FNAME1, %5, %104,
     RECSIZE1, \\ \\ \\ \\ FSIZE1, GIVING FNO1.

CALL INTRINSIC "FOPEN" USING FNAME1, %1, %1300,
     RECSIZE1, DEVICE, GIVING FNO1.

CALL INTRINSIC "FOPEN" USING FNAME1, %10004, %4,
     RECSIZE1, DEVICE, GIVING FNO1.
```

FIGURE 2

EXAMPLES OF "FCLOSE"s

```
CALL INTRINSIC "FCLOSE" USING FNO1, DISP-0, SECCODE

CALL INTRINSIC "FCLOSE" USING FNO1, DISP-1, SECCODE

CALL INTRINSIC "FCLOSE" USING FNO1, DISP-2, SECCODE

CALL INTRINSIC "FCLOSE" USING FNO1, DISP-3, SECCODE

CALL INTRINSIC "FCLOSE" USING FNO1, DISP-4, SECCODE

CALL INTRINSIC "FCLOSE" USING FNO1, %11, SECCODE
```

FIGURE 3

	POS 1	POS 2	POS 3	POS 4	POS 5
0	STD	ALLOW :FILE	NO CCTL FIXED	FILENAME	BIN NEW
1	RIO		NO CCTL VARIABLE	$STDLIST	BIN PERM
2	CIR	NO :FILE	NO CCTL UNDEF.	$NEWPASS	BIN TEMP
3	MSG	NO :FILE	NO CCTL SPOOL	$OLDPASS	BIN TEMP/PERM
4		KSAM :FILE	CCTL FIXED	$STDIN	ASCII NEW
5			CCTL VARIABLE	$STDINX	ASCII PERM
6			CCTL UNDEF.	$NULL	ASCII TEMP
7			CCTL SPOOL		ASCII TEMP/PERM

FIGURE 4 "F" options

	POS 1	POS 2	POS 3	POS 4
0	WAIT NON MULTI	BUF DEFAULT	NO FLOCK NO MULTI REC	READ ONLY
1	WAIT INTRA-JOB	BUF EXCLUSIVE		WRITE ONLY
2	WAIT INTERJOB	BUF SEMI READ	NO FLOCK MULTI RECORD	WRITE (SAVE) ONLY
3		BUF SHARE		APPEND ONLY
4	NO WAIT NON MULTI	NO BUF DEFAULT	FLOCK NO MULTI REC	READ / WRITE
5	NO WAIT INTRA-JOB	NO BUF EXCLUSIVE		UPDATE
6	NO WAIT INTER-JOB	NO BUF SEMI READ	FLOCK MULTI RECORD	EXECUTE
7		NO BUF SHARE		

FIGURE 5 - "A" options

FIGURES 4 and 5 show the combinations of "F" and "A" options. Though there are many combinations, in actual practice you will use very few. The left most column contains the value you would use in the position across the top. If the position is not specified, the defaults (0 row) are in effect.

In FIGURE 2 most of the "F" options use the defaults in the first four positions. They are:

STANDARD SEQUENTIAL FILE
ALLOWS :FILE OVERRIDE
NO CARRIAGE CONTROL
FIXED LENGTH RECORDS
USE THE FORMAL FILE DESIGNATOR (FNAME1)

The exception to this is example four. It is described as an RIO (Relative I/O) file. This allows for the deletion of logical records.

Position five is where the most variation is found.

EXAMPLE 1 is a NEW ASCII file (%4)
EXAMPLE 2 is a PERMANENT ASCII file (%5)
EXAMPLE 3 is a PERMANENT BINARY file (%1)
EXAMPLE 4 is a NEW ASCII RIO file (%10004)

The "A" options (FIGURE 5) describe the type of access.

Position 1 sets two options. The first is the NOWAIT option. You must be running in privileged mode to use NOWAIT and control is returned before completion of the I/O. I would recommend you not use NOWAIT. The second part is MULTI access. This would allow processes located in different jobs or sessions to open the same file. The default is NO MULTI access. In the third example we have set this to INTRA-JOB access.

Position 2 also sets two options. The first is the BUF option. BUF allows for normal MPE buffering of records. NOBUF allows for physical block transferring. I have always used the BUF option. (Let MPE do the work). The second option is the type of access. The most common are EXCLUSIVE (1) and SHARE (3). In example four I have used the default. The default is dependant on the option selected for position 4. If READ ONLY is set, then the option is SEMI-READ. All other settings for position 4 result in EXCLUSIVE access (1).

Position 3 sets locking and multi record access. Normally the default of NO FLOCK and NO MULTI READ is used. Most of the files I have used are not updated by multiple sessions. There is usually a singular batch program that will update them and no locking is required. The MULTI RECORD option would allow you to read in more than one logical record depending on the file size parameter used in the call. Don't tempt fate, stick to one record at a time.

Position 4 declares how the file will be used. Most times it is READ ONLY, WRITE ONLY, or READ/WRITE. UPDATE would imply READ/WRITE and is used occasionally. APPEND ONLY implies WRITE ONLY access, but starts at the end of file marker.

These options may seem confusing, but by reading the manual and experimentation you should get a feeling for how these options make your file react.

The next step is "FCLOSE". This tells the system what to do with your file when you are done. Figure 3 shows all the possible "FCLOSE"s. Note that FNO1 is now used to reference the file. The last parameter is the security code for the file. I have always used 0, which allows unrestricted access. You can tell I'm a liberal. If you use 1, the file can only be accessed by its creator. This is for the power hungry. The second parameter is the disposition.

DISP-0 No change. The file remains as it was before it was opened.
DISP-1 Saves the file as a permanent file
DISP-2 Saves it as a job/session temporary file (rewind)
DISP-3 Same as DISP-2 with (no rewind)
DISP-4 The file is deleted from the system. I have sometimes named this parameter "POOF"

There is another option in the disposition parameter of "FCLOSE". This option allows for the return of unused disc space to the system. If the file is opened with a large file size it could be "shrunk" by using a %1 in front of the other half of the parameter. A %11 would make the file a permanent file and return space to the system.

Figure 6 is an example of how you can check if a file exists, delete it if it does, and then open it as a new file.
The first call opens the file as a permanent ASCII file.
The next sentence checks the condition code after the open. On all intrinsic calls the ERROR-FIELD is set to one of three conditions. ERROR-FIELD is set to 0 if the call was successful. The other two, greater than 0 and less than 0, indicate an error condition.

In this example we have checked for ERROR-CODE being equal to 0. Since the "FOPEN" was for an existing file, this tells us the file does exist. In this example it should not exist. By using an "FCLOSE" with a disposition of %4, the file is deleted. Next the file is opened as a new ASCII file and the processing would continue.

[CASE 1] Figure 7 shows how the processing would be handled by the batch file looking for orders.
The INIT paragraph determines the range of file names you wish to cover. It would be customized to each situation, but here I have stored a beginning number and the number of reads I want to execute.

In the Perform statement the variable I becomes the beginning file number and is varied until the last file number is reached.
In the PROCESS paragraph the file number (I) is moved to the variable portion of the file name. Remember they are prefixed by "WO". The file is then opened as if it existed (%5). If it doesn't exist, who cares, go get the next potential file. If it does exist, then go add the order and close the file with DISP-4 to remove it.

```
CALL INTRINSIC "FOPEN" USING FNAME1, %5, %101,
     RECSIZE1, DEVICE, GIVING FNO1.
IF ERROR-FIELD = 0
     CALL INTRINSIC "FCLOSE" USING FNO1, DISP-4,
                                   SECCODE
END-IF.
CALL INTRINSIC "FOPEN" USING FNAME1, %4, %101,
     RECSIZE1, DEVICE, GIVING FNO1.
```

FIGURE 6 - Sample "FOPEN" "FCLOSE" Combination

```
INIT.
  get BEG-NO
  ADD TOT-READS TO BEG-NO GIVING END-NO.
  PERFORM PROCESS VARYING I FROM BEG-NO BY 1
          UNTIL I > END-NO

PROCESS.
  MOVE I TO F1-NUM.
  CALL INTRINSIC "FOPEN" USING FNAME1, %5, %100,
     RECSIZE1, DEVICE, GIVING FNO1.
  IF ERROR-FIELD = 0
     PERFORM some work
  ELSE
     who cares.
```

FIGURE 7 - [CASE 1]

```
01  REC-CNT   PIC S9(4) COMP.
01  FDISP     PIC S9(4) COMP.


    MULTIPLY REC-CNT BY -1 GIVING FDISP.
    CALL INTRINSIC "FSPACE" USING FNO1, FDISP.
    MOVE 1 TO FDISP.
    PERFORM DELETE REC-CNT TIMES.

DELETE.
    CALL INTRINSIC "FDELETE" USING FNO1.
    CALL INTRINSIC "FSPACE" USING FNO1, FDISP.
```

FIGURE 8 - [CASE 2]

[CASE 2] In case 2 I mentioned being able to delete records. Figure 8 shows the pertinent code.

Up to this point the file has been opened as an RIO file, the part requested by the planner has been displayed, and he has processed it as a requisition. Under the program specifications I want to delete the work already completed.

When the records were first read, they were counted and I kept that count in REC-CNT. Now we need to delete those records, except the file pointer is at the end of the processed records. We can use the "FSPACE" intrinsic to point to the beginning of the records. By negating REC-CNT we can use this to space BACKWARD. The first call to "FSPACE" does this.

The next step is to put the displacement value to 1. Now the DELETE paragraph is performed once for each record to be deleted.

The call to "FDELETE" deletes the logical record. For some mysterious reason (known only to HP) the file pointer is not moved by an "FDELETE"! Therefore I used "FSPACE" with a value of 1 to move the pointer forward allowing the next "FDELETE" to delete the next record.

```
01  FILE-SWITCH   PIC X.
      88  FILE-FOUND    VALUE "Y".
01  J              PIC S9(4) COMP.
01  SAVE-DATE.
      05  S-MO      PIC 99.
      05  S-DY      PIC 99.
01  FNAME1.
      05  F1-PREF   PIC X(03) VALUE "DSR".
      05  F1-MD     PIC X(04)B.

FIND-LAST.
        set save date to todays' month-day
  MOVE "N" TO FILE-SWITCH.
  MOVE 0 TO J.
  PERFORM OPEN
      UNTIL FILE-FOUND OR J > 40.
  IF J > 40
      DISPLAY "File not Found E-O-J"
      GO TO PACK-IT-IN
  ELSE
      process file.

OPEN.
  ADD 1 TO J.
  MOVE SAVE-DATE TO F1-MD.
  CALL INTRINSIC "FOPEN" USING FNAME1, %5, %104,
      RECSIZE1, DEVICE, GIVING FNO1.
  IF ERROR-FIELD = 0
      MOVE "Y" TO FILE-SWITCH
  ELSE
      SUBTRACT 1 FROM S-DY
      IF S-DY < 1
            MOVE 31 TO S-DY
            SUBTRACT 1 FROM S-MO
            IF S-MO < 1
                  MOVE 12 TO S-MO.
```

FIGURE 9 - [CASE 3]

[CASE 3] Figure 9 shows the code for the third case. I needed to search backward for a file with the date as part of the name.

The FIND-LAST paragraph sets up the necessary parameters for the recurring "FOPEN" statement. The counter J is used in case someone keys in a first century date and the program gets "lost in space" and runs forever.

The OPEN paragraph tries to open a file as an existing ASCII file (%5). If the file doesn't exist it changes the filename (date portion) to try again.

One is subtracted from the day and when zero is reached it moves 31 to the day and subtracts 1 from the month. If the month reaches zero, 12 is then moved to month.

What a great routine!

My favorite part is the fact I don't have to worry about how many days there are in a month. I start at 31 and work backward. If I don't find it, so what. If someone (another programmer) was careless enough to create a February 31 (0231 suffix) file, then I will find it. I get to be smart in this program. Of course if someone creates a February 32 (0232 suffix) file I will never find it, but neither will they!!

```
01  ECODE      PIC S9(4) COMP.
01  MSGB       PIC X(72) VALUE SPACES.
01  BLEN       PIC S9(4) COMP VALUE 0.


READ-A-FILE
  CALL INTRINSIC "FREAD" USING FNO1, IN-REC1,
RECSIZE1.
  IF ERROR-FIELD = 0
       process record
  ELSE
       IF ERROR-FIELD > 0
         MOVE "Y" TO END-OF-INPUT
       ELSE
         CALL INTRINSIC "FCHECK" USING FNO1,
ECODE
         CALL INTRINSIC "FERRMSG" USING ECODE,
                  MSGB, BLEN
         DISPLAY MSGB.


