

INFORMATION
CROSSROADS
OF THE 80s

SEPTEMBER
8 - 13, 1985

■■■■■■■■■■ INTEREX

HOSTED BY BALTIMORE/WASHINGTON RUG



PROCEEDINGS
HP 3000/SERIES 100

————— VOLUME I —————

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

INTEREX

the International Association of
Hewlett-Packard Computer Users



Proceedings of the 1985 CONFERENCE at Washington, D.C.

Hosted by the
Baltimore-Washington Regional
Users Group

Papers for the
HP3000
and
Series 100

VOLUME I
PAPER 3001-3042

Sam Inks, Editor

VOLUME I

Introduction

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

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

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

REVIEW COMMITTEE

Nick Demos
Sam Inks
Suzanne Perez
Joan Peters
Kevin Rhea
Chris Seiger

SPECIAL PEOPLE

Dean Gabersek
Millie Gabersek
John Grether
Dorothy Inks
Lee Mauck
Mary Moorer
Nancy Murray
Ron Smirlock

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

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

Index by Author		Vol.
Beasley, Dave	Hewlett-Packard	
How Dispatching Queues Really Work.....	3065	11
Bircher, Carolyn	Hewlett-Packard	
Writing Efficient Programs in Fortran 77.....	3076	11
Boles, Sam	Hewlett-Packard	
Unix Thru The Eyes of MPE.....	3083	11
Boles, Sam	Hewlett-Packard	
A Blend of HP3000/HP9000 For Computer Graphics.....	3066	11
Bowers, Keith & Beauchemin, Denys	Northern Telcom	
Things That Go Bump in The HP3000.....	3018	7
Boyd, Larry	Dallas Times Herald	
The Segmenter.....	3005	1
Butler, Stephen M.	Weyerhaeuser	
Dictionary/3000--Extended Tour.....	3006	1
Carroll, Bryan	Hewlett-Packard	
MPE Disc Caching.....	3068	11
Casteel, Michael	Computing Capabilities Corp	
Anatomy of a True Distributed Processing Application.....	3029	1
Chang, Wanyen	Longs Drugs	
The Sorted File Access Method.....	3036	1
Clark, Brice	Hewlett-Packard	
Positioning Local Area Networks.....	3092	11
Clemons, Brett	Consultant	
Using Intrinsic in COBOL Programs.....	3027	1
Clifton, Roy	Hewlett-Packard	
North American Response Center.....	3054	11
Cornford, M. G.	Northrop Corporation	
There's Got To Be A Pony Here, Somewhere.....	3024	1
Depp, James A.	UPTIME	
Recovery by Design.....	3053	11
Duncombe, Brian	Carolian Systems Inc.	
Performance Self Analysis.....	3025	1
East, Ellie	Media General	
Training: The Key To Success With Personal Computers.....	3011	1
East, Ellie	Media General	
Information Center: Implementation Using HP3000 and HP150.....	3026	1
Engberg, Tony	Hewlett-Packard	
Response Time: Speeding Up the Man/Machine Interface.....	3060	11
Fisher, Eric S.	Wellington Management Co.	
You Said You Have a Bunch of Micros Linked To Your HP3000? Great!! Now What?.....	3035	1
Floyd, Terry H.	ASK Computer Systems	
CIM Is Not A Software Package or a Magic Wire.....	3010	1
Fochtman, Jerry	Exxon Chemical Americas	
Emulating A Real Time, Multi-Tasking Application System on the HP3000.....	3038	1

Franklin, Bill	Hewlett-Packard		
	Software Technology for the 80's (Understanding		
	Key Current and Future Technologies).....	3009	
Gerstenhaber, Peter	CMS Ltd.		
	Cooperating Processes in an Information Network....	3051	
Grim, Jelle	Holland House		
	The Twilight Zone, Between MPE Capabilities.....	3045	
Gross, Gail	Hewlett-Packard		
	Training and Supporting Office Systems Users.....	3019	
Hirsh, Scott	RCM		
	Change Management: An Operations Perspective.....	3084	
Hoeft, Mark L.	Hewlett-Packard		
	Developing Cost Effective Utilities and		
	Applications Using Business Basic/3000.....	3067	
Holt, Wayne	Union College		
	Communicating in a Mixed Vendor Environment.....	3031	
Idema, Tom	Westinghouse Furniture Systems		
	The Role of The System Manager.....	3071	
Isloor, Srekaanth	Cognos		
	The Ultimate Challenge in Application Design:		
	Managing Data Integration.....	3074	
Kaminski, Thomas J.	Singer		
	Migrating Information Between HP3000 Data Bases,		
	Electronic Spreadsheets and Microcomputer		
	Data Bases.....	3042	
Kane, Peter	Hewlett-Packard		
	TurboIMAGE Run Time Options.....	3039	
Karlin, Robert	Consultant		
	Auditability: or What's a Nice Byte Like You		
	Doing in a Base Like This?.....	3061	
Kopecky, Jerry	Illinois Criminal Justice Authority		
	Operational Considerations for Police Networks.....	3077	
Korb, John P.	Innovative Software Solutions		
	Store-and-Forward Data Transmission in a		
	Multi-System Network.....	3046	
Larson, Orland	Hewlett-Packard		
	Application Prototyping: A Proven Approach to		
	Information Systems Design and Development.....	3081	
Lawson, Roger	Proactive Systems		
	The Use of IMAGE Transaction Logging in a		
	Multi Data Base, Multi Machine Configuration to		
	Achieve A Non-Stop, Fault Tolerant HP3000 System....	3023	
Lewis, Donn	Allegheny Beverage Corp.		
	Building Your Own X.25 Data Network.....	3040	
Mattson, Robert R.	WIDCO		
	Why Software Projects Don't Quite Succeed.....	3034	
McDermott, James T.	Consultant		
	Rational Structuring Techniques for COBOLII/3000		
	Maintainability.....	3012	
McGinn, Dennis	Hewlett-Packard		
	An OSI Networking Architecture for Multi-Vendor		
	Networking.....	3007	

Miller, Marv	Hewlett-Packard		
	The Application Development Environment of the 80's.....	3052	
Naber, Lance	L.J. Naber & Associates		
	Ergonomics and VPLUS/3000 Screen Design.....	3020	
Neilson, Tom	Hewlett-Packard		
	Simple Steps to Optimize Transact/3000 Applications.....	3043	
Neuhauss, Peter	Hewlett-Packard		
	Techniques for Developing Device Independent Graphics Software.....	3090	
Olsen, Roger	Productive Software Systems		
	A Guide to Software Evaluation and Selection.....	3017	
Olson, Tad	Hewlett-Packard		
	The Effectiveness and Shortcomings of Using Programming Tools.....	3037	
Overman, James S.	EXXON		
	Manufacturing Application Experiences Implementing HP's Materials Management (MM), Production Management (PM), Maintenance Management (MNT) and HPFA.....	3001	
Porter, Steve	DP Systems		
	Turbo Pascal and AGIOS on the HP150.....	3050	
Rego, Alfredo	Adager		
	Natural Data Base Normalizing.....	3075	
Rego, Alfredo	Adager		
	The Drama Behind The System Status Bulletin (SSB)...	3082	
Remillard, Robert	Infocentre, Ltd.		
	Opportunities and Dangers of 4GL's.....	3004	
Rodriguez, Julia	Hewlett-Packard		
	The New COBOL Standard: "What's in it for You".....	3003	
Scheil, Dennis	Base 8 Systems Inc.		
	KSAM Survival Techniques.....	3049	
Schulz, Duane	Hewlett-Packard		
	Fitting Printer Technologies with Personal Computer and Office Applications.....	3073	
Scott, George B.	ELDEC		
	Using Process Handling to Optimize Throughput in a Transaction Oriented System.....	3058	
Scroggs, Ross	Telemon		
	Everything You Wanted to Know About Interfacing to the HP3000: The Inside Story.....	3055	
Setian, Kathy	Hewlett-Packard		
	Personal or Powerful.....	3002	
Shoemaker, Victoria	Mitchell Humphrey		
	Software Design: Building Flexibility.....	3044	
Simmons, E. R., Ph. D			
	Information and Humanity.....	3059	
Skrabak, John T	Baltimore Aircoil		
	HP3000--Gateway To Success.....	3022	
Snesrud, Wallace M.	General Mills		
	4GL and Reality.....	3021	
Solland, Leigh	Cognos		
	How to Design for the Fourth Generation.....	3013	

Stewart, Dwight	Hewlett-Packard		
	Spoolfile Recovery Without a Warmstart.....	3091	11
Sullivan, Charles	Pacific Coast Building Products		
	The HP3000: A Data Base Engine.....	3030	1
Tobak, Bruce	Consultant		
	Performance Optimization in COBOL.....	3057	11
Van Geesbergen, Rene'	Holland House		
	The Poor Man's DS, Fact or Fiction.....	3016	1
Volokh, Eugene	VESOFT		
	Secrets of System Tables..Revealed.....	3014	1
Wallace, Mark	Robinson, Wallace & Co.		
	4GL's: Use and Abuse.....	3085	11
Whitehurst, Otis	Vermont Housing Finance Agency		
	Writing Intelligent Software.....	3028	1
Wilhelm, Lisa & Lukoff, Stan	E.I. DuPont		
	Transact & 3rd Party Software Tools Used in a Large On-Line Environment.....	3069	11

Index by Title	Vol.
4GL and Reality.....	3021
Snesrud, Wallace M. General Mills	
4GL's: Use and Abuse.....	3085
Wallace, Mark Robinson, Wallace & Co.	
A Guide to Software Evaluation and Selection.....	3017
Olsen, Roger Productive Software Systems	
Anatomy of a True Distributed Processing Application.....	3029
Casteel, Michael Computing Capabilities Corp	
Application Prototyping: A Proven Approach to Information Systems Design and Development.....	3081
Larson, Orland Hewlett-Packard	
Auditability: or What's a Nice Byte Like You Doing in a Base Like This?.....	3061
Karlin, Robert Consultant	
A Blend of HP3000/HP9000 For Computer Graphics.....	3066
Boles, Sam Hewlett-Packard	
An OSI Networking Architecture for Multi-Vendor Networking.....	3007
McGinn, Dennis Hewlett-Packard	
Building Your Own X.25 Data Network.....	3040
Lewis, Donn Allegheny Beverage Corp.	
CIM Is Not A Software Package or a Magic Wire.....	3010
Floyd, Terry H. ASK Computer Systems	
Change Management: An Operations Perspective.....	3084
Hirsh, Scott RCM	
Communicating in a Mixed Vendor Environment.....	3031
Holt, Wayne Union College	
Cooperating Processes in an Information Network.....	3051
Gerstenhaber, Peter CMS Ltd.	
Developing Cost Effective Utilities and Applications Using Business Basic/3000.....	3067
Hoeft, Mark L. Hewlett-Packard	
Dictionary/3000--Extended Tour.....	3006
Butler, Stephen M. Weyerhaeuser	
Emulating A Real Time, Multi-Tasking Application System on the HP3000.....	3038
Fochtman, Jerry Exxon Chemical Americas	
Ergonomics and VPLUS/3000 Screen Design.....	3020
Naber, Lance L.J. Naber & Associates	
Everything You Wanted to Know About Interfacing to the HP3000: The Inside Story.....	3055
Scroggs, Ross Telemon	
Fitting Printer Technologies with Personal Computer and Office Applications.....	3073
Schulz, Duane Hewlett-Packard	
HP3000--Gateway To Success.....	3022
Skrabak, John T Baltimore Aircoil	
How Dispatching Queues Really Work.....	3065
Beasley, Dave Hewlett-Packard	
How to Design for the Fourth Generation.....	3013
Solland, Leigh Cognos	

Information Center: Implementation Using HP3000 and HP150.....	3026	
East, Ellie Media General		
Information and Humanity.....	3059	
Simmons, E. R., Ph. D		
KSAM Survival Techniques.....	3049	
Scheil, Dennis Base 8 Systems Inc.		
Manufacturing Application Experiences Implementing HP's Materials Management (MM), Production Management (PM), Maintenance Management (MNT) and HPFA.....	3001	
Overman, James S. EXXON		
MPE Disc Caching.....	3068	
Carroll, Bryan Hewlett-Packard		
Migrating Information Between HP3000 Data Bases, Electronic Spreadsheets and Microcomputer Data Bases.....	3042	
Kaminski, Thomas J. Singer		
Natural Data Base Normalizing.....	3075	
Rego, Alfredo Adager		
North American Response Center.....	3054	
Clifton, Roy Hewlett-Packard		
Operational Considerations for Police Networks.....	3077	
Kopecky, Jerry Illinois Criminal Justice Authority		
Opportunities and Dangers of 4GL's.....	3004	
Remillard, Robert Infocentre, Ltd.		
Performance Self Analysis.....	3025	
Duncombe, Brian Carolian Systems Inc.		
Performance Optimization in COBOL.....	3057	
Tobak, Bruce Consultant		
Personal or Powerful.....	3002	
Setian, Kathy Hewlett-Packard		
Positioning Local Area Networks.....	3092	
Clark, Brice Hewlett-Packard		
Rational Structuring Techniques for COBOLII/3000 Maintainability.....	3012	
McDermott, James T. Consultant		
Recovery by Design.....	3053	
Depp, James A. UPTIME		
Response Time: Speeding Up the Man/Machine Interface.....	3060	
Engberg, Tony Hewlett-Packard		
Secrets of System Tables.. Revealed.....	3014	
Volokh, Eugene VESOFT		
Simple Steps to Optimize Transact/3000 Applications.....	3043	
Neilson, Tom Hewlett-Packard		

Software Design: Building Flexibility.....	3044	
Shoemaker, Victoria Mitchell Humphrey		
Software Technology for the 80's (Understanding Key Current and Future Technologies).....	3009	
Franklin, Bill Hewlett-Packard		
Spoolfile Recovery Without a Warmstart.....	3091	
Stewart, Dwight Hewlett-Packard		
Store-and-Forward Data Transmission in a Multi-System Network.....	3046	
Korb, John P. Innovative Software Solutions		
Techniques for Developing Device Independent Graphics Software.....	3090	
Neuhaus, Peter Hewlett-Packard		
The Application Development Environment of the 80's.....	3052	
Miller, Marv Hewlett-Packard		
The Drama Behind The System Status Bulletin (SSB).....	3082	
Rego, Alfredo Adager		
The HP3000: A Data Base Engine.....	3030	
Sullivan, Charles Pacific Coast Building Products		
The New COBOL Standard: "What's in it for You".....	3003	
Rodriguez, Julia Hewlett-Packard		
The Poor Man's DS, Fact or Fiction.....	3016	
Van Geesbergen, Rene' Holland House		
The Role of The System Manager.....	3071	
Idema, Tom Westinghouse Furniture Systems		
The Segmenter.....	3005	
Boyd, Larry Dallas Times Herald		
The Sorted File Access Method.....	3036	
Chang, Wanyen Longs Drugs		
The Twilight Zone, Between MPE Capabilities.....	3045	
Grim, Jelle Holland House		
The Effectiveness and Shortcomings of Using Programming Tools.....	3037	
Olson, Tad Hewlett-Packard		
The Use of IMAGE Transaction Logging in a Multi Data Base, Multi Machine Configuration to Achieve A Non-Stop, Fault Tolerant HP3000 System....	3023	
Lawson, Roger Proactive Systems		
There's Got To Be A Pony Here, Somewhere.....	3024	
Cornford, M. G. Northrop Corporation		
Things That Go Bump in The HP3000.....	3018	
Bowers, Keith & Beauchemin, Denys Northern Telcom		
Training and Supporting Office Systems Users.....	3019	
Gross, Gail Hewlett-Packard		
Training: The Key To Success With Personal Computers.....	3011	
East, Ellie Media General		
Transact & 3rd Party Software Tools Used in a Large On-Line Environment.....	3069	
Wilhelm, Lisa & Lukoff, Stan E.I. DuPont		
Turbo Pascal and AGIOS on the HP150.....	3050	
Porter, Steve DP Systems		
TurboIMAGE Run Time Options.....	3039	