WRITE-TO-A-FILE.
  CALL INTRINSIC "FWRITE" USING FNO1, IN-REC1,
           RECSIZE1, CCNTRL.
  IF ERROR-FIELD = 0
       keep processing
  ELSE
       IF ERROR-FIELD > 0
            DISPLAY "File Full"   OOPS !!
       ELSE
            CALL INTRINSIC "FCHECK" USING FNO1,
                     ECODE
            CALL INTRINSIC "FERRMSG" USING ECODE,
                     MSGB, BLEN
            DISPLAY MSGB.
```

FIGURE 10 -[ERROR ROUTINE]

The last two calls are "FREAD" and "FWRITE". Lest you think I was running the UTOPIA operating system, I have included the all-purpose error routine in FIGURE 10.

The "FREAD" and "FWRITE" are pretty straightforward. They use the system assigned file number and the record size. The other parameter is the working storage area where the record is stored. In the "FWRITE" call there is also a carriage control parameter that would be used for print files. It is set to zero for sequential files.

If the ERROR-FIELD is not equal to 0 you can call "FCHECK" using the file number for the call. The returned error code (ECODE) is then used in the "FERRMSG" call to obtain a description of the error. This is for all of us that don't have the error codes memorized. The description is returned to the second parameter (MSGB in this case). You can then display the error and do whatever processing would be necessary.

The ERROR-FIELD > 0 for an "FREAD" indicates that it is the end of the file and is not always an error. On an "FWRITE" this same error means the file is full.

HP's "F" words are not really bad words.

They give you better control over sequential files and help you design better systems. You will have the opportunity to look very clever without spending any extra effort or time.

I hope this has you thinking of the possibilities that exist when you start using file intrinsics.

THANK YOU

Paper #3911

"Data - Now that you've got it... What are you going to do with it?"

John Bomba

Innovative Information Systems, Inc.
123 Commons Court
Chadds Ford, PA 19317
(800) 766-7880

Today's "typical" data processing organization has become reasonably proficient in it's development of online transaction based systems. These systems support basic business functionality by collecting data, automating redundant tasks, and hopefully expediting daily operations. As well as these systems may address individual needs, they fall short in their ability to meet the growing demands of the business community.

Capturing the data is only part of the battle. Providing access to the data, in a timely manner, in an understandable and useable format is the remainder of the puzzle. Solving these issues will result in turning the "data" into information, and can be accomplished through the development of an Information Plan.

Information Planning is the first phases in the systems life cycle. By definition, information planning is the process by which an organization determines what data it will need to collect, how to collect it, who needs it, and how it should be delivered over the next several years to meet the organization's objectives. It reviews these issues in order to maximize the use of hardware, software, communication, and personnel resources. The plan provides a framework through which the remaining phases of the systems life cycle can be achieved. Yearly updates to the plan will ensure that system development is consistent with the changing business needs of the company.

Organizations have spent years and countless dollars "automating" business functions. They now possess an unquantifiable and valuable asset, "data". As valuable as this data is, it is tremendously under and ineffectively utilized in most environments. These islands of data need to be integrated, turning data into the information needed to direct your business into the future. An Information Plan will assist you in this effort by providing a road map that identifies where you are, where you need to go, and a way to get there.

Pitfalls in Moving to a 4GL

Author - Billy Hollis

Abstract

Several generations of computer languages have set a trend of increasing programmer productivity. The fourth generation of languages has continued this trend, but has led to new problems. 4GLs in general have not fulfilled their promise, and this has slowed their acceptance.

We will discuss the 4GL concept, including how they increase programmer productivity. Then we will look at common problems in moving to a 4GL and how to avoid them. Performance issues in going to a 4GL will be addressed. Methods for facilitating migration, such as conversion of COBOL to 4GL software, will be covered. Finally we will discuss the overall business case (pros and cons) for moving to a 4GL environment.

This presentation is relevant to those interested in the topic of fourth generation languages on the HP 3000 and HP 9000 platforms.

The Fourth Generation of Computer Languages

The first generation

The first computers were programmed in binary. Each program was actually produced as a series of bits. This is usually called the first generation of programming.

The second generation

Later, mnemonic assemblers were introduced. This allowed programmers to specify an instruction and have the "assembler" translate it to binary and perform things like address assignments. This was the second generation of programming.

The third generation

Then higher level languages were introduced. These "third generation languages" included COBOL, FORTRAN, and BASIC. Later, languages such as C and Pascal were added, and even though they had some improvements, they were not different in concept.

The driving force behind these generations was improved **programmer productivity.** As computer prices came down, it became more and more feasible to shift work to the computer rather than the programmer.

The fourth generation

In the late seventies, the concept of a "**fourth generation language**" (abbreviated as **4GL**) was introduced. The idea was to achieve another jump in programmer productivity. System Z from Zortec was one of the results of this new generation of languages, which have several general methods to achieve the increase in programmer productivity.

What are the requirements of a 4GL?

James Martin lists the following requirements of a Fourth Generation Language:

- It is user-friendly
- A non-professional programmer can obtain results with it
- It employs a data base management system directly
- Programs can be created with an order or magnitude fewer instructions than COBOL
- Nonprocedural code is used where possible
- It makes intelligent default assumptions about what the user wants, where possible
- It enforces or encourages structured code
- It is easy to understand and maintain structured code
- Non-DP users can learn a subset of the language in a two-day training course
- It is designed for easy debugging
- Results can be obtained in an order of magnitude less time than with COBOL or PL/1

How 4GLs increase productivity

Non-procedural logic - the programmer can specify what is to be done rather than exactly how to do it.

Concise code - One generally accepted feature of fourth generation languages is the ability to perform a task with one tenth the number of lines of code as a 3GL such as COBOL. This means fewer lines to write, fewer lines to debug, and fewer lines to maintain.

"Intelligent Defaults" - The language should try to "guess" what the programmer wants to do as often as possible. The programmer should never have to specifically request the most common option or parameter - it should be assumed. The programmer is only concerned with exceptions to the rule.

Application Generators - For common programs such as reports and file maintenance, application generators should be used to produce the finished product, relieving the programmer of the tedious work of producing the same type of program over and over with minor differences.

Other desirable attributes of 4GLs

Other features of 4GLs that contribute to their advantages over 3GLs include:

Prototyping Tools or Abilities - Either the language itself is constructed to allow easy prototyping, or there are special tools built in to help with the prototyping process (or both). Note that this ability is completely different from CASE methods, although there is some overlap.

Portability - 4GLs often are seen as a way to become hardware independent. Having learned the lessons of COBOL and FORTRAN, 4GLs usually try not to vary in syntax from computer to computer. This gives some degree of portability of the software.

Documentation Tools - Sometimes included in a 4GL is some capability to assist with documentation of the finished product. Again, this is not the same as CASE, but there is overlap.

Summary of Major 4GL Characteristics and Benefits

- Primary purpose - increased programmer productivity
- Brief code to perform complex actions
- Let the software take care of nit-picking details
- Ease of program maintenance because code is brief and easy to understand
- Easy prototyping
- Desirable capabilities include portability, documentation tools, and end-user tools which are accessible to non-programmers

The end result of all these features is intended to be faster application development.

Components of a typical 4GL

Most 4GLs share the following components:

- Data dictionary
- Interactive editor/compiler
- Integrated debugger
- Integrated database engine

Let's look at these elements one at a time.

Data Dictionary

One of the major innovations of 4GLs is the required use of a data dictionary. While the data dictionary concept is not new (many systems designers grafted data dictionaries onto 3GL languages as a useful supplement), it was only with 4GLs that the data dictionary became an integral part of the system.

A data dictionary is a central repository of information about data structures. In the best case, just about everything a program (or programmer) needs to know about data can be stored there.

Putting all the definitions for data files in one place has several advantages. Programs do not need to contain data definitions, so they have fewer lines. Systems are easier to maintain because any changes to data base structure only need to be made in one place.

The data dictionary also allows end-user oriented tools to work. Users don't need to know anything about data structures to use query languages, for example. They just need to know the data names they want information about. With the data name, the 4GL will look up the rest of the database information in the data dictionary.

The data dictionary contains definitions for data records. Each record definition includes field definitions for the individual fields in the record. The field definitions consist of things like data name, data type, length, default screen labels, and so forth.

Interactive Editor/Compiler

4GL compilers (or interpreters) are usually interactive, which means they can check source statements as you enter them. If you make a mistake while entering a program line. you find out about it immediately. 4GL compilers/interpreters usually have a great deal of intelligence built in. For example, programmers can usually omit explicit OPEN and CLOSE statements and the compiler will insert them as necessary.

Integrated debugger

Debuggers have been around a long time as add-ons to languages. Some 4GLs have a debugger built in. This means it can be a lot easier and more convenient to use. Properly used, a debugger can save tremendous amounts of time in finding program logic errors.

Integrated database engine

No major 3GL was designed with a fully functional built-in database. External databases were grafted on with varying degrees of success. This is one of the factors that made variants of 3GLs so different and incompatible.

4GLs have an integrated database. This means the databasing methods are transparent to the programmer. A program using an ISAM database on one platform can still look exactly the same on another platform that uses hashed indexes. This makes the language easier to learn and use, and more portable.

If 4GLs are so great, why doesn't everybody have one?

The description of a 4GL above covers all the positive aspects. But 4GLs are a long way from being the dominant software development tools. Why?

Of course, one factor is that most existing 3GLs are locked into one brand of hardware. The programmers are secure in their knowledge of their development language and existing applications. In a word, there is inertia - a definite resistance to change.

But there's a lot more to it than that. The fact is that there can be many hidden costs in moving to the typical fourth generation language. We will examine the reasons for this in detail below.

Things to look out for when choosing a 4GL

Here are the major items that can cause problems when trying to shift to a 4GL:

- Radically different proprietary syntax - no compatibility with existing 3GL source code
- Proprietary data formats
- Resource hogging (10x is not uncommon)
- No coexistence with current environment - need complete rewrite before starting to use new system.
- Not flexible enough to handle whole range of development needs
- Sometimes ties the user to one hardware platform

Let's look at these one at a time.

Proprietary syntax

Many 4GLs were designed according to theoretical models developed in an academic environment. The resulting syntax often ignored the lessons bitterly learned in real-world computing environments. Though 3GLs are not perfect, there are many aspects of them that work well, and existing programmers have a huge knowledge base gathered while using 3GLs. Inventing a totally new, proprietary syntax meant creating new problems of steep learning curves, and no preservation of the existing knowledge base of programmers.

Proprietary data formats

As with syntax, 4GLs were often designed to use theoretical models of database structures. While these models may have some advantages, there were two problems in using them. Old data structures could not be read or written by the 4GL (meaning massive file conversion), and these models were usually not designed with performance in mind.

Resource hogging

As mentioned above, new database models that did not take performance into account caused some 4GLs to use resources over 3GLs. The amount of extra resources was, in some cases, so large that existing hardware environments had to be massively upgraded to perform the same tasks in a 4GL environment.

Early 4GLs were also typically interpreted rather than compiled (and some still are). This resulted in another performance degradation.

These major factors plus some other minor ones, often caused a typical application to consume **five times** as much memory and disk space as an equivalent application in COBOL or FORTRAN.

No coexistence with current environments

The proprietary features mentioned above meant that a typical 4GL could not be put in place next to a COBOL environment, for example, and co-exist with it. Since different data structures are

required, the transition to a 4GL had to be abrupt and complete. There was much work to be done before the 4GL could be used at all, and then there was no going back. This dramatically increased the risk in trying a 4GL for a development environment.

Not flexible enough to handle the whole range of development needs

4GLs must contain high-level non-procedural syntax to be effective. But that can taken to an extreme. If the 4GL contains only high-level syntax, then low-level bit-oriented operations are impossible. This makes it necessary to have another language as a supplement to the 4GL.

In summary....

So the overall concern is the large hidden expenses in transition. These can make cost justification difficult. The move to a 4GL can mean a significant period of chaos for the MIS department, which means a great expense in man-hours throughout the company.

Let's look at the worst case situation. To use the new 4GL, the data needs to be converted to a new proprietary format. All the existing programs have to be re-written before the 3GL can be discarded. The lack of co-existence between the current 3GL environment and the new 4GL means the company will have to make a significant investment to see if the 4GL is even going to work out.

And the syntax most 4GLs employ can make a significant impact on the usefulness of the new language. Because the syntax is radically different than the 3GL, programmers will have a steep learning curve. This means a fairly long period of time before they can be productive using the new development tool. And with a more limited syntax, the new language may not allow the same degree of complexity in programs as the former 3GL. This means such languages as C or COBOL will be necessary to fill the voids where the new 4GL fails to measure up.

Some notes on portability

4GLs vary in their portability. Some run on a wide range of systems with virtually no change. Others are limited to a few systems, or are not really exactly the same on the different platforms they support. And if you thought that because a 4GL meets certain ANSI standards, you would have a very portable language, you may be disappointed to find that there may still be a lot of work in moving from one platform to another because typically ANSI standards are not complete enough to ensure portability.

The Good News

Many of the problems mentioned above have been addressed in the latest 4GLs which are now available. Some of these have an architecture that overcomes the traditional 4GL problems. A 4GL that eliminates the difficulties discussed needs the following features:

- Works with existing databases
- Has easy-to-learn syntax because it supports familiar 3GL procedural logic
- Less resource utilization - ideally less than what 3GLs use

- Conversion utilities that can convert 3GL code to 4GL code automatically (thereby preserving the investment in existing code)
- High portability - all the way to the compiled code
- More powerful language syntax so that 4GL can be the only language required, from low bit-level operations through 3GL procedural logic up to true 4GL concise code.

There are some other desirable capabilities of a modern 4GL that builds on the evolving technology of advanced development environments:

- Supports 3GL syntax in line
- Lots of utilities to increase productivity, preferably written in the 4GL itself to allow easy customization
- Application generators for commonly used functions - that produce finished, customizable source programs
- Ideally, should give relational access to data without overhead of a typical relational database engine

4GLs are available with these features. But some 4GLs, because of their architecture, cannot implement some or all of these abilities. So it's important to check on any of these which are important to you before choosing a 4GL.

Another important aspect to check in detail is portability - not only to the platforms you use now, but the ones you expect to use in the next few years. Also remember that many 4GLs attain portability by simply being interpretive. This means portability has a high price - extremely high resource requirements. A truly portable 4GL can make virtually all utilities and compiled code exactly the same on all platforms. On the following page is a diagram showing the modules of our 4GL, System Z. To show its extreme portability, you should note that **all modules** except the Interface Module and the Data Base Files are **exactly the same on each platform** which System Z supports. This level of portability is the ideal situation.

System Z Component Overview

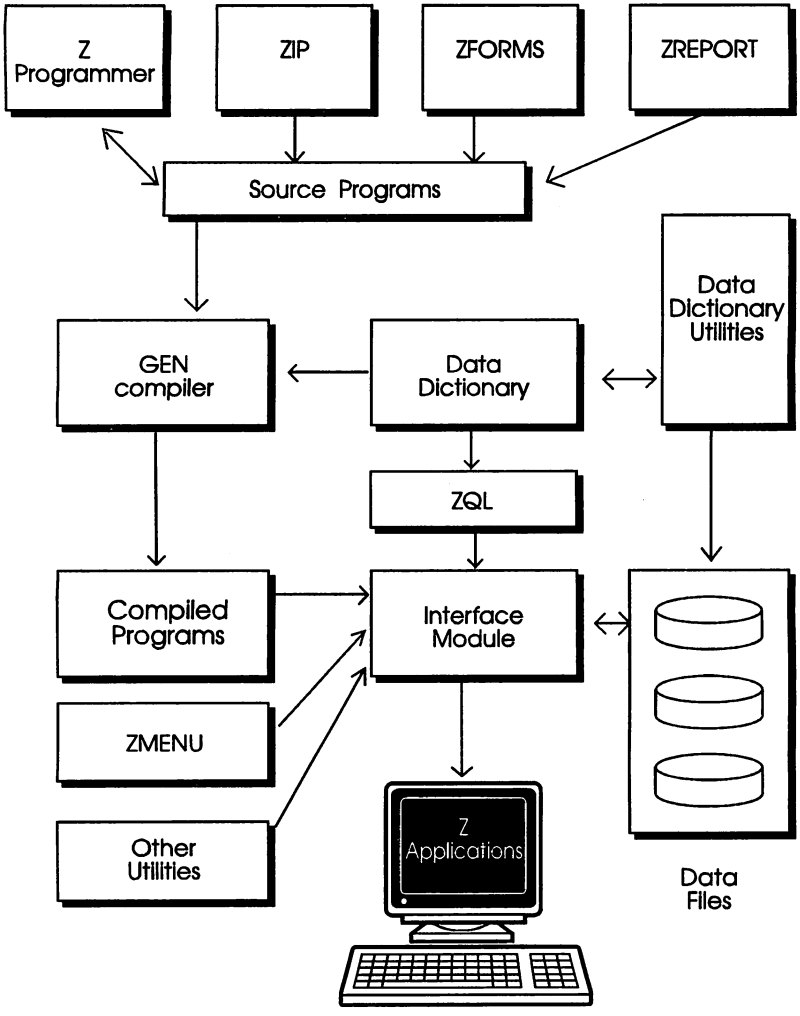


Figure 1

Converting COBOL to a 4GL Environment

One of the desirable characteristics of a 4GL that was mentioned above is the capability to convert existing COBOL code to 4GL syntax, thereby preserving existing investment in programs. This is one of the most misunderstood capabilities of 4GLs. Some prospective 4GL users view COBOL conversion as a panacea - others don't understand how it can work at all.

Those who are suspicious of the possibility of converting COBOL have usually used COBOL translators before. These programs, which are intended to convert one "flavor" of COBOL into another, have been around for quite a while. Most of them have limited usefulness. Even a simple COBOL program has many aspects that are difficult to pin down with a translator.

But, as strange as it sounds, it's actually easier to convert from COBOL to a 4GL than it is to translate flavors of COBOL! That's because 4GL programs tend to be simpler and more non-procedural. Processes don't have to be specified in detail.

At the opposite extreme, some users believe they can just dump COBOL in one side of a conversion utility and get complete, efficient 4GL programs out the other. This is not realistic. To take one example, no conversion utility is going to understand how a call to a user-defined library works. The best COBOL conversions have a table look-up facility to allow substitution of 4GL syntax for user-defined calls, but setting up such tables takes work. And even if the conversion facility produces a functioning end result, the program will not be as efficient as a program coded from scratch in the 4GL.

But if these points are taken into account, the ability to convert COBOL to 4GL syntax can save a large amount of time in getting to a complete 4GL environment.

Bottom Line -The Business Case for Using a 4GL

Why should you consider a 4GL? Here are some of the possible reasons:

- A 4GL can allow you to get MIS changes and additions on line faster, making your company more competitive
- A 4GL can allow you to serve your customers better
- A 4GL can protect your investment in existing databases
- A 4GL can protect your investment in software by allowing portability to newer, better machines
- By accomplishing some or all of the above, a 4GL can allow your company to have higher earnings per share

The goal is to find a sane migration from the third computing generation to the fourth. There are certainly pitfalls to avoid. We have presented some of the potential problems to watch out for. Taking these into account, you can choose a 4GL which will enhance your ability to deliver software systems, but won't destroy your peace of mind in the process.

Paper Number: 3913

Presentation Title: TurboIMAGE and Allbase/SQL Converting and Integrating these Data Sources Using 4GLs

Author: Marlene Nesson

Address: 1250 Broadway, New York, New York 10001

Telephone: (212) 736-4433

Situation Analysis

The HP environment was predictable before the introduction of RISC (Reduced Instruction Set Computing) architecture. All midrange computers ran the MPE (Multi Programming Executive) operating system, and the database used was almost always TurboIMAGE. A small minority used KSAM (Keyed Sequential Access Method), and most used flat MPE files for secondary storage. HP View Screen was the interface for Cobol applications, and networking services consisted of transferring data to and from a mainframe via RJE.

For those in search of a 4GL (Fourth Generation Language), the choices were few but familiar. Whether the choice for application development was Powerhouse, Speedware, or HP's Rapid, the database was always TurboIMAGE, and for the most part, HP 3000 users were accustomed to working in a closed proprietary environment.

The HP Environment Today

"Open Systems" is the operative term on the tongues of HP users today, a phrase which carries with it a more complex application model, and a variety of tools, a database management systems and network interfaces from which to choose.

The introduction of 32 bit RISC architecture ushered in new and powerful databases for the HP environment. Many of these products are designed to replace TurboIMAGE rather than co-exist with it. Some products offer limited access to TurboIMAGE data, others offer conversion utilities, and others have no provisions at all for working with TurboIMAGE data.

While Open Systems provides flexibility and increased options, it also jars HP users from the comfortable pre-Spectrum environment. The choices are more complex, and with many third party vendors designing their products without a commitment to protect existing investments in TurboIMAGE data, the risks are greater than ever before. The difficult transition between the advantages of an Open Systems environment and the easy, safe decision-making process of the proprietary closed one has arrived.

The Thrust toward Open Systems

The X/Open definition of open systems is "Software environments consisting of products and technologies which are designed and

implemented in accordance with standards established and defacto; that are vendor independent and that are commonly available." Using this definition as a baseline, we can extract the real world benefits of an open systems environment for each type of user in an organization:

End Users - Open Systems will give them transparent access to data, using one common user interface. This results in reduced training time, and higher productivity across a broad range of applications. Gain

MIS Administrators - The hardware and software independence of Open Systems allows them to provide better integration of multi-vendor environments, scalable applications across platforms, and mor product choices to suit specific requirements.

Application Developers - A common source for application development reduces training and porting costs, as well as increasing the productivity of this group. A common source also allows for tighter development schedules, and better workflow through an area that traditionally gets bottlenecked with application requests.

Solution Vendors - Open Systems gives providers the ability to offer new technologies to users of many different systems faster than before. Products will be available to users of many platforms at the same time.

Open Systems and Operating Systems

Open systems has undeniably created an increase in choices not only in databases, tools and interfaces, but choices in operating systems as well. HP users can upgrade to MPE XL, migrate to HP-UX, or commit to relational databases such as Allbase/SQL.

In this emerging, multi-vendor, multi-database environment, users need tools that provide freedom of choice and that efficiently integrate old and existing technology. These tools must also provide a growth path for integrating future technologies as well.

4GLS: A Perspective

A 4GL must provide the ability to work with multiple data sources across multiple platforms, provide application interoperability, and fully support industry standards.

The major 4GL components that form the foundation from which these requirements are satisfied are outlined below. Although it is not an exhaustive list, it provides the functional requirements that should accommodate needs of today, and offer an architecture designed to handle software and hardware advances in the future:

4GL Components

1. Single language for application development.
   A single language for application development should be standard regardless of the data structure and regardless of the complexity of the application requirements. The lan-

guage should be easy to learn and use but also provide computational capabilities found in traditional programming languages. The language should support sub-routines and procedural controls so developers can specify process flows and interactive dialogues.

2. Flexible database choice.
The ability to implement the database of choice is critical for a 4GL now and in the future. The 4GL should provide developers the ability to snap-on the desired database for reporting and maintenance of database.

3. Comprehensive Decision Support facilities
A full complement of Decision Support tools should be included in the 4GL. Decision support covers the process by which data is retrieved, analyzed, formatted, displayed and transformed into information. These functions should be specified with simple language, easy to learn, yet powerful enough to handle any reporting requirement. It should be a powerful offering including a Report Writer, statistic and graphic packages. These tools will enable an integrated environment for turning data into information.

4. Automatic code generators
The ability to automatically generate the 4GL code for application development is critical to enable a productive environment. The code generators for reports and maintenance procedures should be user friendly, easy to use and generate error-free code.

5. Universal access to diverse data structures
The ability to access and integrate different structures together, across operating systems, enables productivity especially in the light of the mixed technology both hardware and software environments today and in the future.

6. Combine diverse data structures
The ability to dynamically join different data sources together, such as relational, hierarchical, network, etc. The facility should be simple to use so even casual users can produce reports, for example, which consolidate data from diverse sources.

7. Screen and window painters
Productive facilities such as screen and window painters enables developers to add window-based front ends to applications without the need to know the 4GL syntax. These facilities should provide the flexible for the developer to create pop-up and pull-down windows that will be attractive and easy to use for end-users accessing the systems. The

generated code should be able to be enhanced easily so as to provide the flexibility for application changes and enhancements.

8. Dictionary is flexible and functional
   The dictionary of a 4GL should be the central warehouse of information about field names, lengths, data formats, security, data validation criteria, etc. The dictionary can even play a more powerful role if, for instance, it supports predefined relational joins within the dictionary itself. The dictionary for data sources (such as TurboIMAGE, Allbase/SQL, etc.) should be automatically generated by accessing the native database dictionary.

9. Obey existing security
   The 4GL should honor existing security of the database and provide the ability for the application developer to further refine existing security.

10. Manage simultaneous updates
    Simultaneous updates by multiple users should not lock out users but employ a mechanism to manage simultaneous updating of a database while ensuring the integrity of the data. Commit and rollback facilities to permit (commit) or deny (rollback) updates should be available.

11. Available across all major platforms.
    With the diverse environments today and in the future, it is critical the product run under all major operating system. If this is not the case, it could present an unpalatable situation if a different hardware platform is incorporated into your enterprise. This is costly both in terms of the purchase of the product as well as the time required to develop expertise in the new product.

12. Code is portable across all major platforms
    Portability of code in a mixed environment gives the application developer the ability to choose the right processor to develop an application. For instance, the application developer should have the ability to program the applications on a PC, thereby freeing up resources on the HP, and porting those applications directly to the HP.

13. Client/server architecture
    The 4GL should support and support client/server architecture enabling processing to be distributed.

14. Connecting diverse hardware platforms and commitment to standards

Connection to diverse hardware platforms is critical. It is equally important, however, for the technology to support standard protocols such as SNA LU 2.0, LU 6.2, and TCP/IP.

15. Futures
The commitment of the company to the integration of new technology into its product line including: Expert System support, GUIs, Object Oriented paradigms, and CASE tools, will provide the tools to further enhance productivity.

If you are thinking about incorporating a 4GL in your environment, currently evaluating a 4GL, or evaluating the functionality of your current 4GL, the product you choose should provide comprehensive functionality and should embrace the strategy that will make the product viable now and in the future. Challenges, such as the integration and/or conversion of TurboIMAGE with Allbase/SQL, will then be a painless task and enable you to maintain investment in current and future technology.

4GLs in the HP marketplace provide HP data support but, for instance, do not run on all major platforms, such as the IBM mainframe environment. So although the 4GL may be attractive functionally, it may not be suitable solution in a diverse hardware environment that includes a mainframe.

An example of a 4GL that would meet these requirements, is FOCUS from Information Builders. FOCUS provides the breadth of functionality and strategic direction by providing a robust 4GL, runs on all major platforms including: HP MPE XL, HP-UX, IBM mainframe, and PC both DOS and OS/2. Code is portable across operating

systems, and provides an interoperatable environment. Access to TurboIMAGE, Allbase/SQL, KSAM, MPE are all supported.

Advantages of Relational Technology

If you have not yet made the decision to convert your TurboIMAGE to Allbase/SQL, it may be worthwhile, to briefly look at the advantages of a relational database. Certainly, there are both pros and cons. HP users know that TurboIMAGE database has been an excellent performer on the MPE XL operating system but Allbase/SQL is catching up. In the old debit-credit benchmark, TurboIMAGE scored a 19 TPS; Allbase/SQL (SQL 2.0) scored an impressive 14.4 TPS. HP has made significant performance improvements since this time, and plans to further improve the performance of Allbase/SQL in 3.0 of the HP MPE XL release.

Let's look at some of the benefits using a relational versus a network, in this case, TurboIMAGE structure.

Application Development Needs and Flexibility

Relational databases lends itself to easier translation of data needs into an application than that of a TurboIMAGE database. Once the relational databases are created, there is greater flexibility to modify the database design to meet changing data needs. Applications depend less on the underlying database design in case of a relational database management system (RDBMS) than they would in case of a DBMS like TurboIMAGE.

Multiple Keys

Relational tables provide the ability to define multiple keys whereas in a TurboIMAGE master dataset only one key is allowed. This could possibly mean less programming effort and less complexity in the application because the need for navigation of data is reduced.

Ease of Sorting

Sorted output is required in almost every single application. Output is sorted from a table by keys with only a small effort required in terms of runtime system and programming effort versus the effort required with TurboIMAGE structures.

Integrated Package

More and more third party 4GL vendors offer an integrated package including: interface to relational database systems (Allbase/SQL, Oracle, Sybase, Ingres, Informix), as well as comprehensive application development and end-user tools.

Future Technology

Relational technology is the choice today and probably of tomorrow.

Innovation and Growth

Finally, growth and innovation of SQL products are constant because of the intense competition. This trend should continue and ensure increased functionality and flexibility for the future. Figure 1 summarizes the advantages of relational technology.

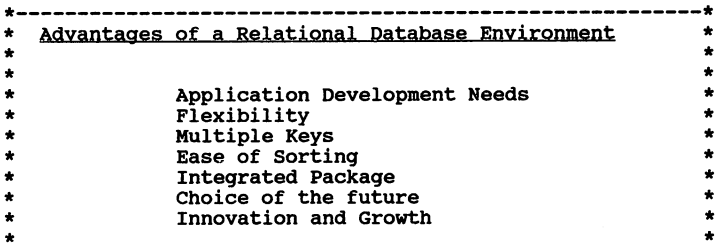


Figure 1

Although there are advantages to moving to a relational database, the best decision is to analyze the business need you are looking to solve and implement the most appropriate database structure to meet your needs.

Terminology and Basic Concepts

Before we actually look at converting and integrating TurboIMAGE and Allbase/SQL, it would be helpful to look at the terminology and basic conceptual differences between TurboIMAGE and Allbase/SQL. With this knowledge, it will then become more apparent how powerful a 4GL can be when we look at converting and integrating these two dissimilar structures.

In order to understand Allbase/SQL it's important to learn some new jargon. Figure 2 describes a description of new terms introduced with relational technology, from a TurboIMAGE perspective.

Given the various terms defined, perhaps the most confusing is also the simplest database. We "open" a TurboIMAGE database. For every application, whether the application program uses one database or several databases, each is opened individually. The relationship between databases is non-structural; any structure is imposed by your program. In Allbase/SQL, however, the operative term is "connect" to a database environment.

In addition to terminology differences, there are some basic differences with regard to data names, data types, subitems, security, and table definitions.

Data Names

TurboIMAGE allows names to be up to 16 characters long whereas, Allbase/SQL allows basic names to be up to 20 characters long.

Data Types

The mapping between data types is shown in Figure 3. All TurboIMAGE data types except Z (External, Zoned, Decimal) have an equivalent in Allbase/SQL.

Subitems

Subitems do not exist in Allbase/SQL. You would need to translate any TurboIMAGE field with a subitem count greater than one to multiple columns in Allbase/SQL.

For example, given the following TurboIMAGE field:

COMMISSION, 4I2

would need to be translated to four column in an Allbase/SQL table:

```
COMMISSION1, INTEGER
COMMISSION2, INTEGER
COMMISSION3, INTEGER
COMMISSION4, INTEGER
```

TurboIMAGE	Allbase/SQL	Definition
Entry	Row	'Record' in TurboIMAGE or a 'tuple' in SQL.
Field	Column	Data unit within a row or entry
Set	Table	Logical entity containing data entries or rows
Database	Owner	Logical entity containing database sets or tables
Query	ISQL	The basic HP supplied programs for ad hoc queries and updates of TurboIMAGE and Allbase/SQL databases.
Root File	DBECON file	File contains control information

Figure 2

TurboIMAGE	Allbase/SQL	
P	DECIMAL	
Z	DECIMAL	
R	FLOAT	
I,J,K	SMALLINT	If length 1 (16 bits)
	INTEGER	If length 2 (32 bits)
	CHAR	If length greater than 2
X,U	CHAR	

Figure 3

Security

Allbase/SQL and TurboIMAGE security are fundamentally different. Security in TurboIMAGE is established by the password determining items a user can read and/or write. With TurboIMAGE, passwords are a structural element of a TurboIMAGE database and restructuring is required to accommodate any access changes. Any user who has knowledge of the password can access the database.

Read/Write access to a data field is determined in Allbase/SQL by assigning user logon IDs to Allbase/SQL groups and assigning various access capabilities to the groups. This information can be changed anytime by the DBA. Capabilities of the USER are thus completely determined by the user's logon ID and can be revoked or expanded at any time.

Table Definitions

With TurboIMAGE, the first step is to define all of the items available in your database. Next, within each set, specify the items

to be fields in that set. If you wish to add fields or items later, you would need to restructure.

In Allbase/SQL, define the columns for a table within the definition of the table itself. If you wish to use a column of the same name and data type in another table, you must redefine it in that table. Additional new columns can be added to the end of a table at any time with a single Allbase/SQL command. If you wish to add columns in the middle of the table you can do so but it requires three or four Allbase/SQL commands.

Indexing

TurboIMAGE and Allbase/SQL both feature indexes. Changing a TurboIMAGE index definition requires database restructuring.

Allbase/SQL, on the other hand has an interesting concept of immediate creation or deletion of indexes. Certainly, there are both positive and negative ramifications. A positive offshoot of immediate creation of an index is the construction of a quarterly report. Once the report is completed, the index can be dropped so online applications will not be slowed down by the maintenance of the index. On the negative side, you can incur a huge overhead from regularly generating an index.

Converting a TurboIMAGE Database and Application to Allbase/SQL

Assuming, you have made the decision to convert your TurboIMAGE database to Allbase/SQL what are key elements to consider and work

to be done? We will approach this from 2 angles: 1) with a 3GL and 2) with a 4GL. Although both approaches are feasible, the 4GL solution will require less resources and provide the flexibility for new application needs for the future.

First, with a 3GL the steps are:

1) Create a logical database to meet all your known data needs. You can either try to best mirror the TurboIMAGE data structure or you can start from the beginning without regard to the current data structure and create the logical design.

2) Convert the logical database design into an Allbase/SQL database design. Of course, you must invest the time to read the Allbase/SQL reference manuals.

3) Create the Allbase/SQL database (structurally).

4) Convert Data from TurboIMAGE database to Allbase/SQL. One alternative is to write a COBOL program to select all or selected portions of data from the TurboIMAGE database. Write the selected data to a flat file. Then create another program to load the data into the Allbase/SQL database.

   This step would also include determining the conversion from the existent TurboIMAGE datatypes to the corresponding Allbase/SQL datatypes (refer to Figure 3).

   This process would be duplicated for every TurboIMAGE database.

5) Write or rewrite your application code.
   The number of reports, the requirements of maintenance, and complexity of application, will indicate the number of programs that will need to be rewritten or created.

From the above discussion, it is evident that the time required can be substantial in the analysis, creation, extraction, loading of the data, and finally the re-engineering of the existing programs for

reporting and maintenance. The actual time required would be dependent upon a number of variables including: the number of TurboIMAGE databases, the number and reports and complexity of reports (sorting, totals, subtotals, aggregation (sum, count, etc.)) and the complexity of the maintenance procedures required.

Let's now look at the process that would be required with a 4GL. Certainly, the functionality and capabilities differ among the existent 4GLs available in the marketplace. But let's assume you choose a 4GL that meets most or all of the suggested requirements of a 4GL (outlined above). The process would be as follows:

1. Create a Logical Structure
2. Determine Data Requirements

   Determine the fields from the TurboIMAGE database that you wish to extract (can be all or selected fields).

3. Convert data from TurboIMAGE to Allbase/SQL

   *a. Use an End-User tool to extract data

   4GL code should be generated automatically that will extract the TurboIMAGE data based on the user requirement.

   *b. Use an End-User tool to automatically create Allbase/SQL Tables

   Tables should automatically be created based on user requirement (i.e., fields chosen, sort fields identified, etc.). The database administrator should have the ability to change the structure created if desired.

   *c. Use an End-User tool to automatically load the data

   The 4GL request to load the data should be automatically created and should be flexible to run in batch mode if appropriate or required.

* The ultimate situation would be a 1 step end-user conversion tool that combines all 3 steps (a, b, c).

4. Create maintenance and reporting programs

   The 4GL requests to create screen driven update procedures and reports should be automatically created.

Steps 1 and 2 above would be the only required "think" time. Step 3 and 4 should be performed quickly. Given the 4GL has a powerful report writer that provides for automatic totals, subtotals, aggregation, etc., the time required to convert the existent application from TurboIMAGE to Allbase/SQL would be significantly reduced using a 4GL.

The conversion of TurboIMAGE databases and applications to Allbase/SQL is, as we can see from above, is feasible using a 3GL if there is the time and necessary resources available to perform the tasks. The number of TurboIMAGE databases, and the complexity of the application, will determine the feasibility of converting using 3GL technology.

Using a 4GL however, the task is quicker and does not present the challenge of the requirement for additional resources. A 4GL that is rich in functionality will promote the quick and efficient creation of SQL tables, and reports and maintenance procedures (very time consuming in a traditional 3GL language).

An extremely important benefit of a 4GL that must not be overlooked is the increased productivity for new applications as well as enhancements to existing applications. If the 4GL has powerful tools, applications can be enhanced quickly and new applications can be created quickly and efficiently.

Integration of TurboIMAGE and Allbase/SQL

Integration of TurboIMAGE and Allbase/SQL databases presents a whole new challenge. The question is: How can you integrate these structures together using a 3GL and a 4GL. Firstly let's look at a solution using a 3GL then a 4GL.

Because the two databases are inherently different, TurboIMAGE is a network structure, Allbase/SQL is a relational structure, there is no apparent simple method to integrate these data structures together using a 3GL.

One possible approach for reporting is to extract the requested data from TurboIMAGE, write the data to a file and then save the file. Similarly, perform the same exercise using the Allbase/SQL database. Write a program to merge the data from the two files into one file. Finally write the required reports. Based on application requirements, the creation of reports can be simple or very complex with requirements such as numerous sorts, subtotals, totals, aggregation, headings, footings, etc.

Maintenance of both data structures would require the creation of programs to update the Allbase/SQL databases. Essentially, this mears you must commence from the beginning, creating all the required maintenance programs including screens, etc. to update the new Allbase/SQL structures.

The number of Allbase/SQL tables will dictate the amount of time and resources required to accomplish these tasks using a 3GL. Realistically how much resource would be required to integrate and create the required reports? In addition to the number of Allbase/SQL tables, other variables include: the selection criteria, report requirements (subtotals, totals, aggregation functions, headings, footings, etc.) and the number of reports.

Let's now look at what would be required to perform these two activities using a 4GL.

Report generation would entail logically joining the two structures together (assuming there are fields in common across the data structures). Once this is completed, the desired reports would be created using an end-user report generator. Given a powerful 4GL report writer with effective end-user code generators, complex report generation, as described above, will be easy and quick.

For maintenance, combine the TurboIMAGE and Allbase/SQL databases together and then create the maintenance procedures to update the combined structure. The 4GL should supply end-user tools to automat-

ically generate the procedure to update the combined structure. The time and resource to accomplish these tasks should be minimal if the 4GL is extremely flexible, functional and designed architecturally to integrate diverse structures for reporting and maintenance.

Summary

Let's conclude by highlighting the issues discussed in this session. With the introduction of RISC technology, HP users find themselves with many more choices and decisions. There are more powerful 4GLs, operating systems to choose, and relational technology to consider. A diverse hardware and software environment suggests the tools of choice be flexible, comprehensive, and provide multivendor comprehensive, and provide multivendor connectivity and interoperability now and in the future. A well architected 4GL with a full complement of features today with a solid strategy for the future is worth pursuing.

Relational technology has become more or less a standard. Although there are several advantages to this relatively new technology, the decision to invest in relational technology should not dictate the requirement to convert all TurboIMAGE applications to Allbase/SQL. The business need should be examined and the structure that meets the need should be employed. The best of both worlds is a 4GL that will enable integration of both databases into existing and new applications.

Using a 3GL, the conversion of TurboIMAGE to Allbase/SQL can be a time consuming and resource intensive process. There are conversion issues that would be required to be addressed including: data types, subitems, security, and table definitions, to name a few. There are changes to existing programs and creation of new ones. The requirements and volatile state of the applications, would dictate the time and effort required now and in the future using 3GL technology.

Using a 4GL, the conversion can be simple. If the 4GL enables the joining of diverse structures together, has powerful end-user tools, and addresses a majority of the conversion issues automatically, the conversion from TurboIMAGE to Allbase/SQL should be quick and easy with minimal allocation of time and resource.

Similarly, the integration of TurboIMAGE and Allbase/SQL for application development for reporting and maintenance can be extremely resource intensive using a 3GL and with the choice of a flexible, powerful 4GL, can be expedient.

The 4GL you choose can make the integration and conversion of todays data structures and tomorrow a simple task and provide productivity tools to enable efficient and effective application development in the future.

Paper # 3917

TurboImage/XL Performance

By Denys Beauchemin
Bradmark Technologies, Inc.
4265 San Felipe Suite 800
Houston, Texas 77027
(713)621-2808

TurboIMAGE/XL has been around for about 4 years now, and already it has changed significantly. The performance improvements have been dramatic and there are more features which are available now, or are planned for the future, which make this DBMS still the uncontested winner in overall performance and usage on the HP3000 with MPE/XL. You may have heard a lot of talk about how Image is dead and ALLBASE/SQL is the DBMS to which everyone and everything is going. Well, don't believe it, TurboIMAGE/XL is alive and well and is acquiring more features, at a faster rate, than ever before. And these features are not little things either, they are very wide ranging in some cases. Today, the subject I would like to cover, is something that is not talked about often enough; TurboIMAGE/XL performance. I have given talks on the subject and have actually gotten very good feedback from folks who listened and went ahead and did the things I discussed.

Let's face it, the present is TurboIMAGE/XL. This is also true for the foreseable future. The information herein is long overdue, and I am trying to catch up.

A) TurboIMAGE/XL from 0.0 to 2.05 -- HPIMAGE w/ TurboWINDOW and TurboIMAGE/V

When MPE/XL was first born, TurboIMAGE was available but only in Compatibility mode. In order to go Native Mode with your programs and your databases, you had to migrate your databases to HPIMAGE. This new DBMS was similar to TurboIMAGE but "similar to" is not "exactly the same as", and therefore there would have been quite a bit of code modification and even logic changes in the programs which were to be migrated to native mode to use HPIMAGE. In order to facilitate this migration, HP introduced TurboWINDOW which was to provide a TurboIMAGE-type window into an HPIMAGE DBEnvironment. This window had various degrees of transparency and the whole thing was not an overwhelming success.

It was decided in late 1985 to migrate the current TurboIMAGE into native mode, and HPIMAGE w/ TurboWINDOW was put on the back burner.

B) TurboIMAGE/XL -- Migration, what migration?

The issue of migration for the databases and the programs which access them, is really a non-issue. One only needs to restore the database from an MPE/V store or DBSTORE and presto, it is available right now, and in Native Mode access no less. Just make sure that you turn off ILR on MPE/V before you store the database as the ILR formats of V and XL are incompatible. The neat thing about TurboIMAGE/XL is that your programs can use it right then and there. If you

start recompiling your programs with a native mode code producing compiler, they will run without having to change anything in the code or the calling sequence. The programs accessing the databases can be NM or CM, mix and match and everything works. It can't be any cleaner. The databases can be ported back to an MPE/V machine by just using STORE with the TRANSPORT option. The only caveat here, is the size of the individual datasets. More on this later in this presentation.

What is this new file in my group?

Each time a database is first opened, a file is created with the name xxxxxGB, with a filecode of -406. This file represents what used to be the database global block (DBG) and the database buffers (DBB). It is now a permanent file which is accessed by everyone using the database. It is purged after the last user closes the database.

Bye Bye ILR, Hello XM.

The first thing which comes to mind when one gets on a MPE/XL system is the Transaction Manager (XM). This is the method developed by HP to guarantee that information is not lost due to a system interruption. The first by-product of the XM is the need to turn off ILR on your TURBO/XL databases. We will talk more about XM later on, in relation to volume sets. On MPE/XL 2.1, when you enable ILR, the 00 file is no longer created. Everything just gets posted to the XM logfiles immediatly, thus slowing down the system. Some folks are saying that ILR should be enabled on XL system, but that is just a little white lie, XM does an excellent job and thus, ILR should not be enabled.

Speed, speed, speed. $10000 words.

One of the things which was changed to increase performance without having to change anything in the logic of TurboIMAGE/XL was the increase of the global block from 32767 words to $10000 (65535) words. So crank up your buffspecs to take advantage of this expansion, but beware of opening them to much on 2.1. More on that later.

Oh no, not private volumes! No, it's Volume Management.

Probably the most performance-improving action one can take on a MPE/XL system, is also the one which is least talked about. Each and every MPE/XL system now in operation has at least one volume set; MPEXL_SYSTEM_VOLUME_SET. So even if you do not want to get into volume management, it is there intrinsically. The main point that I want to get across is that you need at least two volume sets, and probably more.

First, let's see what comprises a volume set. When a volume set is created, 400k sectors on the master of the volume set is immediatly reserved by MPE/XL. This chunk is needed for the directory of the volume set and for the XM logfiles. I understand that this figure is supposed to be reduced in an upcoming version of MPE/XL, but I don't think it's an issue at this time (by the way, this may explain why the 7912/14 are not supported on MPE/XL).

After the master volume has been declared, you can add member volumes to the new volume set. All the volumes, the master and the members, make up a single unit of recovery for the XM. It is not the intent of this presentation to describe the workings of the XM. But let's look at the impact XM has on performance and let's see how multiple volume sets will

affect said performance.

TurboIMAGE/XL transactions all go through the XM, provided that AUTODEFER is not enabled. Now let's take a large XL system with many databases all on MPE_XL_SYSTEM_VOLUME_SET. Since all transactions go through the same XM logfiles, the system appears to "slow down" for everybody at the same time, and quite frequently. This is because, PIN 9 the checkpoint processor is busy posting all the transactions to the volume set. You see up to this point all the DBPUTs, DBDELETEs and DBUPDATEs have not in actual fact been written to the databases. Rather, they have all been accumulated in the volume set's XM logfile. If the system were to fail while the transactions were in the XM logfile, nothing would be lost, because they would be recovered automatically by the XM when the volume set would be placed on-line, ON ANY XL MACHINE.

The only (small) danger would be if the master volume set were to break and XM would not be able to read the logfiles, ever. This would effectively disable the entire volume set, but not the system, or the other volume sets.

Back to the checkpoint processor. Now, while PIN 9 is busy updating the various volume members, the user activity on these volumes is somewhat curtailed. It used to be quite dramatic a few releases ago of MPE/XL, but even if it's better now, it is still there. Therefore, if all your transactions go through the same XM logfile, on LDEV 1, along with having some of the the directory on LDEV1 and most of transient memory (old virtual) also on LDEV 1, you can see that poor LDEV 1 is quite busy and on things that are all high priority.

What I propose is the following. Leave MPEXL_SYSTEM_VOLUME_SET alone. Create new volume sets and place your production accounts with their databases on these other volume sets. This will bring you many benefits, some of which are:

1- You will relieve MPEXL_SYSTEM_VOLUME_SET from a lot of work thus increasing the system perfor mance overall. You will be able to protect LDEV 1 from a lot of work better done on another drive. LDEV 1 is loaded enough as it is by things over which you have absolutely no control.

2- The XM processing will be spread around multiple volume sets over different periods of time, thus increasi ing overall system performance.

3- You will be able to control your disk utilization much more efficiently on seperate volume sets.

4- It permits one to perform backups for different areas at different times, and simplifies reloads immensely.

5- It gives the overall system an increased measure of resilience in case of disk failures and reduces the recovery time for a system restart.

6- It is required for MIRRORED DISK/XL.

A more in-depth discussion of XM would be a subject for another presentation Suffice it to say, that this is the most important action that can be taken to increase overall system performance.

C) The near-present, MPE/XL 2.1-2.2 and TurboIMAGE/XL

Mapped files, block sizes, pages and performance.

With MPE/XL 2.1-2.2, TurboIMAGE/XL uses a technique known as mapped files for access and updates. When pages are retrieved from secondary storage (read disk drive), they no longer have to pass through the main Image buffer. Instead pointers are moved around and the pages are left in transient memory. The pointers are in effect, headers for the buffer areas, and as such are much smaller, only a couple of dozen words. Therefore it is easy to reach whatever maximum buffspecs is specified in DBUTIL. However, the search for the appropriate buffer is, at the present time, a linear search, so one must look at balancing the search time, with the buffer specs. It is recommended by HP that the maximum buffspecs be set between 60 and 80. Note that this is no longer true on 3.0.

A note on DBUTIL buffspecs. When the database is first DBOPENed, the DBG is built automatically to the largest buffspecs specified in DBUTIL and never changes size, so beware of defaults! Since TurboIMAGE/XL now works on pages (4096 bytes), thus closer to the machine, I wondered how the blocking size would affect performance. My thoughts were that there would be a much smaller variation than before, as long as the block did not contain too much unused space. I am now conducting tests and surveys of various databases with various block sizes and the effects of block sizes on performance. This will surely be the subject of a future presentation.

Delta Logging.

At this time, the DBUPDATEs are logging only the items which have changed, in a range. For example, if there are 20 items in the set, and we modify item 3, 8 and 15, what will be logged will be items 3 thru 15. So if you do standard updates, try to keep the items which are changed, contiguous. The DBPUTs and DBDELETEs still do before and after images, at this time.

MPE/XL file size and you thought FSERROR 106 was gone.

When I first logged on to an MPE/XL system, almost 5 years ago, I noticed that many files now had a file limit which appeared on the LISTF like this: 4096000. For example, do this