Kane, Peter	Hewlett-Packard		
The Ultimate Challenge in Application Design:			
Managing Data Integration.....		3074	
Isloor, Srekaanth	Cognos		
Unix Thru The Eyes of MPE.....		3083	
Boles, Sam	Hewlett-Packard		
Using Process Handling to Optimize Throughput in a Transaction Oriented System.....			
Scott, George B.	ELDEC	3058	
Using Intrinsic in COBOL Programs.....			
Clemons, Brett	Consultant	3027	
Why Software Projects Don't Quite Succeed.....			
Mattson, Robert R.	WIDCO	3034	
Writing Efficient Programs in Fortran 77.....			
Bircher, Carolyn	Hewlett-Packard	3076	
Writing Intelligent Software.....			
Whitehurst, Otis	Vermont Housing Finance Agency	3028	
You Said You Have a Bunch of Micros Linked To Your HP3000? Great!! Now What?.....			
Fisher, Eric S.	Wellington Management Co.	3035	

3001. Manufacturing Application Experiences :
Implementing

HP's Materials Management (MM),
Production Management (PM),
Maintenance Management (MNT),
and
HP Financial Application (HPFA).

James S. Overman
EXXON NUCLEAR IDAHO CO., INC. (ENICO)
P.O. Box 2800 1955 Freemont
Idaho Falls, Idaho 83401

Introduction

ENICO is a government contractor assigned the responsibility of implementing a new manufacturing facility near Idaho Falls, ID. The project was started in early 1984 and the decision to purchase an HP3000 with its Manufacturing Software was made in MAY 1984. The computer system was delivered in AUG 1984 and the Manufacturing Information System (MIS) implementation was begun.

This paper will attempt to provide an overview of ENICO's implementation experiences in its first year of using the HP3000 and its software.

System Configuration

The hardware configuration consists of an HP3000 Model 48 with four megabytes of main memory, three 404 megabyte 7933 disc drives, a 6250/1600 dual density 7978 tape drive, a 1600 density 7974 tape drive, several printers, and twenty+ terminals. The operating system was MPE V/P with the RAPID package, SPL, FORTRAN, and COBOL II for development tools.

Application Experiences

HPFA

The financial application was the first to be addressed by the programming staff. With HP's ASSIST program providing an application engineer to aid in implementation, an interface to an IBM procurement system was developed using magnetic tape as the data transfer media. The HPFA system was up and functioning in time for our beginning of year on 1 OCT 84.

As this was a startup operation of a new entity, ENICO was able to accept the HPFA system as provided, without having to adapt to old accounting practices and procedures. We have continued to utilize the HPFA software with only a minimal amount of customization to support a special budget reporting requirement.

In MAY 85, the HPFA software was converted from the originally installed 01.02 version to a newer 02.01 version. During this conversion, the few customizations which had been done were deleted and had to be re-entered. This was ENICO's choice, as we could have kept our changes and not accepted HP's enhancements, but it was decided to remain current with the latest software versions as HP releases them.

HPFA will eventually be linked to the other three applications, but the validity and usefulness of the cost data during startup was judged to be of questionable value and so was delayed.

ENICO purchased the entire HPFA package which includes General Ledger, Dual Ledger, Accounts Payable, Accounts Receivable, Allocator, Report Facility, and the Interface Facility. Thus, the utilization of all of its features is requiring a staged development plan. Overall, the package has fulfilled its promised capabilities and appears to be more than adequate for our needs.

Maintenance Management

MNT was a relatively new HP product when ENICO purchased it. We were told that we were among the first twenty-five users to take delivery. When installed in SEP 84, the software version was 00.00B, the second patch level to the initial release. Although planned for a later implementation than MM or PM, circumstances soon required that MNT be developed first.

The Document Control Section developed a need to track Vendor Documentation as it was received against contracted purchases. The Purchase Order capability of MNT appeared to lend itself to this document tracking requirement and so we proceeded to customize the software to perform the function. Both good and bad came of this early customization effort.

First, the package was proved to be quite "bendable" in that data fields were expandable to larger sizes and screens were cosmetically changed to refer to documents, etc. Second, the amount of effort to produce the various customizations was learned. Generally, the changes required more thought and effort to implement than the training classes had indicated. This is due in most part to the inter-relatedness of the application. A change in one area often has repercussions and side-effects in another. However, with adequate fore-thought and analysis, extensive changes were completed in a very short time with far less effort than building a document tracking system from scratch.

The bad news came as the actual Maintenance staff began to use the system for its intended purpose. Then the side-effects and uncovered bugs began to really show themselves. Problems in the MNT software with expanded data fields in particular, as well as several other bugs, caused much debugging effort and long hours

on the phone to the Response Center. Finally, in Jan 85, we updated to the first major fix level 00.01 and the problems were mostly corrected. The Maintenance users began "production" use of the system in JAN 85. User training was conducted in FEB 85, on the system, using all portions of the software. Eventually, the Vendor Data system was separated from the Maintenance databases. They continue to use the same programs while having their own databases, screens, and reports.

The HP ASSIST program showed one of its deficiencies here. The Application Engineer could not devote the necessary time on-site to delve into all the bugs and their corrections. Extensive training classes were conducted and application expertise was given as to how the package should work, but cranking through the Prep Dictionaries and testing and retesting were ENICO's responsibility. It should be mentioned that the Response Center in California was very patient and helpful during a rather trying period.

Not to be daunted, ENICO has purchased the Advanced Customization Option (User Exits) and have implemented a custom Purchase Requisition capability as well as custom Work Orders and extensive data validity checks beyond those provided by HP. Also, a Tool Management need has been seen to be 85 percent implementable using MNT. Custom programming is being used to add the remaining portion under the MNT application umbrella.

All-in-all, ENICO is satisfied with the MNT package and continues to broaden our usage of its features.

Materials Management

MM was installed in SEP 84, version 07.04, and proceeded to be virtually unusable due to recurring software aborts. After a month of negative progress, we purged the MM account and re-installed MM on version 07.05. After correcting a few installation oversights, the software finally settled down and only produced a few minor glitches over the last quarter of 1984. The upgrading of the Application Tools from 02.04 to 02.06 also corrected several operational hassles and stabilized the MNT, MM and PM products.

The Lot Control Module was purchased to fill a lot traceability need. Unfortunately, this MM software does not provide an efficient method of serializing individual parts and assemblies. After some fruitless attempts to discover how other MM users had solved the problems, ENICO developed a proposal to "insert" serialization data into the Lot Control system. The Development Lab listened to our proposal and indicated that it should work and would be glad to hear how it all comes out. Once again, ENICO forged ahead, purchased Advanced Customization for MM, and is busily implementing a Serialization feature combined with HP's Lot Control Reports. The results of this effort will not be known

until late summer 85 when we begin to enter production histories into the MM system.

The use of the Standard Product Costing Module was only begun in MAY 85 with no measureable results as yet. The Master Production Scheduling Module was purchased in MAY 85 to develop some Management What-If scenarios and has yet to be utilized in other than a test mode.

Production use of the MM package starts in JUNE 85 with the Purchase Order receipts and Inventory Control portions being used. Work Order Scheduling should begin in the OCT 85 timeframe. The large size of the MM application, as well as the linkage to the PM system, has served to stretch out the planned implementation. The HP ASSIST program has provided needed on-site training and implementation expertise, but has not served to actually perform the initialization and definition tasks which must be completed prior to full utilization of the packages features.

Production Management (PM)

Although version 02.02 of PM was installed in OCT 84, little was done with it, other than that required to support MM, until JAN 85 when PM was upgraded to the 03.01 release. A pilot/walk-thru was conducted in conjunction with MM and all major features of the software were exercised in MAR 85. Detail definition of the workcenters, operations, and part routings are in progress and should be completed well before the planned Production usage in the Fall of 85.

As the production area is to be controlled by multiple DEC VAX computers, they will replace the normal user terminal functions of PM with an automated interface to the HP3000 and a background "batch" execution of the production data transactions. Luckily, the development of the VAX software (by independent contractors) is taking longer than planned and the HP side of the interface is still under development. Advanced Customization will be used to extract the serialization information from the normal PM transaction stream, and to insert the data into MM as discussed above.

The HP ASSIST program has failed to provide much help, beyond moral support, in solving the communication problem. This appears to be due to the unique nature of the problem and HP's general lack of enthusiasm for communicating to DEC machines directly from the HP3000. ENICO has currently chosen a terminal emulation approach to the HP-DEC communication problem, and is hoping that a better solution will surface prior to volume data transfers begin. The recent Ethernet announcements of HP appear to have the greatest promise of being the required computer link.

Other Applications

Graphics

Extensive use has been made of the HPDRAW package to produce presentation slides and proposal graphics. Limited use of the DSG package has been due to the lack of actual production data. The HP2627A color graphics terminals have proven to be the best match for the HP7545A six pen plotters. The 2623A graphics terminals are used mostly for normal block mode applications.

Text Processing

TDP is used as the standard programmer editor. Program documentation and some larger procedure documents have been developed using TDP. HPSLATE is used by some employees as a quick and dirty memo maker. HPSLATE's ease of use has proved to be very good and will probably expand its usage with time and greater availability of terminals.

Management Support Systems

In addition to the Budget application, ENICO has developed two simple tracking systems for management action items. These were programmed using Transact and Report with V-Plus and Image. We are able to provide rapid implementation of these special requests using these standard HP products. The inefficiencies of the final run-time routines are not measureable on the system, probably due to the small sizes of the databases involved.

System Performance

HP has repeatedly warned ENICO that the Model 48 may not have the capacity to support all of the applications which we have purchased. The user terminal response times have generally been maintained at a very good level, but many steps have been taken to keep the system performing well.

Probably the greatest impact on the system came from the starting of the HP applications. The system Load process would consume the entire system for thirty seconds or more when one of MM, PM, MNT, or HPFA were started. Once loaded, the Application monitors would spawn several son processes and open several databases. When a user activated an Application Module, the application program would tie up the Load process for tens of seconds and then open from three to seven databases. This situation was greatly improved by allocating virtually every program in the system. At system startup, a Job is run to perform the allocations which requires about ten minutes of stand-alone cpu time. All databases have had their BUFFSPECS parameters set to larger values for all users via DBUTIL (Typically, 24(1/120)).

Other general steps taken were the optimization of UDC's, restricting the JOB limit to one at a time, and activating disc caching at all times. The OPT program was purchased to aid in the performance tuning effort as many of the Contributed Library

programs (SOO, TUNER, IOSTAT) no longer work with the MPE V enhanced tables.

The conversion from MPE V/P F.00.01 (the resegmented version) to MPE V/E T-delta-1 is planned for late JUNE 85.

Hardware Experiences

Terminals

ENICO has eight flavors of HP terminals: 2621A, 2622A, 2623A, 2624B, 2627A, 2392A, 3093A, and HP150's. This mix of terminals has produced many problems. Some applications did not support the newer terminals (the 2627A and the 2392A) and software upgrades or special terminal configurations had to be made. The HP150's cause special problems as occasionally the PC will generate an error message which the user confuses for an undocumented application message. Also, keyboard layouts, power-on procedures, and configurations are different and require greater user training and documentation.

If your company has a choice, standardize on one or two types of terminals. ENICO has chosen the 2392A as the terminal of choice for all future purchases. Use of the HP150's as part-time terminals is only recommended for knowledgeable users who can decipher any unusual error conditions that may occur.

Printers

HP's relatively new 2563A 300 line per minute dot-matrix printers have proven reliable and dependable. We have them attached via the HPIB and the ATP (9600 baud) to a statistical multiplexor and leased phone lines. The HP2934A Serial Office Printers provide 200 cps draft listings and correspondence quality output at 67 and 40 cps. ENICO is planning on placing several of these printers in separate buildings for light-duty local printing.

The HP2565A 600 lpm dot-matrix printer experienced several problems:

1. The 132nd column was not being printed due to an MPE bug. This problem had been identified and corrected three months prior to delivery of our machine but HP had not supplied the patch nor well documented the bug.
2. Slow ribbon movement, due to a bad batch of motors, caused poor printing quality and was not detectable by ENICO as the ribbon speed was not discussed in the printer manuals. Correction of this problem also delayed the resolution of the next problem.
3. Short ribbon life due to stretching and uneven wear which caused the ribbons to fold and jam, was eventually corrected by a factory team visit that replaced a ribbon guide. We are now

getting thirty or more print hours per ribbon as opposed to eight or less previously.

4. One print position hammer did fail and was replaced. Also, some characters dropped print dots on startup at times, this problem was corrected also.

Nonetheless, the printer is fast with good print quality. The high density print option is used for final documentation and correspondence. The printer is interfaced via the ATP at 19200 baud over a 400 foot long RS232 cable.

Tape Drives

The HP7974A 1600 density tape drive has been reliable. The HP7978A 6250/1600 dual-density tape drive is the unit of choice for Backups due to the greater capacity per reel. Some difficulty was experienced in configuring the auto-reply feature, but this was finally worked out and nightly stores can now run unattended.

Communications Equipment

As do all 3000/48 computers, we have an ADCC for the console, modem, and two more ports. The remainder of the ports are two ATP's with RS232 cable options. With the ATP's, an upgrade to the 3000/68 would not require changing the terminal cabling. Also, the 19200 baud transmission rate is available for those terminals that can use it.

Two MICOM Micro 800/2 eight port statistical multiplexors with Paradyne LSI-96 modems are used to support a remote site. No problems have been experienced with the terminals running in block mode at 9600 baud. One of the 2563A printers is also operating at the remote site.

ENICO is planning on using a broadband Local Area Network (LAN) among several buildings. A SYTEK LAN has been under test to develop the required interface configurations of various devices to the LAN. Development of the necessary device and interface box parameters is a slow process which requires much manual reading (SYTEK and HP), physical configuration, and testing. LAN's are not transparent media and are not "just like a special cable". However, the reduction in cabling and the distances covered at high baud rates seem to be ample justification for the implementation effort.