```
:LISTF XL.PUB.SYS,2
ACCOUNT= SYS GROUP= PUB
FILENAME CODE ————————LOGICAL RECORD————————    ——SPACE——
                      SIZE   TYP    EOF    LIMIT   R/B   SECTORS #X MX
XL          NMXL      128W   FB     38215  4096000 1     38912   19  *
```

Look at that limit, I thought that was real large. Recently, I pulled out my HP41CV calculator, the MPE/XL file system and TurboIMAGE/XL manuals and started doing some figuring. Let's take it from the top. The MPE/XL file system manual, along with the Intrinsics manual say that the biggest files you can open are as follows: If you use HPFOPEN with the short mapped option, it's 4 megabytes or 2^22-1 bytes., If you use the long mapped option, it's 2 gigabytes or 2^31-1 bytes. How does that translate into old-fashioned sectors,(hey, I'm an old- fashioned guy), well here it is, just divide 2^31-1 by 2^8 bytes in a sector which gives (let's see now, 31-8=23) 2^23-1 or 8,388,607. Let's try it:

:BUILD DENYS ; DISC=8388607; REC=-256

or 2^23-1 records of 2^8 bytes, which works out to 2,147,483,392 or 2^31-1 bytes. Let's try a bigger file:

:BUILD DENYS ; DISC=8388608; REC=-256

EXTENT SIZE EXCEEDS MAXIMUM (FSERR 106) BUILD OF FILE DENYS.SOURCE.USM FAILED. (CIERR 279)

Well, it looks like my calculations were on the nose. There is a limit, easily reachable on MPE/XL for file size. But beware of that mesage about extent size exceeding maximum, it's very misleading. All this discussion brings us to this with regards to TurboIMAGE/XL; The extents specification seem to play some role, however we get affected only at creation of the datasets. MPE/V used to play round-robin games extent by extent. Whichever disk drive was next on the configuration list for that class of disk was where the next extent was placed. So during the creation of a large dataset, provided that we are the only ones creating a file/extent at that time, this would happen:

Configuration for class DISC: 1,2,3,4,5

For 8 extents they would be placed, starting at say ldev 2 :

Extent #	Disc ldev
1	2
2	3
3	4
4	5
5	1
6	2
7	3
8	4

Not so on MPE/XL. No, on MPE/XL each extent is on a most-available-space order. Which means that the next extent is going to be created on the disk drive which has the most space left, on that volume set, for the volume class. Even further, there seems to be a tendency for MPE/XL to attempt to place the entire file on the same drive. I guess this is predicated by the fact that one would try to do a BUILD or an HPFOPEN/FOPEN and allocate all the extents at that time. So if you reload the volume set, you will find that your database datasets will "clump" up on the same volume, set by set, given enough space on that drive of course. Let's look at a few last things and the end this numbers games.

In a database the largest set capacity is, you guessed it: 2^31-1, which works out, again to : 2,147,483,392 records. This, according to the TurboIMAGE/XL manual. How does that compare to TurboIMAGE/V on MPE/V? Well, on MPE/V the file size was dictated by the extents, a file could have a maximum of 32 (2^5) extent, and each extent could have a maximum of 65,535 (2^16-1) sectors, each sector was, of course, 256 (2^8) bytes. Therefore the biggest file is (2^5 *(2^16-1) * 2^8) or (2^29) - (2^13) or 536,862,720 bytes. You can also calculate it like this: 32*65535*256 bytes, but I find it easier with exponents. (Big numbers boggle my mind.) As you can see, the improvement is (2^31)-((2^29)-(2^13)) or about 2^2 or about 4 times what is was on MPE/V.

MPE/XL 2,147,483,392 bytes (2^31-1)

MPE/V 536,862,720 bytes ((2^29)-(2^13))

At the SIGIMAGE meeting in Reno, March 1991, one of the enhancement which was requested was the possibility of increasing a dataset to 4 gigabytes and a further request was to increase it beyond 4 gigabytes. If the current size limitation is a concern to you, please contact SIGIMAGE and let them know.

D) MPE/XL 3.0 and TurboIMAGE/XL, the present.

With 3.0, HP brought in a lot of major enhancements. Let's look at a few of them:

1- Performance enhancements:

The buffspecs are now fixed at 1280, and the buffers are searched with a hashed table, not serially. So whatever settings you use, they are now ignored, the buffspecs are only kept for compatibility's sake. There are also changes in the modification intrinsics to enhance concurrency, but this will be the subject of a future presentation.

2- MDBX Multi-Database Logging:

It is now possible to log transactions which affect multiple databases, up to 15. These transactions will be recovered as one unit of work by DBRECOV. To use this feature, there are 2 new modes for DBBEGIN.

Mode 3: For multiple databases transaction, generates multiple log records, one per database.

Mode 4: Same as above , but only generates one record.

Note: All databases must log to the same logfile.

DBBEGIN and DBEND denote the start and end of a STATIC transaction.

3- Dynamic Rollback.

On TurboIMAGE/XL, thanks to the XM, it is now possible to rollback a transaction, with one intrinsic. Of course one must define a transaction and this is done with a DBXBEGIN/DBXEND pair. If at any time before the DBXEND is called the system is interrupted, the program aborts or DBXUNDO is called, the transaction will be rolled back. A transaction within DBXBEGIN/DBXEND is known as a DYNAMIC transaction. Throughout the transaction, strong locking must be maintained and if locks are released before the end of the

transaction, or if any error is encountered by the modification intrinsics (DBPUT/DBDELETE/DBUPDATE) all other IMAGE calls will fail until a DBXUNDO is performed (all the intrinsics will return a status of -222 until the DBXUNDO).

The sequence would look like this:

DBLOCK

DBBEGIN <<if so desired, but mode 1 only>>

DBXBEGIN

.

.

transaction steps (excludes any calls to dbunlock) .

If any modify error is encountered, goto DBXUNDO now! .

DBXEND or DBXUNDO

DBEND <<if so desired>>

DBUNLOCK

At this time, it is not possible to define a dynamic transaction which encompasses multiple databases. It will surely come, but be aware that it will most assuredly be limited to databases within the same volume set. In the case of system aborts, the transaction will be rolled back at the first DBOPEN to that database. It's like an expanded ILR. When the volume set is brought on-line, and at the very first DBOPEN to any database on that volume set, XM will spawn off a 00 file for every database which had an active transaction at the time of the sysabort. Then as each database is DBOPENed, TurboIMAGE/XL takes over and applies the rollback for the incomplete transactions contained in the 00 file.

Note: You can't use DBXBEGIN/DBXEND if AUTODEFER is enabled or if the database is opened in mode 2! There is one mode for DBXBEGIN, 1, but there are 2 modes for DBXEND, 1 and 2. Mode 1 just signifies the end of the dynamic transaction. Mode 2 does the same but it also forces the write of the logging buffers to the disk and if logging is enabled, it will also force the logging buffers in memory to disk. This would probably degrade system performance but would enhance database stability. It is now possible, if only dynamic transactions exist for a database, to run without logging enabled for the database. However, logging is used for far more than just recovery, and it is still required in case of media failure (aka disk crash).

4- A new DBUTIL flag MUSTRECOVER.

This flag is set/reset with DBUTIL. Once the flag is set, the first person to DBOPEN the database with write access, sets another internal flag. When the last person with write access DBCLOSEs the database, this other internal flag is reset, signifying that the database in consistent. If the system crashes whilst there is one

or more user with write access in the database, the flags are then interpreted to mean that the database is i inconsistent and must be recovered before it can be used. Until the database is recovered, or the flag reset with DBUTIL, only DBOPEN mode 7 (exclusive read access), is allowed.

Note: Enabling MUSTRECOVER for a database forces logging to be enabled also. Thus, if your database is not set up for logging, the MUSTRECOVER facility cannot be used.

5- Expanded DBINFO.

DBINFO has a series of new modes which return a lot of useful information to the calling process:

401: TurboIMAGE logging information.

402: ILR information.

403: Dynamic Transaction information.

404: MDBX transaction logging information, including RDBA.

901: Database NLS code.

E) MPE/XL and TurboIMAGE/XL, the future.

In an upcoming version of TurboIMAGE, the Third-Party indexing products will be part of TurboIMAGE. This will meet the most requested enhancement for TurboIMAGE, generic key searching. Of course these products are not free, but they offer much more than just generic key search, And they will live within TurboIMAGE. Another widely requested enhancement, the capability to update a critical field, search or sort, in a single detail record, is to be made available by HP. This will need to be covered more at a later date.

In Reno, at SIGIMAGE, there was quite a series of enhancements which were enumerated, and I will let the officials of SIGIMAGE be the folks to dissiminate this information. Suffice it to say at this time, that the 2-day meeting was really excellent and fruitful, and that folks will be happy with the requests which are being forwarded to HP for review. As a note, HP's participation in the meeting was intense. HP's commitment to TurboIMAGE is very apparent.

If you are interested in participating in SIGIMAGE, please contact Steve Cooper with Allegro Consultants in California.

In retrospect, one can sit back and ponder the history of TurboIMAGE/XL. I wish I would have taken the time to record much more information on the different releases of MPE/XL and the impact on TurboIMAGE/XL performance which each new release. But I did not. Maybe someone else did, and if so, please share it with us. As a last observation, TurboIMAGE/XL now only shares the squeleton, the static structure with TurboIMAGE. The functionality has changed and the run time environment is evolving much faster than most people are aware of up to now.

Paper 3920

The Future of IMAGE is SQL

An Outline of the Simple Steps Necessary to Merge IMAGE & SQL

Wirt Atmar
AICS Research, Inc., University Park, NM 88003 USA
(505) 524 9800 • FAX: (505) 526 4700

At the most recent Southern California Regional Users' Group (SCRUG) meeting in Los Angeles (May 8-10), Charles Finley, SCRUG Chairman, asked during the final day's luncheon, "How many of you believe, given the way things are going, that MPE is dying?" Almost everyone in attendance raised his hand. Charles then asked, "How many of you believe that HP is purposefully trying to kill MPE?" About half raised their hands. Charles asked a third time, "How many of you think HP is *not* purposefully trying to kill MPE?" The other half raised their hands.

I was in this second group. I sincerely believe that HP has absolutely no intention of trying to kill the HP3000, MPE, or IMAGE; indeed, I believe that the reaction common in the HP3000 installed user base has come as a complete surprise to the people at corporate. But the reaction should not have been all that unexpected. It is a direct result of HP losing sight of some very fundamental objectives, primary among them is misunderstanding the desires and needs of its long-term customers.

The lynchpin of the HP3000's success is and always has been IMAGE. To most users, the HP3000 *is* IMAGE. However, for the last several years, HP corporate marketing has felt strong pressure to appeal to current DEC and IBM users, believing that these people are their future market. SQL is the common database structure among these users, and that has been the only reason for HP's preoccupation with SQL. The result has been a marketing mistake that very nearly duplicates that of Coca-Cola.

No corporation has any intention of doing themselves or their customers harm, but Coca-Cola, by intensely tracking marketing reports of falling market share, and by carefully listening to "industry watchers" and stock market analysts, with great deliberation and prudence, came to the most extraordinary conclusion in American corporate history: on one day in 1984, they simply stopped manufacturing their 100 year-old product in favor of New Coke. What lay at the heart of such a monumental mistake? Undoubtedly, the primary reason was that no one who participated in the decision was a *user* of Coke. If they had been, abandoning the product would have been unthinkable.

The situation has been startlingly similar at HP. For some time now, HP has paid far too much attention to its stock analysts and industry watchers. And HP has paid far too little attention to its users. It was consumer reaction that turned Coke around. It will be user reaction that similarly corrects HP's path.

There is however a very simple solution to what otherwise might well become a grave situation. Merge SQL and IMAGE. It is to everyone's benefit. It can be done. And it can be done in a manner that is wholly compatible with all prior usage of IMAGE. And it is surprisingly easy to do.

When the participants at the recent SIGIMAGE meeting in Reno (March 4-5) were asked for their opinions as to the most pressing enhancements to IMAGE, the number one vote-getter was to put a full

read/write SQL shell on top of IMAGE. IMAGE, as it is now, is a world-class database. No competing database is as efficient, as robust, or as reliable. But like all things well done, IMAGE can be improved. An SQL interface shell would represent a significant advance for three reasons: (1) a standard query language is the enabling technology necessary for the "open systems concept", an idea which will only continue to grow in importance. (2) An SQL shell would minimize new user resistance when moving to the HP3000 from an environment in which SQL is the common database structure. And, (3) the shell would actually promote migration from SQL to IMAGE, and thus the expansion and market acceptability of IMAGE. Maximally efficient database structures will always be a desired quality, and no database structure is more efficient than IMAGE.

Putting a full read/write shell on top of IMAGE would probably be accomplished in two stages. The first logical step would be simply to expand the capabilities of the existing ALLBASE/TurboConnect product to full read/write capability, a move HP is now considering. But the *bold vision* is to ultimately merge the two products into one, so that there is only a single IMAGE-SQL database, *completely bundled in with the FOS*, with dual-level access and two interfaces, one through standard TurboIMAGE intrinsics and the other through standard SQL.

What is particularly appealing about working towards the complete merger of the two database structures is that the process does not have to be done all in one step. Intermediate versions of IMAGE can be released which would represent significant improvements while work continues towards a complete ANSI-standard SQL-IMAGE merger. As Ken Sletten, SIGRAPID Chairman, has said, "This is a way of having your cake and eating it too."

There are however several enhancements that must be made to TurboIMAGE before an SQL shell can be added. It is no coincidence that these same enhancements have been among the most commonly requested enhancements to IMAGE over the past 15 years. There is a certain inevitability in the design of any well implemented database structure. The enhancements which are necessary to prepare IMAGE for an SQL merger are:

- *Add the capability to add or drop indexes easily.*
- *Add the capability to specify the type of index to be used (hashed or b-tree).*
- *Add the capability to add or drop datasets easily.*
- *Add the capability to add new dataitems to the end of a dataset easily.*
- *Add a KSAM-like capability for overlapping keys.*
- *Optimize IMAGE for maximally efficient serial reads.*
- *Add automatic capacity management of detail datasets.*

These few enhancements are among the most commonly touted advantages of SQL databases, although they have nothing to do with SQL per se. They should be, and should always have been, part and parcel of the standard IMAGE database, too. IMAGE is particularly amenable to these modifications, and none of them are difficult.

When comparing IMAGE and SQL, it is important to clearly understand what is an attribute of the database and what is an attribute of the query language. The basic SQL commands are listed in Table 1. The first seven commands (CREATE, DROP and ALTER) deal solely with restructuring the SQL database. Only the last four (SELECT, UPDATE, DELETE, and INSERT) are part of the query language.

Because there is this very clean break in structure, it's not only possible—but quite

reasonable—to have *dual-level access* to the same database. The SQL query language is only an upper-level shell imposed on a generic database. IMAGE intrinsic-level access would continue to work as it always have in an IMAGE-SQL merged database, and will likely always be the preferred high-speed, high-efficiency path into the database.

CREATE TABLE CREATE VIEW CREATE INDEX
DROP TABLE DROP VIEW DROP INDEX
ALTER TABLE
SELECT UPDATE DELETE INSERT

Table 1. The basic commands of SQL

The Steps Necessary for Merger

First, a full read/write SQL query language shell must be created.

The SQL query language interface to IMAGE already partially exists. It is currently half built, and is called ALLBASE/TurboConnect (ATC). ATC is, at the moment, a read-only shell. To be sure, a read-only product is easier to design than one with write capabilities. Considerations such as locking and rollback recovery need not be addressed in the design of a read-only interface that must be accounted for in a full read/write shell. But, once the read-only interface is up and working, as it is now, 80-90% of the task has been accomplished. HP is now considering whether or not to complete the task, and to its great credit, is asking for your thoughts on the matter. The person to write is:

Douglas Dedo
IMAGE Product Line Manager
Hewlett-Packard Company
19111 Pruneridge Ave. M/S 44MP
Cupertino, CA 95014 USA

FAX:(408) 447-0125
(408) 447-0872
(408) 447-4966

Doug Dedo has emphasized that the size of your organization is not important. Everyone's comments will be given due consideration and are equally valued.

Second, IMAGE must be made more plastic.

Virtually all of the current attractiveness of SQL lies in the freedom associated with restructuring the database, not in the query language. With a sufficiently plastic database structure, it is not necessary to predict all future access paths at design time. This plasticity often saves significant overall program development time. As needs evolve, a well-designed database structure can often be modified without affecting existing programs, thus greatly simplifying application program maintenance.

Now that critical item update capability will soon be a reality, IMAGE can be made to be at least as plastic as SQL, if not more so. With critical item update in place, data-items which are not now keyed search items can be made to be so with guaranteed certainty that the change will have absolutely no effect on any pre-existing application program. Moreover, HP has all of the remaining code in hand necessary to make IMAGE as plastic as SQL with its DBChange product. Only a very small portion of the code is necessary to bring IMAGE up to SQL standards: the capacity to add or drop detail datasets, the capacity to add or drop indexes from detail datasets, and the capacity to add dataitems at the end of an existing dataset. Adding this code to IMAGE's standard capabilities will not be particularly expensive or difficult.

Third, b-tree generic searches must be added to IMAGE.

Adding generic search capabilities has been one of the most commonly requested enhancements to IMAGE, second only to critical item update capability. And because a true SQL shell can't be added until b-trees exist in IMAGE, this is one of those rare opportunities where a design process works synergistically in everyone's favor. There exists a very simple method available to invisibly add high-speed b-trees to IMAGE. In the procedure outlined in the appendix, IMAGE b-trees would take full advantage of the master dataset hashing algorithm inherent to IMAGE and require only the most minute of changes to IMAGE master datasets. The method is surprisingly simple to implement, and surprisingly fast. DBGETs (database retrievals) would be, on average, 2x faster than equivalent KSAM retrievals. But more impressive yet, DBDELETEs and DBPUTs of b-tree search items would be 5-200x faster than KSAM.

IMAGE's hashing algorithm is an extremely efficient search algorithm—but the search technique works only if you know the whole key. If you do not, as is often the case, you are condemned to a serial read of the dataset. "Generic", partial-key searches are the mechanisms which eliminate the necessity of having to perform time-consuming serial reads of datasets. The DBFIND procedure outlined in the appendix takes advantage of the best attributes of both b-trees and hashing keys.

Because the method is surprisingly simple to implement, measurements on the performance gain over KSAM generic searches can be precisely simulated and measured. The results of a set of simulated IMAGE b-tree experiments using a payrecord dataset that was duplicated into both IMAGE and KSAM formats are presented in the figure below.

The mechanism proposed here is simple enough and easy enough to be included in IMAGE as a universal feature, available to

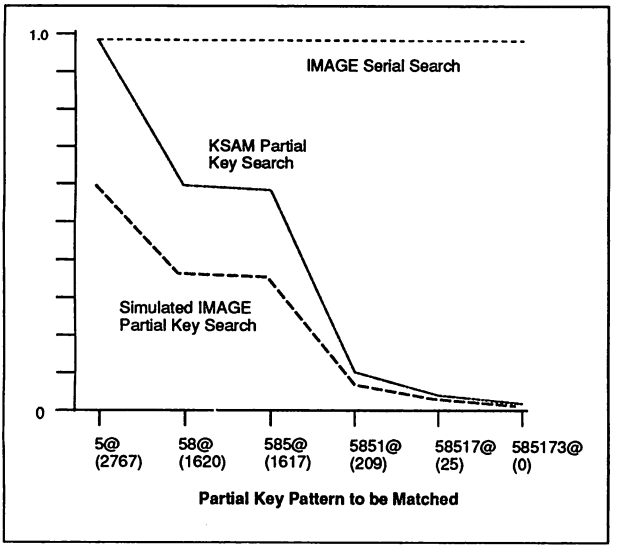


Fig. 1. The relative speeds of KSAM and IMAGE b-tree generic searches, as compared to an IMAGE serial read. The generic search pattern in presented along the X axis. The "@" symbol represents a wild card. The number of records qualified for each pattern is indicated in parentheses.

The trial programs used identical code, other than the necessary differences in intrinsic calls.

all users, without charge. The advantages of this type of b-treeing are significant: (1) The method is easily implemented. (2) It is guaranteed to be fully compatible with all prior usage. (3) It will be quite fast. On average, it will be much faster than KSAM b-trees—and by implication, most other SQL implementations. (4) Under most circumstances, the method presents quite minimal (often no) overhead. The b-tree structure will be modified only when the master dataset is modified, a rare event. (5) The technique will not require the massive disc space usage characteristic of other external b-trees. (6) The b-tree is automatically shared among IMAGE's paths (up to 16 paths/master dataset). And, (7) master datasets may be quickly "converted" from normal hashing keys to b-trees and vice-versa.