HP Support

Idaho Falls, Idaho has a HP sales and hardware support office, but no software engineers. Thus, ENICO chose to try the Response Center Support (RCS) option for all software. This choice has been a good balance between the computer system group's needs and

the rapid problem resolution possible with a local HP Systems Engineer.

The HP ASSIST program was purchased with all four major application packages. The Application Engineers were very helpful in training inexperienced users and in describing the software functionality to management. Their expert advice on matters of implementation choices and procedures saved many man-hours of time. The Applications Needs Analysis document was a disappointment as it was mostly boilerplate and sales hype generalities. For those customers without large, experienced application staffs, or those who require help in getting started, the HP ASSIST program could be useful. Remember that HP does not actually do the work, rather they ASSIST you in planning and directing the implementation effort.

Future Plans

ENICO is currently purchasing twenty more terminals and half a dozen printers to support the manufacturing areas. We anticipate a computer upgrade to a Model 68 in late 85 or early 86. Additional Applications packages are being considered such as HP's Production Cost Management and a Project Scheduling Package.

Closing

I wish to thank ENICO management for supporting the development of this paper. The opinions and conclusions contained within are those of the author and may not represent the official policy of ENICO.

3002. PERSONAL OR POWERFUL:
CAN WE COMPUTE BOTH WAYS?

Kathy Setian
Hewlett Packard
Cupertino, CA, USA

I. SYNOPSIS

The rapid proliferation of personal computers in business is proof of the growing demand for computing power that is available quickly and easily. Basic human needs for intuitive, flexible tools are being met by manufacturers of personal computers with varying levels of success.

But the personal computer alone cannot satisfy all requirements. Powerful processors, standardization and control are some of the strengths of mainframes and minicomputers. Many companies have sizeable investments in these data resources which users must tap in order to be effective.

The personal computer, ergonomically designed as an extension of the user, can provide an optimum human interface to the network of corporate data resources.

The combined approach of attention to human factors and development of integrated capabilities for personal computers results in an information network that is personal AND powerful.

II. PERSONAL COMPUTING: A QUANTUM LEAP IN DISTRIBUTED PROCESSING

By any measurement, the tremendous growth of personal computers has been an undeniable phenomena, one that data processing professionals in corporations today cannot ignore. Consider some of the statistics:

- From less than 200,000 personal computers in use in 1977, the installed base will grow to 80 million units by 1987, worldwide (1). We are witnessing exponential growth.
- Today, the shipments of personal computers roughly equals that of minicomputers worldwide; by 1987, shipments of personal computers will equal that of mainframes (1).

These statistics are impressive when we consider that in 1975, personal computers were toys for hobbyists. But the numbers are even more awesome when we focus on the penetration of personal computers into the business arena:

- Although only 1/3 of the units being shipped go into business as opposed to homes, 3/4 of the dollar value go to businesses (1). This means that much of the serious computing power is being directed into businesses.

This trend toward business applications comes as no surprise when we consider the history of personal computing. With the advent of electronic spreadsheets which offered timely budgeting and "what-if" analysis, and with word processing, cost-justification of personal computing in the business environment became practical. Personal computers expanded their influence from the home into the corporate environment.

But are personal computers here to stay or are they just a passing fad? Compare the penetration of personal computers to other popular inventions with lasting impact:

- The rapid acceptance of personal computers is similar to the pattern of penetration of televisions in the post-war period (1).
- The degree of penetration of telephones into businesses over a 75 year period will take personal computers only 10 years to accomplish (1).

Personal computers are here to stay; and they will change the way we do business in as fundamental a way as telephones transformed corporate life.

What are the needs that personal computing answers in such an undeniable way? Simply put, personal computing addresses a fundamental need to know and to communicate. While distributed data processing has gone a long way toward addressing this need, personal computing represents a quantum leap forward. Historically, distributed data processing meant moving mainframe computing from the corporate data center to local departments with minicomputers. The next step, which is really a leap in order of magnitude, is to move the computing power onto the desktop.

The attraction of this step to end-users is undeniable. The personal computer is personally manageable. Long procurement cycles, high level sign-offs, and lengthy software development cycles can be circumvented. And the personal computer offers the promise of "ease of use". This term used to apply to programming ease, but now it suggests that one need not be a programmer or computer specialist at all. Personal computing is, or should be a "people's" tool.

How successful have personal computers been at fulfilling their promise as friendly, easy, personal tools?

III. COMPUTING CAN BE PERSONAL: ERGONOMICS

To be accessible to the legions of potential users, personal computer design must overcome some formidable barriers. The amount of time spent learning how to use a new tool, and the amount of frustration provoked in the process must be severely lowered. This is largely a matter of system and software design, and the most successful software vendors in the long run will be those who decrease learning curve constraints creatively. Higher processing capacities and artificial intelligence are also parts of the solution, but will not be covered in this paper.

The goal of a system designer taking human factors into account should be to relate the user's goal to use of the system in a natural or intuitive way. The user should be able to figure out what to do and how to do it without obstruction from the system. In fact the system should facilitate the user in accomplishing tasks. Menus as software interfaces presented a partial solution to this design goal, but menus also gave rise to a new question: what is the easiest way to make selections from a menu?

This brings us to a second challenge, one complementary to software design. It is related to the problem of keyboard limitations. Many alternate input and control devices have been developed over the past years, each with its own strengths and application niches. What follows in this section is a discussion of keyboard limitations and a survey of the most popular input alternatives. These devices are all part of the continuing microprocessor revolution with the goal of personalizing the human/computer interface and tailoring it to the task.

KEYBOARD LIMITATIONS

The following are some of the reasons which have been advanced to explain user discomfort with the keyboard as the sole interface to the computer:

- Many new computer users lack typing skills, making the keyboard an ineffective tool and a deterrent to personal computing.
- Unspoken yet powerful subjective factors include keyboard avoidance due to its "non-executive" connotation.
- Environmental factors such as "hands-free" or low light environments.
- Problems of data accuracy and reliability when entering data.
- The profitability problem caused by the labor intensive attribute of keyboard data entry.
- The productivity drawbacks of long learning cycles.

For these reasons and many others, several alternatives to keyboards have been developed and explored, and ergonomic factors have influenced the success of these new devices.

The goals of ergonomic design are to enhance the comfort and efficiency of the human interface. Unfortunately, since efficiency is the more measurable of the two in terms of ROI (Return on Investment), the "comfort" goal tends to be overlooked by hard-line bottom-liners. However, it must be remembered that a tool which is more comfortable, intuitive, natural and less frustrating is more likely to be used in the first place.

There are more than 20 types of devices used as alternatives to keyboards, and more are being developed. We will briefly survey those devices which are most likely to be of benefit to personal computer users in a business environment: touch sensitive screens, mice, joysticks, track balls, graphics tablets and digitizers, bar code readers, video imaging, and voice recognition.

TOUCH SENSITIVE SCREEN

The most natural, intuitive method of making a selection is by pointing. For this reason, the touch sensitive screen is perhaps the most learnable interface available today. It experiences the high levels of user acceptance since it requires no training. It is superior even to the popular mouse, trackball, or joystick since the latter devices require the user to translate information from a vertical plane (video display) to a horizontal plane. Further proof of high user acceptance is the fact that touch screens are being used in public videotex systems and information kiosks where novices walk off the street and interrogate the system without any assistance or instruction.

There are several touch screen technologies in use. Hewlett Packard has pioneered the use of infra red LED's and phototransistors in an array around the screen. A finger (or other pointer) breaks the infra red beams registering the xy coordinates of the intersection. Every line and every other character can be accessed giving a 27x40 matrix.