Fourth, this new, complete, much-enhanced IMAGE-SQL database should be bundled into the price of the HP3000.

Rebundling IMAGE is the most important enhancement request of all. The presence of one common database on the HP3000 has been the glue that has bound users in London, England and Paris, Texas together as a single community. It cannot be repeated too often that the lynchpin of the HP3000's success is and always has been IMAGE. It is entirely arguable that if IMAGE had not been bundled into the HP3000 in 1977, the HP3000 would not exist today.

The few steps outlined here are not original ideas. Indeed, they were part of the promise that HP made to itself and its customers in 1986:

> "HP is introducing ALLBASE, the dual database-management system, designed for all the new HP Precision Architecture machines. ALLBASE will enable your customers to have both relational and network access to data in one all-encompassing database-management system. Your customers no longer need to choose a DBMS that fits some of their application needs and force fits the remainder. ALLBASE will be the foundation for HP solutions for many years to come. On its second release, ALLBASE will even provide *dual-access*. Dual access will enable our customers to access IMAGE data through the relational interface via SQL, the relational data language, and through the HP-IMAGE interface via IMAGE intrinsics. *Dual access will be a unique competitive advantage for HP"* (the italics are HP's).
>
> —Terrie Murphy/CSY
> *Information Systems & Manufacturing News,*
> March, 1986.

Five years have elapsed since this paragraph was written. Should another five years pass without the promise coming true, the HP3000 and MPE will surely die. As with every evolving process, the HP3000 is either actively growing or dying. It cannot stand still. But the enhancements outlined in this article are relatively simple. They could all be in place in two or three years. The original intention was that all of this structure would be fully bundled into the price of the HP3000, as IMAGE had been for virtually all of its commercial life.

Why has HP backed away from its original promise? For very human reasons, no doubt. Time moves on and people change jobs. And marketing pressures demand instantaneous solutions to lost sales. So for reasons of meeting monthly sales quotas, the inevitable thousand technical problems that plague any project, and a loss of vision in a sucession of managers, short cuts are taken and existing, third-party products are touted in favor of investing the necessary time and manpower to bring IMAGE to full competitiveness. But with a very little encouragement from the user community, I suspect that the promise can be kept alive.

Appendix

A Simple Method to Add B-Trees to IMAGE

The term "b-tree" is short for *binary-tree*, a decision tree where you are presented with a series of greater-than/less-than decisions. B-trees characteristically have a number of decision layers, simply because not much sorting can occur at each decision layer with only two responses. Nonetheless, individual items, or groups of items, can be isolated in a large set surprisingly quickly. Twenty questions can isolate a single item in a set of a million records.

But more importantly, b-trees are the mechanism which allows *generic search* capability. A generic search is one such that the query asked requires finding all of the entries in the database between two dates or the names that begin with "SHA". IMAGE, as it presently stands, cannot directly find such entries. IMAGE's keying method is called a *hashing* key technique. The advantage of hashed keys is their extraordinary efficiency. The disadvantage of hashed keys is that you can only search for one key item value at a time, and you must know the search item value in its entirety. If you do not, as is often the case, you have no choice but to serially read every record in the dataset.

However, b-trees can be added to IMAGE quite easily, and in a manner that is completely compatible with all prior usage. If a key were to be specified as a b-tree key, a normal IMAGE master dataset (manual or automatic) would be built as a "hashing" master as it always has been. Indeed, all normal rules of IMAGE would apply other than the master dataset would be marked as a b-tree key. This "marking" would be done by simply using an unused word in the master dataset's user label, and that one word would be a pointer to the corresponding b-tree's file name (Fig. 2). (Every IMAGE dataset, either master and detail, has a 128-word user label attached. Only 6 words in the user label are currently used; 122 words are unused).

Otherwise, the construction of a master dataset would be unchanged. The only difference would be that a b-tree file would be invisibly "attached" to the master dataset. A consequence of this simple structure is that a normal IMAGE (hashed) master dataset could be converted at any time into a b-tree key. The backwards conversion would even be simpler. The "attached" b-tree would simply be dropped.

The mechanism that accounts for the great efficiency of IMAGE b-trees is that the structure of the b-trees would not need to be modified for the majority of fundamental database transactions. The key item values in the b-tree would be designed to automatically maintain a one-to-one correspondence with the search item values in the master datasets, which are, as a consequence of the way IMAGE is designed, guaranteed to be unique. The only time that the b-tree structure would need to be changed is when a key item value is either added or deleted from a master dataset, a relatively rare event. The simple lengthening or shortening of a chain when records are added or deleted from a detail dataset would have no effect on the b-tree. Thus, for most transactions, the b-tree would represent no overhead cost at all.

How would a generic search work using these IMAGE b-trees? Presume that the query you wished to ask was (in QUERY syntax):

Find invoices.jobnum ib 15,60

where *ib* means "is between". Because a b-tree is attached to the JOB-ID master dataset (as shown in Fig. 2), IMAGE would first "walk" through the b-tree, identifying the qualifying search item values within the specified range (in this case, the values are 16, 18, 23, 30, and 42). Once this has been done, and because the search item values are now known in their entirety, the hashed masters of IMAGE can be used to

full advantage. Each search item value would be applied in turn in a chained search of the INVOICES dataset. The search would end when all of the qualifying records in INVOICES have been found and recorded.

Quite obviously, a few of the standard IMAGE intrinsics would have to be modified to take advantage of the new b-trees. But again this modification can be made invisible to all prior use. DBFIND (mode 1), as it now exists, inherently assumes an "equals" relational operator. Very little work would be required to modify DBFIND. The addition of six new modes would be the only necessary changes to the intrinsic call.

DBFIND(*base,dset,mode,status, item,argument(s)*)

where

mode = 1 implies "equals"
2 implies "greater than"
3 implies "less than"
4 implies "greater or equals"
5 implies "less than or equals"
6 implies "is between (incl)"
7 implies "is between (excl)"

Multiple arguments would only be used in conjunction with modes 6 and 7.

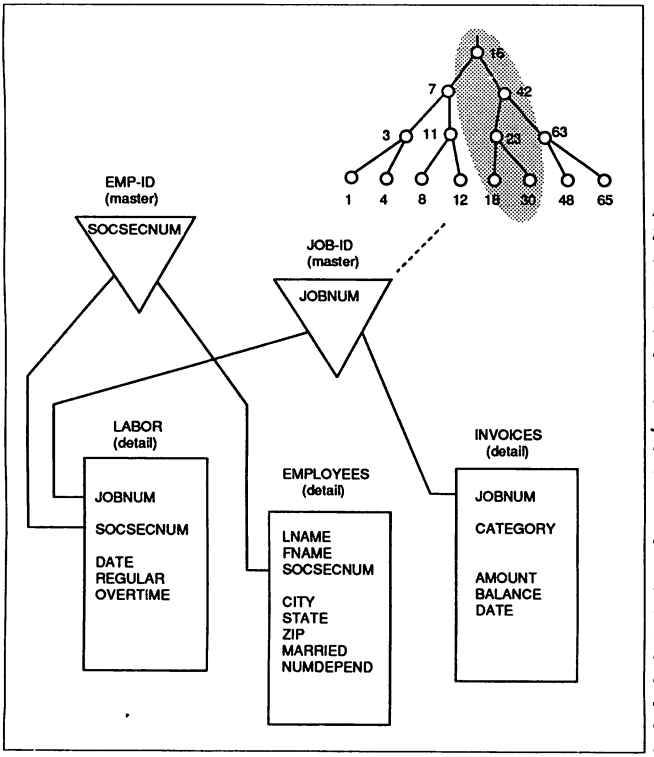


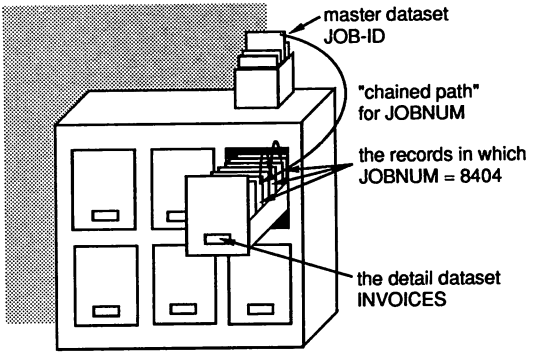
Fig. 2. An automatic master dataset with a b-tree attached. Because there is a one-to-one correspondence between the key item values in the master dataset and the b-tree, walking through the b-tree from the minimum job number value to the maximum allows a very rapid way to find all of the appropriate invoices in a hashed "generic" search.

Each of the specific job-number values shown in the shaded portion of the b-tree would be applied in turn to the chained searches until all of the qualifying invoices had been found and recorded.

Acknowledgements

A slightly modified version of this article entitled, "The 110% Solution", appeared in the July, 1991 issue of HP World, *a European HP user magazine.*

Various drafts of this article were reviewed by Valerie Atmar, Steve Cooper, Mark Halstead, Charles Hill, Steve Martin, Alfredo Rego, Stan Sieler, Ken Sletten, Frank Smith, Fred White and René Woc. The author wishes to express his sincere appreciation to everyone for their careful reviews and thoughtful comments.



The TurboIMAGE Database

VENTURING INTO ALLBASE

C. Bradley Tashenberg
Bradmark Technologies, Inc.
4265 San Felipe, Suite 800
Houston, Texas 77027
713-621-2808

After 18 years of familiarizing ourselves with IMAGE, HP now has the audacity to tell us that there is a new DBMS on the HP3000 which we should seriously consider. It is known as ALLBASE/SQL and is not getting an enormous impetus from HP.

What is ALLBASE/SQL? And more importantly, why should we consider using it?

SQL, (Structured Query Language) is an ANSI standard in relational database access. This means that even though the underlying structure may be, and usually is, different between computing platforms, the access methodology looks the same to the end user. ALLBASE, the HP relational DBMS, is devised in accordance with the 10 rules spelled out by Dr. E.F. Codd.

Beyond just introducing a new concept in data management to the current IMAGE users, there are other concerns which require careful deliberation before jumping into the RDBMS pond. One of which is: "How deep is the pond?", while others relate to the wealth of technology, which we as IMAGE users have come to expect.

Conceptually, ALLBASE/SQL offers a plethora of features not found in IMAGE. However, as with most new ideas, there is very little established today through which one can take advantage of these features. HP has been very effective in selling the merits of ALLBASE/SQL to many of the larger 4th GL companies and VARs, but at this time, very little software is on the shelf. There is currently a sizeable development effort going on: Cognos is developing Powerhouse for ALLBASE/SQL, Infocentre is to announce shortly a Speedware version for ALLBASE/SQL, and both Collier-Jackson and ASK have committed to have a future release of their fine products running on ALLBASE/SQL. Again, very little is available now or in the near future.

Besides the concern of a lack of application software and utilities presently available, there is the one about "ye olde" learning curve. Even though there are some similarities between ALLBASE/SQL and IMAGE, there are vast differences in concept and implementation and use of these database management systems.

Perhaps the most fundamental difference is that ALLBASE/SQL consists of an extensively layered structure, which can be changed dynamically at any time. This, more than anything, may be a source of difficulty in understanding. Let's examine this a little bit.

IMAGE is quite rigid in its structure. It has a root file, which contains the database dictionary and a series of associated datasets.

The datasets are either masters, accessible through unique key value, or details, which house related information chained to one or more masters. Paths are hard-linked between master and associated detail entries.

ALLBASE/SQL does not have the hard-wired relationships between objects. The relations between the objects are declared dynamically. ALLBASE uses the concept of Data Base Environment (DBE) to store all the objects such as tables, indexes, views, modules, groups, etc. The views, or stored select commands, can be used to create some semblance of permanent relationship between objects. They can also be used to limit the access to information within the various tables. Layers of security, known as groups and users, can also be declared to further protect the information.

The use of indexes, views, and groups used in conjunction with the tables can greatly enhance the accessibility and security of the data within the DBE.

One area of similarity between ALLBASE/SQL and IMAGE is in the indexes. ALLBASE/SQL has currently 4 types of indexes available:

HASHED
UNIQUE
NORMAL
CLUSTERED

With judicious use of these indexes, ALLBASE/SQL can be made to appear very similar to an IMAGE database. Consider first the master datasets; one can create a table with a hashed index on the columns (or fields) and thereby make the data behave the same way as it would in an IMAGE master set, as far as uniqueness of key and hashed insertion and retrieval of the data is concerned.

In order to emulate detail datasets with the data dependencies of master entries, one can make use of the normal (balanced tree, or b-tree indexes) and referential constraints on other existing tables.

The indexes, unique or normal, resemble the old Index Sequential Access Method technique and the KSAM approach of HP. The clustered index is an attempt to cluster rows (or records) with similar key values within the same areas of the tables.

By using indexes, information can be retrieved quickly and efficiently. So, although ALLBASE/SQL professes not to have any fixed or concrete structure, it is usually used with these index structures. Furthermore, the overhead required to maintain these indexes is not less than that which is required to maintain the IMAGE hashed and chained information. ALLBASE/SQL's layering isdefinitely more substantial. Benchmark tests performed by Hewlett-Packard have demonstrated that ALLBASE/SQL is 25 to 50% slower than TurboIMAGE for similar applications.

So why would anyone want give up performance? The reason is very simple: Industry Standards. Application developers wish to develop to known industry standards, and one of these standards is SQL. Thus, by developing applications complying to standards such as POSIX and SQL, one should be able to port these applications to any other platform adhering to the same standards.

Am I in favor of this concept? Of course I am! Am I in favor of this approach? At this time, no, I am not!

To recommend to people that they should go from a technology that has a high satisfaction level and comfort zone, to one that is still in its infancy is too radical a change. There must be a transition!

Jumping from IMAGE to ALLBASE/SQL is akin to jumping from HP3000 MPE/V based machines to MPE/XL without a compatibility mode. It is both risky and dangerous. It would require a major rewrite of all database accesses, and data storage concepts. At this stage, there are no old friends such as Query, Powerhouse, Speedware, Visimage and others, to help make this transition a success.

However, there is a solution for the curious. With MPE/XL 3.0, HP introduces ALLBASE Turbo Connect (ATC), which provides read-only SQL access. From what I have heard through the rumor mill, there is a strong possibility of supporting the ATC through the new IMAGE Open Architecture. This will further enhance and speed up the data retrievals by using the indexing packages.

Using this approach, there is no gamble. If you don't like SQL because it's not working for you, you haven't lost anything. Conversely, if you like SQL, you have a safe and easy method to effect the transition, without dramatically disturbing your current operation.

So, it looks as though IMAGE users may not have to forsake IMAGE to take advantage of SQL. They may very shortly have the best of both worlds, and isn't this the way it should be?

Improve your relationships and venture forward! If you do it conservatively, you have nothing to lose!

March 1991

Paper 4101
Windows - When the Time is Right

Russell T. Bradford
Bradford Business Systems, Inc.
23151 Verdugo Drive, Suite 114
Laguna Hills, CA 92653
(714) 859-4428

Windows is the buzz-word of the 90s but to that end, not too terribly many companies have plunged headlong into commiting their entire organizations computing strategy to a Windows based environment. Why might this be? Well, there are plenty of good reasons- like cost, complexity, lack of applications, time, and more. None the less, I strongly feel that the time has come, the move toward the Graphical User Interface (GUI) is already well underway, and your organization should not be left behind.

The reasons for implementing a GUI outweigh the reasons not to. Many of the reasons used to justify *not* going to a GUI are the same reasons why you *should*. Reasons like cost, complexity, lack of applications, time, etc. Let me explain.

Many people feel that the costs of installing GUIs as a company wide standard are too high. This reasoning, as far as it goes, is quite sound. You need more powerful and more robust computers to run a GUI, requiring more memory and more disk space, plus a faster processor. With the cheaper non-GUI solution, you make up for the additional hardware costs with people costs. While GUI applications themselves don't necessarily run any faster than the character based counter parts (they often run slower), they are far easier to learn and use. This one aspect alone can save many times the cost of the hardware and development costs. Ease of use translates into less time spent on training in tandem with more and better use of applications. The ease of use comes from the fact that all well behaved GUI applications work in essentially the same way. Once the user learns how to use a word processor they can easily migrate to a spreadsheet, text editor, calendar or graphics package with little or possibly no training. This boils down to savings by not having your people sitting in class for a few extra days saving not only the cost of the class but also receiving a faster payback on the employees salary.

The complexity of creating a GUI application is nothing to gloss over lightly. GUIs are a *bear* to program. My company, Bradford Business Systems, has been developing an application which uses all the different GUIs; Windows, X-Windows, and Presentation manager. The cost and effort to program for these are astronomical in comparison to character applications. As mentioned earlier, one needs bigger, better, and more expensive equipment. A whole host of software is required just to get started and even more if you wish to cut the time and effort required. Experience is at a premium. There just aren't very many

experts for hire at present, and consultants with this sort of background are few and far between. While complexity is a great reason to stay way from GUIs, making complex tasks easy is the whole reason GUIs can be worth the effort. For example, secretaries commonly are asked to use a desktop publishing package to assemble complicated documents or manuals. In years past this job was relegated to professional typesetters at a cost of over $20 to $30 per page. Cumbersome command oriented packages were able to do a small portion of the work that todays desktop publishing packages do, with far more effort and a much steeper learning curve. The everyday use of desktop publishing would never have been possible without a GUI. This same translation of more complexity in programming can apply to everything from accounting to data entry to statistical analysis to report creation. The efforts of a few expensive programmers can translate into far greater productivity to dozens or even hundreds of end users.

Lack of applications seems to be a good reason for not moving to a GUI, at least superficially. First off, although there are possibly 100,000 non-windowed applications available for DOS, and hundreds if not thousands of non windowed applications for most proprietary operating systems, all but a few are used by only a handful of users each. The reasons for this are many. In some cases it's poor quality. In others it is the lack of suitability to the task at hand while others lack certain features required or desired. The biggest reason why all but a few packages are not in wide spread use is that the costs of marketing software today are astronomical and thus the little guy with a better mousetrap stands little or no chance of letting the world know. One other reason why perfectly good packaged software never goes into wide spread use is the fear by consumers that since the product isn't widely used there will be little help in learning and using the product. These last two reasons are most likely the primary reasons why probably 95% of the applications available today are used by fewer than 1,000 individuals or organizations.

It is this last reason again why a GUI can make relatively obscure software less threatening. By maintaining the same look, feel, and operation of all other GUI applications, users can follow their instincts in learning the application. As part of the GUI guidelines, every item on every menu can display specific help simply by pressing the f1 key. An operation like copying text using a word processor is virtually identical to copying cells from one place to the next in a spreadsheet. A user who has become proficient in a word processor should be able to pick up a desktop publishing system in relatively little time. For Microsoft Windows there are already hundreds, possibly thousands of applications already available and many many more on the way. There are dozens of applications for each of the areas of accounting, data base management, education, engineering, games, graphics, languages, chemistry, fashion, insurance, legal, science, statistics, transportation and more.