Other touch technologies include resistive membranes which create a voltage when touched, capacitive sensing which requires an overlay, and acoustic surface waves. Most of these technologies have shortcomings related to reliability, durability, clarity or resistance to environmental factors.

The best niche for touch screen interaction is among executives who must retrieve data but not necessarily enter data. HP's electronic rolodex application uses the touch screen to its best, most intuitive advantage. Vertical application areas where the touch screen is often used include, as an example, stock and commodity traders who are not inclined to take time to use the keyboard. Also some public information systems use touch interfaces.

THE MOUSE

Perhaps the most well-known alternative to the keyboard is the mouse. Sometimes described as an "upside down track ball", it allows users to point to commands or objects on the keyboard with just a little practice. Developed by Xerox in 1968, the mouse became associated with Apple computers in 1983 when it was adopted as their standard interface.

The best applications for the mouse include those that rely on high resolution graphics such as CAD, CAM, and graphics design. For freehand graphics applications, nothing is more akin to the stroke of a brush than the mouse. A disadvantage is the relatively large amount of open work space it requires to be moved across. This contradicts the reality of most people's desks in a business environment. Another limitation is that it is clearly not suitable for portable personal computing.

A new variation is a stationary touch pad located on a keyboard that functions like a solid state mouse.

JOYSTICKS AND TRACKBALLS

In the mid seventies, the burgeoning video game industry borrowed joysticks and track balls from the Apollo space program for home entertainment. However, it is surprising how many of these devices are sold on personal computers. It has been estimated that at least 25% of personal computers have one of these devices, probably due to the large number of business computers that make their way home and double as entertainment devices.

Track balls provide 360 degree motion control, while the joystick offers a more limited number of angles of motion. However, the joystick is much less expensive.

Niches for joysticks and track balls include CAD, CAM, robotics, aerospace and military applications. Because it takes less desk space than the mouse, the track ball may be preferred in some offices.

GRAPHICS TABLETS AND DIGITIZERS

With an electric stylus that picks up current as it passes over a pad, tablets and digitizers compete with OCR (Optical Character Recognition) for scanning and storing graphics. Application niches are high resolution graphics oriented such as CAD, CAM, graphics design, or industrial training.

BAR CODE READERS

In the mid seventies the first standard emerged for bar codes. The Universal Product Code (UPC) is now widely used in retail stores and supermarkets. A predefined set of black magnetic ink bars with varying widths are read by a scanner. During the past decade, several other standards have been established for different industries.

The applications for bar code readers have expanded from point-of-sale to inventory and process control. In the early eighties the development of laser-based hand-held scanners contributed to their use in industry since operators would not have to touch the unit to read its number. The advantages of bar code systems include speed, accuracy, low cost and convenience as an input device.

VIDEO IMAGE INPUT

This is a young, small but growing area of technology. A video camera is used to capture and store images (graphics and data) in digital form. The images are stored on videodisc or videotex systems (with the latter being cheaper and easier to update). Educational applications and training companies are likely to benefit from this technology. Today, the chief disadvantage is cost, especially due to the high storage requirement. Today videotex is sometimes used with interactive devices, for example in public information systems.

VOICE RECOGNITION

Voice recognition, along with touch screens, offer the most natural form of human/computer interaction, although voice recognition technology is in its infancy and needs much more development before it can be regarded as a practical keyboard alternative. Today, most voice recognition systems are speaker dependent, meaning that the system must learn and store a vocabulary for each user. By the 1990's, it is hoped that voice recognition systems will be speaker independent and will accomodate continuous speech.

Even today, there are niches for voice driven data entry especially in "hands busy" environments such as assembly lines and airports. As the technology improves and the costs come down, voice input could play a major role in the office, where even today voice-over-text editing can be found.

WHITHER THE KEYBOARD?

With all of the development and excitement of alternate input devices, what will be the role of the keyboard? Will it be replaced entirely?

Studies show that the one class of users that consistently chooses the keyboard over other devices is skilled typists. For these people, it is a disadvantage to take the hands off the keyboard to use another device.

Other indications are that while the alternate devices are much preferred during the learning period, they can reach a point of diminishing return, at which time the user will frequently revert to keyboard use. Although in these cases the alternative device can be regarded as transitional, in the long run, the new devices will be seen as a complement to the keyboard. The keyboard will be used as a general purpose device, while the alternative devices have particular application niches as has been indicated.

STANDARDIZATION

Not only is there a wide variety of types of input devices, but within each category there are different implementations. For example, there are both mechanical and optical laser mice, and there exists no standard for connections to computers. Some use RS232 interfaces, others use plug in interface cards, and each device transmits different signals.

Beyond hardware incompatibility, there is an associated cost in software development. Often a software product must be modified to accomodate drivers for different devices.

Hewlett Packard has pioneered a solution to this problem through its HP HIL (Human Input Link). This is an interface board that connects up to 8 devices. Future HP software products will include an HP HIL driver so that alternate input devices can be added without necessarily modifying the software. Our recently introduced Graphics Gallery software incorporates the HP-HIL driver. Perhaps independent hardware and software vendors will choose to take advantage of the HP HIL concept as well.

IV. COMPUTING CAN BE POWERFUL: THE WORKGROUP CONNECTION

Once an interface device is selected which is natural for the user and appropriate to the task, the next challenge for the personal computer user is to integrate the data gathered through a variety of methods and work in tandem with all of the other computer resources located in the company. The essence of powerful computing is this ability to assimilate information from diverse, often traditional sources.

Historically, the information flow within a company was greatly dependent upon the topology of the data processing network controlled by the corporate mainframe computer. The limitations of the mainframe solution included inflexibility, lengthy procurement and development cycles, lack of remote access, and a division between data processing professionals and the business professionals they purported to serve.

The introduction of minicomputers and the concept of distributed data processing contributed greatly to solving these problems. The network became more flexible and easier to tailor to the needs of the users by distributing the computing power on a site or departmental basis. Personal computers should be viewed as an extension of this concept and implementation scheme. Ideally, the network should include an appropriate combination of personal computers for single user applications, minicomputers for departmental communications and shared data, and mainframes as required for processing and consolidation of corporate data.

As we have seen, end users are bringing personal computers into the processing environment in record numbers, and this will be true whether or not the company has a masterplan for integrating them into the workplace. Furthermore, farsighted business professionals know that they can not stand on the sidelines until such a masterplan is developed before they partake of the advantages of PC technology. To do so would be to concede the productivity and profitability benefits to their competitors.

Today, some companies find themselves with a new dilemma. Traditional corporate data processing resources and personal computers exist as two separate realities in information management. This situation presents problems. It often indicates a duplication of effort within the company as each department or individual sets out to resolve their needs, many of which are common to other users. Often times leveraged solutions are overlooked as companies rake up the costs of peripherals and software that are not shared among users. Furthermore, the traditional benefits of centralized data processing have been overlooked by PC enthusiasts: data integrity, on-line accuracy, security, back-up, and auditability. All of these issues are seldom if ever addressed by the isolated personal computer user.

Vendors of microcomputers made an initial attempt to bridge the gap between personal computers and corporate data resources by offering terminal emulation capabilities. Clearly this is a limited first step offering little more than the ability to crudely transfer files. It also uses little if any of the power of the personal computer which may be relegated to the status of a dumb terminal.

To fundamentally address the challenge of integrating personal computing with our corporate data resources, we must take a radically different approach, one that might be characterized as "bottoms-up". This means that we must start from the premise that each professional within the organization wants to use the personal computer as an entry point into the network of data processing resources.