While it is still likely that you may need to change from some of your current applications to different ones which are based on a GUI, others have or will be converted to Windows in the near term. To that end, switching to a new word processor, text editor, spreadsheet or other application might be a beneficial experience in that the training and time spent learning a new, GUI based application will pay off in increased productivity later on. To that end, most applications which might be converted straight across from their character based counterparts tend to break the rules for GUIs and thus eliminate the primary benefit of commonality across the board. These applications should be avoided like the plague because they tend to instill bad habits that are hard to break downstream. We see this tendancy every day with our own customers. They want our product to work like the old clunker they have been *living with* for twenty years. While admittedly there would be a short term benefit of a slightly shorter learning curve, they will pay a penalty for the next twenty years having to rethink other applications that don't conform to their old fashioned, non-standard utility. When we explain this to our users we gain reluctant acceptance. As little as a week later these same individuals are squealing with delight over how easy things are and how much more capability they have and can readily gain access to. Most of these people tell us that once they are hooked on a GUI, they would never go back. Our users, programmers, are the hardest sell of all and so if we can convince them, your users should be a snap.

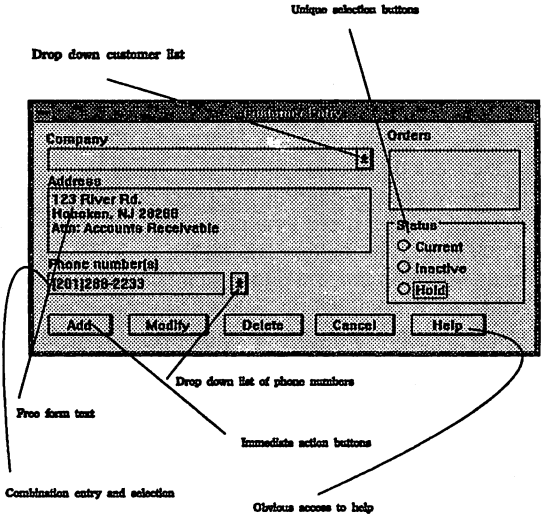
One last point on the "lack of applications" argument. Most companies don't live with just canned software but also have in house applications. If you buy the point that GUIs are beneficial in terms of productivity and also that if your in house application were a GUI that your users would be more productive, then is it worth the effort? Consider the move to a GUI from a competitive aspect, your company against the competition. While I would be hard pressed to argue that the first company to start using a GUI for their main stream application will be the most successful, I might be able to argue that the most productive will be. That being the case, if a GUI makes your users more productive and thus more profitable, it should be a key part of your decision to move to a GUI.

Time is another excuse why one might not dive head long into GUIs. It does take much longer to create a decent GUI application. There will also be better tools for developing GUIs after more time passes, but that will always be true no matter how long you wait. The bottom line on the time aspect of this decision is that time and the competition are working against you. The longer you wait, the more chance that your competitors will streamline their entire operation, including DP. During my many years of consulting, I saw far too many shops dealing with the false economy of waiting for faster, better, cheaper hardware and software while their users sat at their terminals waiting minutes for information that should have been there in seconds. Productivity wasn't even considered in these shops, just time and getting the best deal on hardware and software. Many of these companies don't even exist today, partly because productivity wasn't a big priority.

Performance is another reason why some people are waiting. GUIs do impose greater overhead than character based applications. GUIs can be as much as half as fast at displaying data although a more realistic figure would be only 10% or 20% degradation. The difference is that the data is usually more meaningful. Take a word processor for example. On a character based word processor, all text is usually the same in appearance even though it varies in fonts, size, and attributes on the printed page. So it takes a little longer to display a document for a GUI based word processor that shows all the fonts, sizes and attributes just as they would print, it most likely takes less time to create and properly edit the document with the GUI despite the added overhead. In other cases, GUIs outperform terminals simply by having more facilities available.

An order entry application might require the operator to look up a customer by name. With a GUI the user simply types a few letters of the customer name and then scrolls though a *list box* to find the correct customer. This same capability can be used for product lists, gl account, and anything else that requires choosing from a list. Other items that can be selected from one of only a few choices can be displayed as check boxes. If the choice must be unique, radio buttons are used. These simple devices eliminate mistakes and avoid the tedium of trying to work around the limitations of character based user interfaces. How many times have you stood at the airline counter while the agent ceaselessly banged away at the tab key trying to get the cursor to end up on the right field to make a simple change to your ticket. With a GUI they would have pointed to the correct choice and the line at the counter would have been shorter by one person minutes sooner.

Simple GUI Date Entry Form



What it takes to create a GUI application:

Probably the biggest concern facing anyone thinking about committing to a GUI is the task of writing their own *in house* application. How difficult is it? Will my people be able to hack it or do we need new talent? How long will it take? While I can't accurately answer these questions without knowing your staff, I can tell you this: writing GUI versions of our text editor SpeedEdit was the most difficult programming task our company has ever undertaken. I have heard the same comment from the people at Walker, Richer, and Quinn concerning their Windows version of Reflection. But at the same time, now that the development is over, I can honestly say that it was worth it. I say this not only because we are reaping the financial benefits of strong sales but also it has made our product, like Reflection for Windows and Tymlabs' Session, new, vital, state of the art and preeminent in the industry by being one of the first and the best.

While most of the people reading this paper won't be going into the business of selling software, creating an in-house application which is current and modern can rarely be faulted. I also wouldn't recommend to anyone to quit their job and write another text editor, terminal emulator, word processor or spreadsheet for windows since they take years to develop and the market is rapidly becoming flooded with very strong products. Your in house applications are another matter. You probably have already realized that you will eventually have to take the plunge and create more modern and up to date versions of your applications but are not quite convinced that now is the time. This is what it takes, should you decide your company is ready:

a) The first thing needed is a choice of environments. There are three, possibly four, strong contenders for the standard GUI which will gain the most wide spread acceptance throughout the industry.

   MS-Windows. At present MS-Windows is in use by over 3 million individuals, possibly double that number. By the end of 1992 that number will be anywhere from 6 million to 10 million. By far this is the most prevalent GUI today and in the foreseeable future. The only drawback is that it is only available for PCs. Rumor has it that Microsoft is working on a portable set of libraries which can be used on other systems such as Unix as well as proprietary operating systems such as MPE should the likes of HP care to implement them.

   Presentation Manager. This is the equivalent of Windows for OS/2, but alas, it is somewhat different and requires recoding programs and different libraries to implement. It was IBMs intent to implement Presentation Manager on all their systems starting with the AS400, but that isn't happening at all, or at least as quickly as hoped for. Those

companies that bet on Presentation Manager over MS-Windows lost lots of valuable time and market share as their competitors ran away with the Windows market, passing up the virtually non-existent OS/2 market. Since Microsoft has released a product called WLO (Windows Libraries for OS/2), Windows applications can be readily ported to Presentation Manager simply by relinking with a different set of libraries. PM seems to be a bad bet at present.

X-Windows. At present there are two flavors of X-Window applications, differentiated by their different look and feel, Motif and OpenWindows. While both of these *standards* are based on the same underlying X-Windows libraries, they allow the user to interact with the system in substantially different ways. It is my belief that Motif will become the standard over OpenWindows since Motif is currently in more wide spread use and that it more closely looks like both MS-Windows and Presentation Manager. While Motif has the support of Hewlett-Packard, SCO, and dozens of other vendors, the major proponent of OpenWindows is Sun Microsystems. I've used both and my preference is Motif, since OpenWindows is a large departure from the defacto standard MS-Windows, not to mention too heavy a reliance on the use of a mouse. A couple of other developments may give Motif a little more support. It has been rumored in the press that in DOS 6.0 Microsoft plans to integrate Windows and DOS as well as provide a built in interface to X-Windows. Since Motif looks most like MS-Windows, it would be the most reasonable to have running side by side with native Windows applications.

When all is said and done, I believe there will be three *standards* which we will have to live with, MS-Windows, Presentation Manager, and X-Windows Motif. Fortunately, it may be that some day there will be one library interface to all three of these systems and thus no need to reprogram for different operating platforms.

b. Next, you will need to procure a development *toolkit* for the platform of your choosing. The toolkits usually include the following:

Libraries The interface to the windowing system called by your applications. Applications call hundreds, possibly as many as a thousand, different subroutines (procedures) which handle the jobs of redisplaying windows, processing keystrokes, tracking the mouse, etc.

Resource Editor Resources are things like dialog boxes (like the one shown earlier), strings, menus, pictures, icons, and fonts. You need to be able to graphically create and manipulate these in order to make programming a

GUI reasonable. This tool basically allows you to create the objects which you use in your application.

Debugger Without a symbolic debugger that can handle your windows application the job of creating, testing and debugging an application as complex as a GUI would be impossible.

Misc. tools Most toolkits offer a variety of other tools to allow you to see how memory is being used, which applications are running, view message ques, and monitor other aspects of the GUI.

c) Language. You will need to choose a development language. For most low level jobs, the only choice is **C** or **C++**. For application type software, it is possible to write your code in COBOL but if so, make sure you have located a COBOL which knows about Windows. One PC based compiler claims to simplify the handling of MS-Windows considerably. Many 4th GLs also support most of the popular GUIs. With some of these, you don't even need the toolkit although its probably a good idea to have it around for those jobs that can't be handled any other way. Borlands C++ comes with its own Windows libraries to eliminate the need for the Microsoft Windows SDK.

d) Training. Unless you are blessed with several programmers who are already familiar with the GUI of choice, send several people to as much training as you can afford. The cost and lost work time will more than be made up for later on.

e) Design. You will most likely find that using a GUI on a PC and combining it with *client-server* technology is your best bet. This offloads the host in two ways. The host isn't responsible for any part of the GUI load and the front end portion of your application is completely off-line, allowing the host to do what it does best, process transactions. In the best scenario, the HP-3000 is nothing more than a data base and print server.

What you end up with is a far more complex application requiring more training and more time to develop (unless the 4th GL handles its job completely). The end result is an application which places a lighter load on the host while at the same time makes the users faster, more productive and more accurate, requiring far less training and support.

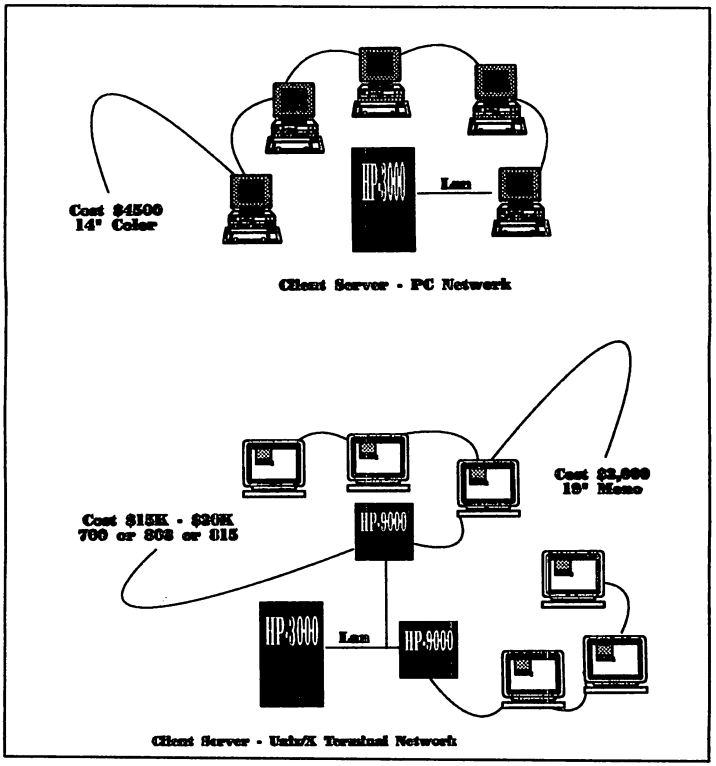
4th GLs

While I can't claim to be an expert on 4th GLs since I haven't used even one, I have been following their development with great interest. Currently, most of the major data base players have their own GUI oriented 4th GLs. Oracle, Ingress and Informix all have tools that promise to speed application creation and allow for *client-server* operation across dissimilar SQL based databases. These products bear investigating as they sound like they can cut the time and cost involved with developing a GUI based application dramatically.

Starting with MPE-XL 2.2, the Ingres development tools are shipped with the Allbase/SQL system which should greatly facilitate the implementation of a GUI with your data base applications. A fundamental part of this package is Windows/4GL, which is a Unix based utility to create client applications that have read/write access to ALLBASE/SQL on HP MPE-XL servers.

Many development tools are already on the market which greatly assist in the development process, and stand to make the chore of developing a GUI based application less threatening, not to mention less costly.

The final item which makes the time right for moving to a GUI is the robust nature of the many network offerings for the HP systems. GUIs require a fair amount of horsepower and that horsepower can come in the form of a PC running MS Windows or a workstation running Unix and X-Windows or an X Terminal. These solutions all require a network and a high degree of connectivity to speed not only your application, but also the development of that application. For running a client server based application which uses X-Windows you may end up with an arrangement with a central HP-3000 acting as a general purpose server, and one or more Unix based systems driving several X Terminals. Alternatively, you might have just an HP-3000 with numerous PCs networked into it. The days of the terminal are numbered and with that the days of point to point connections are also numbered providing a need for high speed networks and ultra smart workstations to take over and enhance their role. The GUI is the medium which provides that ultra smart behavior and *which flavor* GUI, at this time, is a subject for debate and also a matter of your own comfort level.



Envision the day when a 3 foot by 2 foot portion of your desktop is a flat panel display with dozens of windows displaying all sorts of information. Things like your daily schedule, a to do list, the company profit and loss statement, a list of calls to be made, the current production schedule, and various reports and periodicals to which you have subscribed, as well as todays newpaper. Each of these items looks as crisp and clean as their printed counterparts, complete with color graphics and pictures. You even have a window on your screen with the ongoing CNN newscast and another full video window attached to a camera watching over the production shop floor. Now imagine, if you will, your clunky old fashioned, slow, non-dynamic, heart of your business application which hasn't been enhanced in nearly a decade, sitting right in the middle of the whole screen crying out for all the world to hear, someone missed the boat! The time is right, seize the opportunity!

Paper # 4102

The Anatomy of a Successful LAN Installation

Neil R. Brooks
International Foundation of Employee Benefit Plans
18700 West Bluemound Road
Brookfield, Wisconsin 53008-0069
(414) 786-6700

My organization recently completed a year-long installation of a 100 node local area network. In this paper I will evaluate the implementation of this massive project and offer evidence as to why we think it was a success. Every project of this magnitude has its own life cycle, from the realization that you need to do something to the installation of the last workstation. This paper will examine the many factors that must be considered, including what LAN topology to use, hardware platform, software used, connectivity with your HP-3000, systems migration, development platforms, training and many others. I will also discuss the need to construct a realistic implementation schedule while maintaining day-to-day service for your user base. This paper also examines the need to arm yourself with as much expert knowledge as possible in today's ever-changing computing environment. For example, how do you apply the knowledge gained from sessions attended at INTEREX conferences, consultants and other users. It was both interesting and disturbing to discover that Hewlett-Packard is not as committed as they say to working in a multi-vendor environment. We also had the opportunity to discover several hardware problems for them as well. I will also offer some tips how to manage organizational change and maintain your sanity at the same time.

I work for a non-profit educational foundation dedicated to the employee benefits industry. As such we are very similar to INTEREX in our mission and structure. Our user departments are comprised of membership, registrations, educational programs and development, audio visual, graphic arts, printing, research and executive, which must deal with the board of directors and various committees comprised of our membership base. The Foundation was using an HP-3000/52 minicomputer with 35 terminals and an NBI word processing system with 20 terminals. At the time, we had a potential user base of 100 employees out of a total of 125.

Prior to my accepting the position of MIS Director at the Foundation in March of 1989, I had the opportunity to review an MIS audit report prepared by Andersen Consulting. The report detailed many problems, such as lax security, improper system backup procedures, a lack of hardware, poorly functioning systems, an eleven man-year project backlog and many others. Several months after I came on board, I began to realize that the problems were much worse than detailed in the audit. For example, because none of the information systems were integrated, there was much redundant data being entered and maintained. Average system response time on the HP-3000 was ten seconds, and it was not unusual to experience response times of 20 seconds or more. The systems were poorly written in Powerhouse, and contained many modules that performed serial reads through tens of thousands of records because of poorly designed data bases. We also faced a severe lack of hardware, with

only one third of the user base having access to an HP or NBI terminal. I also conducted extensive interviews with department directors, and the MIS department staff spent many hours observing the interaction between departments and the information systems they used. This information was then used to prepare an information systems needs analysis for each department.

In addition to my realizing that solutions were urgently needed for these problems, I also had a mandate from our chief executive officer and the board of directors because of their awareness of the problems detailed in the audit report. We were also fortunate in that the user base realized that their MIS needs were not being met. Given their usual reluctance to change, I took this as a sign that something really had to be done. I set a goal to have a final proposal ready for presentation to the board of directors by August of that year.

After determining the information system needs of the Foundation, we defined the goals that we needed to accomplish.

LAN IMPLEMENTATION GOALS

- **To select a standard for hardware to be used.**
- **To install a network structure which would allow for distributed processing to take place at each workstation, as well as having the ability to obtain information from the HP-3000.**
- **To install microcomputers in all the departments for personnel to use in day-to-day operations and greatly reduce the amount of paper flow between departments.**
- **Provide data security for all users, departments and for the Foundation's information systems.**
- **To select standards for software to be used.**
- **To share the information between all departments via electronic data exchange (EDE) with regard to current software applications being used.**

In order to implement these goals we decided to arm ourselves with as much knowledge as possible, so that we could develop a list of solutions. I felt it was important to remain as objective as possible, and not develop a sense of ownership for any particular solution, thus eliminating those that could be more effective. I began by contacting various consultants, all capable of offering various hardware and software platforms. In addition, I also worked with the local Hewlett-Packard office.

In choosing an effective consultant, I looked for several key factors.

KEY FACTORS IN CHOOSING A CONSULTANT

- **Years of experience in the business.**
- **Areas of expertise.**
- **Information provided by customer references.**
- **Hardware/software vendors they represent.**
- **Willingness to provide solutions that meet your needs instead of theirs.**
- **Level of knowledge as to where MIS is headed, instead of where it has been.**
- **Commission/pricing structures (do they favor one type of solution over another because it means more money in their pockets).**
- **Technical support capabilities.**
- **Cost of goods and services.**

Many of the consultants I contacted appeared to be prejudiced and narrow-minded in their approach to our problems. They could only offer solutions based upon what they had done in the past. IBM seemed to fit into this category the best, where they proposed installing an AS/400 and having a person in Colorado convert all of our existing software so that it could run on this platform. It was not a viable solution to convert bad code so that it could run on a bad machine. I also found that Hewlett-Packard was just as ineffective, because their commission structures favor the sale of an HP-3000 over anything in the PC arena, resulting in their sales force proposing solutions based upon their needs and not yours. I developed a consultant fact sheet, which allowed me to rate each vendor using the same criteria. When all was said and done, I chose three vendors as finalists, with the intent of having each submit a proposal in a competitive bidding atmosphere.

It is also very important that you do not put all of your eggs in the consultant basket. Therefore, we attended seminars, conferences, user group meetings and other forums to try to increase our level of knowledge with the intent of being fully aware of what