As we survey the capabilities needed in an integrated business environment, we see that the propagation of personal computers has if anything increased the importance of departmental minicomputers. However, their application mix has changed to

support more multi-user data base oriented applications, and communication functions. Productivity and decision support software is moving off of the mini-computer and onto the personal computer.

Let us look at the kinds of capabilities a truly integrated personal computing and data processing environment would offer, and note how many of these capabilities are within reach with the appropriate planning and development.

INFORMATION ACCESS AND MANAGEMENT

The issues most critical to powerful personal computing revolve around the ability to access data that is stored on one or more departmental level computers, and to manipulate that data in a meaningful fashion as input to a variety of PC applications. There is no justification for a return to the "dark ages" of pre-data base management systems, when each application "owned" its own copy of the data. Rather, we must take the next step forward in distributed data processing by allowing the PC user to extract those portions of the corporate data bases that are relevant and to which he or she has authorized access.

There can be no doubt that such a solution requires extensive sophisticated network level software in order to become a reality. On the other hand, the theory and direction are already in place and have already been subscribed to by those who have preached the benefits of distributed data processing since the late 1970's. What is required is an extension of that commitment and an extension of those software tools.

Some of the software which lends itself most readily to an integrated PC and departmental computer solution include:

PC-based inquiry facilities with a menu driven interface that allow the PC user to easily access data.

Relational data bases so that data can be extracted (projected) and combined (joined) without predefined limitations.

Data dictionaries which locate the requested data on the user's behalf.

Network directories that handle the routing of the data transparently for the user.

The above software facilitates data access; equal attention must be paid to the other side of the coin: data control. A complete solution addresses the concerns of security, data integrity, back-up and restoration of data, and auditability, among others. The integrated PC and departmental computer solution is the most advantageous method of offering these services to PC users. They

are simply an extension of the control mechanisms that have already been designed for workgroup computers alone.

INTEGRATED APPLICATIONS AND COMPUTING ENVIRONMENTS

For software solutions to be effective as well as powerful, there must be some integration of PC applications. For example, sophisticated users will not put up with a situation in which data must be rekeyed from a spreadsheet application to a data base application. The popularity of integrated software products such as 1-2-3 and Symphony from Lotus or Framework from Ashton-Tate attests to the importance of integration.

Integration across computing environments is another important goal within the context of combined workgroup and personal computing. PC users want to be able to move their data and applications from the PC to the workgroup computer and vice versa without consciously initiating a data conversion or switching from one interface to another.

COMMUNICATIONS

Although users like the autonomy and control that personal computers give them, one of the first capabilities they request beyond the bounds of a stand-alone workstation is to communicate with their co-workers. This may mean that they want to broadcast a request for information, send out memos and reports, schedule meetings, share data, request feedback on a draft proposal, etc. These communications requirements can be predicted by numerous studies which show that people communicate mainly within the same workgroup or organization. Only 14% of written communication is addressed to people outside of the company, a surprisingly small percentage.

An electronic mail system running on the workgroup computer or network offers PC users an appropriate solution. The challenge is to tightly integrate personal computers into the electronic mail network. For example, a user might create a spreadsheet or a memo using a popular PC software package, and then want to mail it to a coworker sitting at the next desk or half way around the world. This process of moving from a local processing environment on to an extensive distributed processing network must be performed smoothly and invisibly to the user.

SHARED RESOURCES

Personal computer users may or may not have their own printers, plotters, and other peripherals, but it is unlikely that each user would be willing to bear the expense of higher performance and/or higher quality equipment. The best solution to shared use of such peripherals is one that enables the PC user to redirect output to equipment that is attached to the workgroup computer. This provides a method of leveraging the cost of peripherals and still

giving PC users convenient access to their output which is located within the immediate workgroup or department.

All of these capabilities constitute the goals and strategic directions for Hewlett Packard's Personal Productivity Center. Hewlett Packard has a unique advantage in its capability to integrate personal computers with office automation, data management and communications. With the personal computer as the professional workstation of choice, and with the minicomputer processing departmental level data, services can be provided which automate and control all of the above functions.

V. CONCLUSIONS AND A LOOK TO THE FUTURE

The personal computer has made an enormous contribution by making computing power available to business professionals. This advance could not have been accomplished without solving some of the human factors problems which stood in the way of wide-spread acceptance, especially among non-typing, non-computer trained executives.

In order to avert the potential disaster of dual, separate computing environments (PC's and traditional resources with no connection), hardware vendors are challenged to produce an integrated strategy. Mini-computer vendors with a long history of commitment to distributed data processing (such as Hewlett Packard) are in the most advantageous position to make a contribution by extending the concept of distributed data processing down to the desktop. This strategy provides a framework within which we can address the need for information access and control, integrated applications and environments, communications, and resource sharing. The total solution requires the planning and execution of software both on the workgroup computer and on the PC. Properly implemented, users can have it all: computing that is both personal and powerful.

What new needs will surface as the current needs are being addressed? Foremost is the need for portable workstations with the same capabilities that have been described for stationary PC's. The current revolution has gone a long way toward addressing the needs of relatively sedentary office workers, but there are many professionals who are not office-bound. People working in sales and service and executives who travel a great deal are first on the list, followed by a wide range of specific workers from medical ambulance personnel to musicians, writers, and other free spirits who find their inspiration on remote mountain tops.

Another need which already exists but which will become more urgent is the need to integrate solutions among multiple hardware and software vendors. Despite the "shake-outs" which periodically occur in new industries, no single vendor could possibly satisfy all market requirements. Therefore strategies for co-existence, compatibility and integration will increase in importance.

Last but not least is the continued demand for ergonomic design in both hardware and software. We have barely begun to create systems that can truly be labeled "easy to use" or "easy to learn." As microcomputers become more powerful, more functionality will become transparent. One by one, the barriers to use will be removed and the extent of training necessary to get started will decrease. The coming years will push today's limits in both the personal and powerful dimensions.

FOOTNOTES AND REFERENCES

1. International Data Corporation: 1984 Industry Briefing
2. International Resource Development Inc.: Non-Keyboard Data Entry, 1984.
3. Chester, Michael: "The mouse and the trackball face off in PC marketplace", Systems & Software, December 1984, pp. 48-50.
4. Roman, David: "Building up Your Personal Computers: Data Input Devices", Computer Decisions, pp. 111 - 128.

Biography

Kathy Setian is a Product Specialist in the Personal Computer Group at the Hewlett Packard computer facility in Cupertino, CA.

3003. The New COBOL Standard: "What's in it For You"

Julia Rodriguez
Hewlett-Packard Information Technology Group

ABSTRACT

A new COBOL standard will soon be adopted by both ANSI (American National Standards Institute) and ISO (International Standards Organization). This standard is the culmination of one of the most controversial standardization efforts in data processing history. Three public reviews took place. Nearly 4000 letters from individuals and corporations formally took positions against adoption of the standard during the first two reviews.

Most of the early objections to the proposed standard centered around incompatibilities with COBOL'74. Most of these incompatibilities have been removed, but a few remain. This paper will discuss those features incompatible with COBOLIII/3000, as well as some of the very attractive new features that the proposed standard will offer.

This paper will also present some techniques to assure smooth migration of code not only to the new standard, but also to other architectures. In addition, the standardization process will be briefly described.