solutions were available at that time, and would be available in the future. I also contacted other organizations in the area that had recently installed local area networks or were exploring the possibility of doing so. This type of networking allowed me to gain insight into what types of solutions were effective, and which ones were not. It always helps to be able to learn from someone else's successes and failures. It was also helpful to perform abstract searches within our library at the Foundation, and at the University of Wisconsin library. I gathered articles pertaining to local area networks and new developments in the minicomputer arena. The knowledge we gained on our own helped us to better judge the effectiveness of various solutions presented by the consultants, thus allowing us to weed out the fact from the fiction.

The next step in the process was to begin brainstorming possible solutions, based upon the needs analysis prepared for each department and the goals we wanted to achieve. Do not eliminate anything because of preconceived notions. If anything, this process should help to solidify your final solution through the process of identifying the weakness of the alternatives. Also, all or part of one solution may be able to be combined with another to provide a more viable third solution. This process should result in the development of a solution that is based upon several factors.

FACTORS TO CONSIDER IN CHOOSING A SOLUTION

- **Will it allow you to meet your stated goals?**
- **What will it cost for installation and maintenance?**
- **Physical plant - does your building present physical obstacles?**
- **User base - level of knowledge, willingness to accept change.**
- **MIS department staffing and their level of knowledge.**
- **Available and emerging technology.**
- **Amount of training needed, and other special needs.**
- **Maintaining service to user departments during installation.**
- **Total disk and memory capacity required.**

- **Security concerns.**

- **Ability for future expansion.**

The solution we chose was intended to meet and exceed the objectives of our plan. It consisted of a Lattisnet (Ethernet) LAN, consisting of 12 Synoptics departmental concentrators, 2 workgroup concentrators and one premises concentrator. This LAN topology would utilize fiber optic cable as the backbone and unshielded twisted pair cable for the connections between concentrators and workstations. The file server would run under the Novell Netware/386 operating system. The hardware configuration consisted of one HP Vectra RS/25C microcomputer for the file server, and 100 HP Vectra 286/12 microcomputers for the workstations. The file server would contain 1.2 gb of mirrored disk capacity and 16mb of main memory. Each workstation would contain 640K of main memory, with the ability to expand it where necessary. Each workstation would be diskless, thus preventing the introduction of software or other data into the LAN without the control of the MIS department. Each user would have the capability to share, process, view and manipulate their own files on the network, and each would have access to inter-office communications and appointment scheduling. If granted access by the network supervisor, individual users would also have access to other computer systems located within the network structure. Printing needs would be met by placing an HP LaserJet printer in each department, providing each user with the capability to send their output to any printer attached to the LAN. Through the use of gateway technology, the HP-3000 would be available on the network to provide users with access to the information systems located on that platform. Remote dial-in access by network users located outside of the physical boundaries of this network would be provided through a dedicated 80386 microcomputer and communications software, and would provide access to 4 remote users simultaneously. A tape backup system would be installed to provide the capability to perform a full system backup on a daily basis, including the hard drives located on microcomputers in the MIS department. We also included 6 portable NEC personal computers to be used by employees at home and offsite at the various educational programs, seminars and conferences conducted by the Foundation.

The standard software platforms chosen included WordPerfect for word processing, WordPerfect Office for electronic mail and scheduling, LOTUS-123 for spreadsheets, Aldus PageMaker for desktop publishing, and DataEase for intradepartmental database development. The platform for system development chosen was Visual Cobol from MBP software, which combines the attributes of a 4GL with ANSI COBOL/85 and includes a screen generator, program editor, prototyping tools and program debugger. The database management system chosen was the BTRIEVE relational database from Novell. The plan also included miscellaneous software intended for use within one or two departments, such as Harvard Graphics for the Audio Visual department, and SPSS/PC+ for statistical processing in the Research Department.

The next step was to obtain comparative pricing for the plan from the three vendors we had chosen to work with. This was done by submitting the final hardware and

software specifications to each, so that an apples to apples comparison could be made. We asked each vendor if they offered any discounts, such as those based upon quantities purchased, or special programs offered to educational or non-profit organizations. If you have done your homework, you should feel comfortable with what they present to you. Now is not the time to question if one or another vendor is presenting a low bid just to get the business without providing the value and service needed to implement a project of this magnitude. We then chose one vendor to do business with. As it turns out, all three were very competitive in their pricing, and the final decision was primarily based upon our comfort level, or how well we thought we could work with them.

We then proceeded to developing an implementation schedule. The most important factor to consider here is not to overextend yourself. It is very easy to attempt to solve the problem all at once, especially if the problem you are attempting to solve is very large and visible to the organization. You will be under pressure to "get the thing in" from a variety of sources, including department managers, users, your boss and the board of directors.

FACTORS TO CONSIDER IN DEVELOPING AN IMPLEMENTATION SCHEDULE

- **Realistic capabilities of the MIS department.**
- **Capabilities of the vendor.**
- **Availability of hardware and software.**
- **Ability to take advantage of emerging technology.**
- **Price increases/decreases and quantity discounts.**
- **Need to maintain day-to-day service to the user departments.**

Based upon the above factors, we decided to install the LAN by phase. It was then necessary to define the unit of measure for each phase and what it would entail. We decided to install one department at a time over the course of one year. Each phase would consist of four to five weeks, depending upon the size of the department. We also decided to install the larger departments first, thus bringing the benefits of our solution to the largest number of users in the shortest period of time. Another advantage of this type of implementation schedule is to provide positive exposure to the users in other departments who may be reluctant to accept the pending change. If they can see the advantages and hear about them from the users in the installed departments, it will make your job that much easier when you are ready to install their department.

It is also very important to determine how you will train your user base in the use of the hardware and software they will be using. We analyzed the pros and cons of training the users ourselves or utilizing an outside training service. We believed that our training needs would be better met by training the users ourselves, for we could better design the training to cover the needs of a specific department, the needs of specific users, provide the flexibility of scheduling, minimize the disruption to work schedules and the fact that we knew the personalities involved. Keep in mind that there is a correlation between a user's level of knowledge and their effective use of the system. However, you may not have the internal resources necessary to do the training in-house.

The final step was to prepare a proposal for presentation to the executive committee and board of directors. The important factor to keep in mind here is to include all of the details. Do not assume that your audience will not understand what is being presented to them. In today's world of computer literacy, you may be making a big mistake with this assumption. If you provide them with all of the detail, they will know that you did your homework, which should increase their comfort level with the proposal. If they do not understand various components they will ask questions, thus providing you with the opportunity to provide greater detail during your presentation. The proposal should also contain data that fully details the cost of the entire system. This was presented in the form of a detailed list of all of the components of the LAN. We also presented the implementation schedule in the form a timeline, detailing the phase identification, department involved and the length of time required for installation. The entire proposal contained over 80 pages of information.

COMPONENTS OF THE LAN PROPOSAL

- **Statement of objectives**
- **Overview of the proposed system**
- **Equipment and product descriptions**
- **Description of service and support**
- **Implementation schedule**
- **Cost for each phase**
- **Outline of training methods**
- **Total cost of the proposal**

It is also very important to present realistic costs. Build in room for price increases, even if you do not anticipate any, and allow yourself some breathing room for the

items you may have forgotten or overlooked. You cannot possibly include everything at the proposal stage. The many unknowns will come back to haunt you if you do not acknowledge that they indeed exist.

When the final proposal was completed we had Andersen Consulting review it for feasibility and to obtain suggestions for improvement. I also obtained input from several MIS directors I knew. Having an unbiased person or organization review your proposal will provide you with an idea as to how the board of directors will react to it. The questions the reviewer asks will also help you to prepare for the questions the board will ask as well. This process should help to clarify items or fill in any holes in your proposal if they exist.

We were then ready to present the proposal to the executive committee and our board of directors. Surprisingly, this went much easier than anticipated. Most of the questions they presented pertained to the need for every user to have a personal computer on their desk. Very little was discussed in the area of costs, since I was given a ceiling as to what we could spend months before, and the proposal came in under this amount. Upon receiving their approval, we then began the long and exciting process of implementation.

According to Michael Beer, Professor of Business Administration at the Harvard Graduate School of Business, there are 5 key components to affect successful change in an organization.

KEY COMPONENTS THAT AFFECT CHANGE IN AN ORGANIZATION

- **Key managers must be dissatisfied.**
- **The top manager must be committed to the change.**
- **Slack resources must exist.**
- **Political support must exist.**
- **Change resources must match the size and kind of change.**

Fortunately, these 5 conditions existed at the Foundation as we were ready to embark on our project. The board, CEO and department directors were dissatisfied with the information systems. As a result they were fully committed to the project, for they felt that the new system could only be an improvement. Slack resources existed in the form of budget dollars that were pre-allocated to the project and then approved by the board of directors. Because we involved the user departments from the very beginning through the process of obtaining their input and observing their use of the existing systems, we had their political support. Finally, we believed that we had the

resources in the MIS department, our consultant and the user base to successfully implement the project. Even though these conditions existed, we still braced ourselves for a certain amount of resistance. We countered this possibility through effective communication with the users and department managers.

We started the LAN implementation by calling a Foundation-wide meeting of all employees to explain in detail just what we would be doing over the course of the next 12 months. Because there was much speculation as to what MIS was up to, and many rumors going around, this meeting served to clear up any misconceptions. This was followed with departmental meetings between the users and their assigned programmer/analyst, providing detail as to what would happen within their department. Our goal was to educate, inform and make the department feel involved in the installation process, thus reinforcing their sense of ownership to the project. Since each department is assigned a specific programmer/analyst, they knew exactly who to turn to with questions about the project.

Prior to the installation of each phase the MIS department met as a group and defined all of the tasks that needed to be accomplished and then assigned responsibility for each item. This was done at our weekly staff meeting, and this forum was also used to review the progress of each phase, any problems that occurred and the items that were a success. In other words, be aware of what works and what doesn't as you go. This allowed us to be proactive rather than reactive as we proceeded with the installation. It also points out the value in installing the project in phases, which gives you the opportunity to correct problems before they become very visible, or grow to an unmanageable magnitude. Each phase was the responsibility of the programmer/analyst assigned to the department. Because this person worked with the department on a regular basis, they were familiar with their day-to-day operations, any special needs and the personalities involved.

The first phase consisted of installing clean electrical outlets for the concentrators. It then proceeded to installation of the premises cabling and concentrators, installation of the file server, UPS system, gateway server for the HP-3000, workstations in the MIS department and all related software. The intent was to get the LAN fully operational in the first phase, thus allowing the MIS department to begin the process of learning the Novell Netware/386 operating system, testing of all software, the functionality of the workstations and each installed node of the network. We did not want to begin installing any given department without knowing that the hardware or software would function according to specifications. Surprisingly, the cabling of the building went very smoothly and took only 4 days to complete. It also had the effect of generating excitement among the users, for it was the first visible sign that something was being done. After completion of the cabling we tested each jack to make sure that each component of the data transmission system was operational. We discovered two jacks to be inoperable through this process and had the problem corrected while the cabler was still onsite. Upon completion of the cabling we created a set of maps, detailing the location each LAN jack and all of the concentrators. The maps also indicate which concentrator each jack is connected to. This can prove invaluable when diagnosing a problem or in determining what your expansion capabilities are.

The first major problem arose during the installation of phase 1. This involved the gateway between the LAN and the HP-3000. We should have known this would be a problem since Hewlett-Packard was not much help when we were trying to determine the configuration of the gateway during the process of putting the proposal together. For example, we were at first told that we could not put in a gateway with an HP-3000/52, then we were told we could. The technical support offered by our local HP office was less than desirable. When it came time to install the gateway hardware and software we encountered one problem after another. First of all, they sent a customer engineer who was inexperienced in gateway configurations. What should have taken several hours to install took several days. During this process, there was much finger pointing between Hewlett-Packard and Wollongong, Inc., the vendor for the TCP/IP software used. The one thing we learned at this point was that HP is not at all committed to their marketing slogans of working in a multi-vendor environment and their commitment to adhering to standards. For example, the LANIC card designed for use in the HP-3000/52 is not IEEE 802.3 standard, which is what HP had told us. After a week of hard work, and through figuring out the problem ourselves, we were able to get the gateway operational. It was a matter of getting both systems to communicate with each other. Fortunately, the other components of phase 1 went in without any problems. However, because of this problem we spent a considerable amount of time insuring that all of the software was fully tested in a LAN environment. Our goal was to have the LAN fully operational and functional prior to proceeding to the installation of the next phase.

Security structure and procedures were defined and implemented during phase 1. An MIS policy was incorporated into the Foundation's regular policy manual which formally defined and enforced the security policy for the user base. The directory structure was established so that each department had its own directory. Subdirectories were created for each user. A public subdirectory was also created for shared documents and data. Each user was assigned a specific login identification based upon their name. Passwords are required and have to be changed each month. It is against the MIS policy to access any file in a private subdirectory, including electronic mail. It is also against policy to login under another persons identification and password. All files reside on the file server in a secure environment. A complete system backup tape cartridge is kept offsite at all times.

The security structure also includes automatic dial-back for all incoming modem access to the LAN through the communications server. All file transfer activity is prohibited accept for word processing documents. All directories are scanned for viruses on a daily basis. All database additions and modifications are also stamped with the user's identification name. As a result of the policy and the defined structure we have been able to maintain data integrity and user confidence in the system.

After the completion of phase 1 I created a series of worksheets in LOTUS for each phase of the project. I used these to track the equipment ordered, the amount budgeted for each item, the actual expense and the variance. This allowed me to manage the total dollars allocated to each phase and the entire project. It also helped in tracking what equipment and software had been received along with the items still on backorder. I also used LOTUS to perform "what if" analysis for substitutions of

hardware and to track equipment warranties and maintenance costs.

The installation of phase 2 involved the Research department, which included 5 users. As I stated earlier, we designed the implementation schedule to include the largest departments in the earliest phases. However, we felt it was prudent to install one of the smaller departments first, using them as a test of our abilities and capabilities. If any problems did occur, they would not involve a large number of users, nor would it disrupt the operations of one of our larger departments, such as membership, which could have had an effect on the Foundation as a whole. It is important that the first department you install be done successfully, for word travels fast if something goes wrong, and you'll be spending most of your time with rumor control instead of on solving the problem. Again, it's the concept of managing the project proactively instead of reactively. Fortunately, everything went like clockwork. When the hardware arrived we unpacked it and set it up within the MIS department, to make sure that it was fully functional. This is very important, for several personal computers arrived from Hewlett-Packard dead on arrival. Again, installing dead equipment in a user department can have a detrimental affect on your user's perception of the project. We then proceeded to install the hardware on the agreed upon installation date. It was at this time we discovered that we did not purchase any power strips. Fortunately, we were able to hurry over to the local computer supply store and purchase the necessary units. That was the last time we let that happen to us. Again, it is impossible to think of everything, so be prepared for that sort of thing. The important thing is to act as though you have your act together when doing anything within a user department.

We had scheduled training in WordPerfect Office to follow immediately after the hardware installation. This is the electronic mail and appointment schedular software. It also serves as the shell or home menu for access to all of the software packages available, including access to the HP-3000. We felt it was important to train the users as soon as possible, thus making the LAN functional for a department as soon as they had the hardware. The training they received was an abridged version, enough to get them started, and it included basics about the Novell operating system, electronic mail, and how to access other packages, such as WordPerfect and Lotus. Users were given the manuals for WordPerfect Office and WordPerfect as soon as their computers were installed. We scheduled full training in WordPerfect Office within several weeks of the installation in each department, along with training in WordPerfect because of the importance of word processing to the Foundation. These training sessions were scheduled over several days, and went into detail, covering each function of the respective packages. Users were grouped together in training by level of ability when possible. Each programmer/analyst, along with myself were assigned a specific software package to train. Each member of the MIS department received training from their fellow employees as well so that they could be available for assistance. This also provided the trainer with valuable feedback and a measure of the effectiveness of their training program. The intent was to have one person act as the expert for each specific application. However, it is important that each person in the MIS department be capable of providing assistance.

The training programs were developed with each department in mind. Sample

documents created in WordPerfect training were of the type used by the department being trained. For example, research questionnaires were used when the Research department was trained, since this is their primary use of word processing. The training programs also concentrated on those features the department would use most, however all of the functions in each package were covered fully. This approach to training has been a huge success. It should also be noted that our intent was to provide each user with word processing and electronic mail/scheduling even if they had never used it before. Many department directors were soon creating many of there own memos in less time than it took to dictate them to their secretaries, who then had to type them or enter them into the old NBI word processing system.

Training in the other packages, such as LOTUS and DataEase was given to those users that requested the software and where there was a demonstrable need for it as part of their job function. With DataEase for example, we assigned a database administrator in each department to act as the database manager. Our intent for this database management system was to provide the means for users to develop simple systems to manage data not related to the Foundation's integrated membership system. However, we did not want DataEase or LOTUS being used to develop redundant systems or spreadsheets within a department or between departments. This also would have defeated the purpose of distributed processing and shared access to files and data. Users were selected based upon their level of knowledge and expertise. This training was provided several months after the installation of a given department. It is important not to throw too much at the users in the very beginning. They will not retain as much knowledge from the training program and they will not have the opportunity to spend time using the package immediately following the training, which should be done to reinforce what they have just learned.

I would suggest putting a lot of thought into whatever training program you develop. This can make or break the success of the LAN. If users cannot use the tools they are given, they will soon wonder why they went through the bother and disruption of the install. It is also very important to get them trained as soon as possible in the basics, otherwise they will become very frustrated trying to navigate the system on their own, and the MIS staff will spend most of its time responding to isolated user questions. Training, along with thorough communication between the MIS department and the users is your most effective tool for managing the change a LAN installation will bring about. Be prepared to respond to any negative reactions as quickly as possible, especially when you are installing departments in phases. It only takes one user to go around the organization bad-mouthing a software package to have a negative affect on other users. We knew we were on the right path when we had departments asking if they could be installed earlier than scheduled.

We had built into the implementation schedule one week between each phase in order for us to sit down and review the phase just completed. It also provided for flexibility in case of late shipments or any other problems that might occur. The review process is very important, for it provided us the opportunity to assess the situation and determine what we could have done better, problems that could have been avoided or methods that could be improved. For example, following the installation of the Research department, we determined that we would add a physical

inspection of each department prior to them being installed. Using a portable computer, we retested each LAN jack and also determined exactly where each workstation would be placed along with the location of the printer. This information was recorded on a site inspection worksheet and was signed off by the department director as well. This process also acted as a final verification of the workstation and printer count for each department. We also developed a phase sign off form which was completed by the department director, indicating that the department was installed to their satisfaction. It also provided an area for comments where they could detail their likes, dislikes, need for further training and other miscellaneous items. In summary, it is important to know the status of the project at any given time, and to know where you are going.

The remaining phases were installed according to our schedule. At no time did we experience any delays that could have resulted in having to delay the installation of any department. We had also allocated extra time for phases that involved smaller departments, and took advantage of this to regroup, assess our progress and deal with any problems. As we completed each phase it became quite routine, with everyone knowing what had to be accomplished. Installing by phase also gave those users not yet installed the opportunity to see first-hand what was in store for their departments. This helped to alleviate the fear of the unknown and greatly contributed to the overall success of the project. One problem this situation did create was an increasing demand to shorten or alter the implementation schedule. This is something we never expected, and took it as a measure of our success. That isn't to say that we did not encounter any problems as we progressed. For example, we encountered a major problem with the display quality in Hewlett-Packard's monochrome VGA monitors, which were too dark in color mode. Because of this we received numerous complaints from the users. We also uncovered a problem with the compatibility of the ROM BIOS chips in the Vectra 286/12 personal computers being used in a diskless environment. As of yet, Hewlett-Packard has not solved either of these problems.

In defining the success of any project of this type it is often difficult to offer concrete evidence. However, I believe I can offer indications that our LAN is functioning to everyone's satisfaction. Software usage analysis indicates heavy utilization of electronic mail, scheduling, word processing and the systems that the MIS department has developed and converted from the HP-3000 to the LAN. Users who had never used the old NBI word processing system or the HP-3000 are now using the LAN and have come to depend upon it. We hear many users state "I don't know what I did without it". Communications between departments has improved greatly through the use of electronic mail. Critical information can be passed along to a user even when they are away from the office. Another indicator is new ways of thinking about old tasks and having the ability to create new ones. This has resulted in increasing the customer service level we can give to our membership base. For example, we can now provide online information to our members offsite at educational conferences, confirming their registrations, hotel reservations and other data. In addition, the LAN has experienced a 99.9% uptime rate since its installation. This speaks for the quality of Hewlett-Packard hardware and the Novell Netware operating system.

INDICATORS OF A SUCCESSFUL LAN INSTALLATION

- **99.9% uptime rate since the first phase of installation.**
- **Acceptance and use of the software packages provided as indicated by observation and usage analysis tools available in a Novell environment.**
- **An increase in creativity within departments, resulting in new approaches to job functions and problems.**
- **The entire project was installed on schedule and 9% under budget.**
- **Data remains secure and users have a high level of trust in the confidentiality of their data.**

Now that the LAN has been fully installed for approximately one year we have moved on to the task of converting the HP-3000 based systems to the LAN. We have employed many of the same techniques we used to implement the LAN in our approach to this massive conversion project. We anticipate that it will take two years to complete. One thing you should be prepared for when you reach this point following the install is the increase in expectations your users will have toward the LAN and the MIS department. This will result in further demands being placed on your resources. However this is more desirable than having them ask you to de-install the hardware. In conclusion, the keys to success for a project of this magnitude incorporate thorough planning, managing change through communication, expecting the unexpected and a little luck.

4103

THE BLACK HOLE OF PC INVESTMENT

James Call
The NPD Group
900 West Short Road
Port Washington, New York 11050

INTRODUCTION

Many companies are having second thoughts about the wisdom of their investments in PC's. After spending millions on PC's and related programming, U.S. office productivity seems stalled. Vendors answer with an ever upwardly spiraling line of advancements: XT...AT...286...386...486...etc. This paper addresses the pitfalls which, in retrospect, have contributed to disappointing PC results.

We discuss these issues in three sections as follows:

- Symptoms
- Underlying Problems and Opportunities
- What to Do About It

We include examples of both successful and unsuccessful implementations and provide a checklist summarizing possible improvement approaches.

I. SYMPTOMS

There is little disagreement that the U.S. is failing to keep up in productivity. The larger industries, automobile manufacturing and consumer electronics, are often cited, but they are examples of a wider problem. PC's have been installed by the millions to improve productivity, but symptoms are emerging which suggest their promise is not being fulfilled.

Few Successes Reported

In a review of some 600 articles in leading PC magazines, only 2 articles dealt with productivity improvement results relating to actual work accomplished. Looked at from another angle, in 70 citations gleaned from a literature search on the key words "labor productivity" and "capital productivity", PC's were mentioned only once, but negatively as "Despite computers, faxes, etc...productivity lags."[1] A cover story in Fortune Magazine lamented the "Puny Payoff from Office Computers".[2] They reported that "so far productivity has grown more slowly in the computer age than it did before computers came into wide use."

Misdirected Productivity Focus

When the word "productivity" does appear in PC literature (usually in PC hardware or software ads) it tends to relate to productivity in doing things with the PC itself, not to the actual work of the company as a whole. For example, a coprocessor makes the *PC* run faster, a spreadsheet add-in makes you more productive *manipulating the spreadsheet*.

Said one user in the Fortune article cited, "If people are doing the wrong things when you automate, you get them to do the wrong things faster."

II. UNDERLYING PROBLEMS AND IMPROVEMENT OPPORTUNITIES

The symptoms just discussed underlie a number of problems which have plagued PC implementations.

PC's Implemented Merely to Speed Up Existing Procedures

PC's may be implemented to merely speed up existing procedures without regard to whether the basic procedure was outmoded or even needed at all. This first wave of "automation" then gives a false sense of security and establishes a stake in a suboptimal solution, discouraging further attention to the real opportunity.

Word processing, spreadsheet software and numerous PC's have replaced IBM Selectric typewriters without streamlining the existing document flows. The new found revision capability also then stimulated trivial rewrites and fine tuning, eating into the little time that was saved.

This inappropriate use of PC's has been called using the PC only as an "electric pencil".

PC Focussed on Administrative Processes as Opposed to the Actual Work Itself

An advertisement for a PC program which is designed to optimize any worksheet cell by backsolving on a worksheet variable promises "Achieve goals, maximize profits, and minimize costs with this one program." Were it this simple.

This is an example of directing undo attention to a process as opposed to the work of the firm. It at best recapitulates what is already known about a real situation. At worst it legitimizes a useless management exercise quite unconnected to the opportunities one would find in the reality of the factory floor. Zaleznick in his provocative book, The Managerial Mystique[3] describes the failure of American management as the substitution of process for substance- "... programs and procedures as a substitute for direct engagement in work."

PC's Implemented in Spite of Superior Alternatives

A spreadsheet program nearly 1000 lines long, replete with macro's and data entry screens was laboriously constructed to perform a rotation of survey questions. (Such rotation is an important requirement to prevent "order bias" in market research surveys. It keeps the same questions from always being first or last.) At first glance the spreadsheet appeared to automate something, but it took 2 hours to set up the inputs for each new project and to run the program. It was subsequently discovered that a simple row and column matrix of predetermined numbers listed on a page of paper could serve the same purpose. To use the latter method, one simply listed the question number down the left, then read across to see the ordered location. Not only did the "manual" table-based approach not require a PC, but it was over 1000 times faster.

PC's Implemented Redundantly

A number of PC implementations provide capacity which is already available on a corporate mainframe. Electronic mail stands out as an example. If people are already connected to a port on the mainframe, why reinvent the wheel. Moreover the end result, on today's PC's, would likely fall short of the file sharing ability, security and support we take for granted in the mainframe environment.

An expensive solution to these comparative limitations, when they arise midway in a PC implementation, may be more PC disk, more memory, more PC's and a LAN. The investment in PC redundancy, played out to its fullest extent sometimes results in PC equipment and support costs which rivals that of the mainframe data center. In essence, the users are trying to build a new mainframe environment out of PC building blocks, hoping to achieve similar functionality.

Companies may not be able to wisely budget or benefit from huge investments when each expenditure, considered alone, is far below an established capital expenditure threshold.

Limited Capability Sneaks Up On You

Things seem so good in the initial stages of a PC implementation. Prototype data cases appear to demonstrate feasibility, but as implementation proceeds, larger files become a reality. Unforeseen tradeoffs of functionality are then mandated by the limits of the technology. To an extent, we are spoiled by mainframes and thus gullible to the initial ease of use of the PC.

Additional examples of PC constraints which may not be initially obvious include:

1. Memory constraint on a spreadsheet, limiting the number of lines.

2. Twenty five lines of only 80 characters each on a screen.

3. Lack of an effective approach for system and data backup and security of sensitive data bases.

Cumbersome Data Entry

Input and output from the PC often presents another snag to overall productivity. Untold hours of labor are spent retyping mainframe generated report data back into personal spreadsheets, an ironic manual intervention. Moreover, while the keyboard is still a major input device, users' typing skills are often uneven at best.

Cheating the Cost Accounting System

The typical end user of the PC, with all tne best intentions, spends numerous hours developing personal applications, entering data and configuring applications software. Much of this time is a substitute for what would have been a more formal MIS project. Putting aside the issue of whether the PC users' work results in a long term corporate asset, the resources put into it bypass the corporate charge back system. Even minimally successful results then seem acceptable, since they appear to be "free". The real kicker comes in when the author of the "PC System" is promoted, transfers or leaves, and something goes wrong or a change needs to be made.

Another important cost accounting issue arises in companies where a charge back system is used for mainframe computer processing. There may be real business reasons to offload mainframe work to PC's but if the associated PC costs are not reflected, users will not be able to make a correct decision on where to best run an application. To an individual PC user it may appear costs have been reduced when a PC application bypasses the corporate cost system. However, in fact, corporate costs may be higher overall.

III. WHAT TO DO ABOUT IT

Having discussed the symptoms and underlying problems which have worked to limit PC productivity, let us now turn our attention to what can be done about it. In any problem we can profit by seeing in it an opportunity for improvement. In that sense, problems are to be appreciated, not lamented, for the improvement direction which they bring to us.

Re-engineer Operations

The centerpiece of improving productivity is re-engineering operations. Re-engineering is a subject worthy of a whole presentation (or even a life's work) in and of itself. For the moment we will limit our scope to a summary of four key aspects of re-engineering work activities as a prerequisite to PC productivity:

- Eliminate
- Streamline
- Automate
- Minimize Hand offs

Eliminate

First look to eliminate existing steps, processes or whole areas of activity. For example, a review of the distribution list for a cost accounting report revealed that only half the 12 recipients even used it. Of the 6 actual users, none of them used all of it; each needed only the section reporting on their own department.

As one author suggested in a recent Harvard Business Review article on re-engineering work, "Don't automate, obliterate."[4]

Streamline

Once you have eliminated needless or redundant activities, look to streamlining what is left. Keep in mind, during the streamlining effort, that PC's or other technology may or may not play a part.

An example of streamlining would be to consolidate information fields, originally on numerous forms, into a single form. This might set the stage for a computer based data file or e-mail. Again though, the streamlined manual approach might be fine.

Automate

Once activities are streamlined, try to automate those tasks which occur frequently or which, if infrequent, are time consuming. It goes without saying that it is a waste of time to automate something that takes little time manually, although such needless automation is surprisingly common. Don't lose sight of the fact that some jobs are actually *better* done manually.

Minimize Hand Offs

In re-engineering, look closely at the pre- and post-steps of any task and see whether you can combine steps, even if they occur in another department.

For example, a computer department issued a daily production report in a spreadsheet format. This was sent by e-mail (HPDESK) to key users. It was discovered that one user department was adding some critical information of their own and reissuing their version of a similar report. By simply enhancing the original report in the computer room, an entire step in another department was eliminated.

A key concept is that once a given item is handled, try to do all the tasks which focus on that item. This also will help you avoid over specialization.

Keep an Open Mind

In all the above, strive to keep an open mind. Challenge preconceptions and consider striking out in new and productive directions. For example, faced with a labor shortage in data entry, specially programmed PC's were installed in over 60 homes for "Work at Home" employees. This tapped a new labor market and avoided the need for expanding the office space as well. The somewhat non-traditional use of PC's would not normally have been thought of without stretching our minds to remove constraints.

Don't set goals too low. Productivity improvements of 5, 10, even 1000 times existing rates are not uncommon in re-engineering situations.

Bringing It All Together

By re-engineering the work before you implement PC's you will stand a much better chance of realizing real productivity gains. With aggressive re-engineering many projects will turn out to deliver significant productivity improvement and not even need the PC after all.

In cases where you do need a PC, the investment in capital and operating costs will have a much better chance of paying off.

PC Productivity Checklist

The checklist on the following page outlines the key suggestions of this presentation and suggests approaches which can maximize your chances of significant productivity improvements.

PC PRODUCTIVITY CHECKLIST

- Try to go beyond merely speeding up existing systems.
- Re-engineer operations first.
- Eliminate needless steps.
- Streamline activities and information flows.
- Automate time-consuming repetitive tasks.
- Minimize hand offs.
- Consider alternatives.
- Acknowledge all costs and benefits.
- Anticipate potential technology limits such as memory, disk and processing speed.
- Keep an open mind.

REFERENCES

(1) New York Public Library, Literature search 5/91.

(2) Fortune cover story by William Bower, May 29, 1986.

(3)Abraham Zaleznick, The Managerial Mystique (New York: Harper & Row, 1989)

(4)Michael Hammer, "Reengineering Work. . .," Harvard Business Review, July-August 1990, 104-112.

TITLE: Cooperative Processing Using Windows 3.0 and Networking

AUTHOR: Doug Walker
Walker, Richer & Quinn
2815 Eastlake Avenue East
Seattle, WA 98102
(206) 324-0350

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING.

PAPER NO. 4104