BACKGROUND

The current version of ANSI COBOL was adopted in 1974. Since 1977, the ANSI X3J4 committee has been working on a new version of the COBOL standard. The new standard is now in the final stages and will be published toward the end of 1985 or in early 1986. Since it is not clear how long the process will take, this revision of the standard will be referred to herein as COBOL'8x. Readers who have been following the trade press know that the proposed standard has been controversial indeed! A draft of the proposed standard was submitted for public review in October, 1981, and by the end of the review period in February, 1982, over 2200 letters had been transmitted to X3J4, most of them protesting potential incompatibilities. A second draft received similar response. In 1984, a third draft of the standard was published containing only a few generally minor incompatibilities. This final draft has been well received and has been endorsed by international COBOL groups. Features described in this paper reflect this latest version of the proposed COBOL'8x standard.

OVERVIEW

The next standard will have changes in the following categories:

- a. New Features
- b. Obsolete Features

- c. Specification Changes
- d. New Reserved Words

A significant effort has been put into incorporating structured programming constructs into COBOL'8x. In addition, other new facilities have been added to make programming in COBOL'8x easier. Some features have been flagged for deletion in the subsequent standard. Those features which are in the new standard, but which are not expected to be in the subsequent standard have been placed in a category called obsolete. There have also been some changes to the rules and the addition of reserved words which may affect existing programs. It is these rule changes and new reserved words which are incompatible with COBOL'74.

The COBOL standard has always contained a list of implementor defined items. These items pose potential problems for those desiring portability between different architectures and implementors.

STRUCTURED PROGRAMMING

The new structured programming constructs which have been defined for COBOL'8x include Scope Terminators, NOT phrases, nested programs, PERFORM statement enhancements, the EVALUATE statement, and the CONTINUE statement.

Scope Terminators

Under COBOL'74, conditional statements could not be included with the statement group following a conditional phrase such as AT END or ON SIZE ERROR. New reserved words have been added such that any conditional statement can be turned into an imperative statement and used as part of the conditional statement group by adding a word of the form END-verb. For example,

```
READ FILE-IN AT END
  ADD A TO B ON SIZE ERROR
  PERFORM OVERFLOW-ROUTINE
  END-ADD
MOVE SPACES TO REC-IN.
```

Under COBOL'74, it is not legal to specify the ON SIZE ERROR phrase in the above example because it turns the ADD statement into a conditional statement and only imperative statements are allowed following the AT END phrase. (Without this rule in COBOL'74, it could have been ambiguous as to which statement was being processed).

However, in COBOL'8x the scope terminator END-ADD allows the ADD statement with the SIZE ERROR option to become an imperative statement. The MOVE statement is then the second imperative statement of the AT END, and the period terminates the READ. If

the READ itself were to be nested under a conditional such as an IF, it would be terminated by an END-READ instead of the period.

NOT phrases

In addition to the conditional clauses which handle exceptions (i.e. AT END, ON SIZE ERROR, INVALID KEY, AT END-OF-PAGE), a clause has been added to the statements which executes when the exception does not occur. For example,

```

READ FILE-IN
  AT END DISPLAY RECORD-COUNT
  NOT AT END ADD 1 TO RECORD-COUNT.

```



This statement would add 1 to RECORD-COUNT if a record from FILE-IN was read. If there were no more records to read it would display RECORD-COUNT.

Nested Programs

The nested program facility allows programs to be contained within other programs so that global data may be easily shared and the program structure and relationships specified. In the following example, program B is contained within program A.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. A.
ENVIRONMENT DIVISION.
DATA DIVISION.
  [Global Data Declarations]
PROCEDURE DIVISION.
  [Program A Procedure Division Statements]

IDENTIFICATION DIVISION.
PROGRAM-ID. B.
ENVIRONMENT DIVISION.
DATA DIVISION.
  [Local Data Declarations]
PROCEDURE DIVISION.
  [Program B Procedure Division Statements]
END PROGRAM B.
END PROGRAM A.

```

Program A may call program B; however, program B cannot call program A. Program B can access data in program A which is declared as GLOBAL unless program B contains a local data item of the same name.

PERFORM Statement Enhancements

The PERFORM statement has been enhanced to allow a list of imperative statements to be embedded within the statement instead of paragraph names and to allow the programmer to specify whether

the UNTIL conditions are to be tested before or after the specified set of statements has been executed. An example of an in-line PERFORM is shown below:

```
PERFORM 10 TIMES
  ADD A TO B
  ADD 1 TO A
END-PERFORM.
```

The two ADD statements will be executed 10 times.

Under COBOL'74, the UNTIL conditions are always tested before executing the specified paragraphs. The new specifications will allow the test to be made afterwards. For example,

```
PERFORM READ-LOOP
  WITH TEST AFTER
  UNTIL EOF-FLAG.
```

Control will always transfer to READ-LOOP at least once. The test option may also be specified with an in-line PERFORM.

EVALUATE Statement

The EVALUATE statement adds a multi-condition CASE construct to COBOL'8x. The EVALUATE statement causes a set of subjects to be evaluated and compared with a set of objects. If the comparisons are all true, a specified group of statements is executed. For example,

```
EVALUATE HOURS-WORKED ALSO EXEMPT
  WHEN 0          ALSO ANY  PERFORM NO-PAY
  WHEN 1 THRU 40  ALSO ANY  PERFORM REG-PAY
  WHEN 41 THRU 80 ALSO "N"  PERFORM OVERTIME-PAY
  WHEN 41 THRU 80 ALSO "Y"  PERFORM REG-PAY
  WHEN OTHER                                PERFORM PAY-ERROR.
```

The above example evaluates two data items, HOURS-WORKED and EXEMPT. If HOURS-WORKED is 0, any value for EXEMPT will be true and NO-PAY will be performed. If HOURS-WORKED is between 1 and 40, REG-PAY will be performed. If HOURS-WORKED is between 41 and 80 and EXEMPT contains "N", OVERTIME-PAY will be performed. If HOURS-WORKED is between 41 and 80 and EXEMPT contains a "Y", REG-PAY is performed. If none of the above conditions are true, PAY-ERROR is executed.

CONTINUE Statement

The CONTINUE statement is a no operation statement which indicates that no executable statement is present. It may be used anywhere a conditional statement or an imperative statement may be used. For example,

```
IF A < B THEN
  IF A < C THEN
    CONTINUE
  ELSE
    MOVE ZERO TO A
  END-IF
  ADD B TO C.
  SUBTRACT C FROM D.
```

The CONTINUE statement allows control to go to the ADD statement following the IF when A is less than C. If the NEXT SENTENCE option had been used, control would have transferred to the SUBTRACT statement instead. It may also be used in place of an EXIT in an EXIT paragraph.

OTHER NEW FEATURES

There is a long list of other new features which should make the job of the COBOL programmer easier. The more significant ones are listed here.

Reference Modification

Reference modification allows you to reference a portion of a data item by specifying a leftmost character position and a length. For example,

```
MOVE A (3:5) TO B.
```

will move the third through seventh characters of A to B.

INITIALIZE Statement

The INITIALIZE statement provides the ability to set selected types of data fields to predetermined values. Assume RECORD-1 was described as follows:

```
01 RECORD-1.
   05 EMP-NO    PIC 9(6).
   05 EMP-NAME  PIC X(20).
   05 EMP-PAY   PIC 9(5)V99.
   05 JOB-TITLE PIC X(20).
```

The following INITIALIZE statements in the Procedure Division could be used to put values into the record:

```
INITIALIZE RECORD-1 REPLACING NUMERIC BY ZERO
REPLACING ALPHANUMERIC BY SPACES.
```

The effect would be the same as:

```
MOVE ZERO TO EMP-NO EMP-PAY.
MOVE SPACES TO EMP-NAME JOB-TITLE.
```

De-editing

Under COBOL '74, it is not legal to move from an edited field to a numeric or numeric edited field. The new specifications will allow moving from a numeric edited item to either a numeric or numeric edited item. The edited item which is the sending item will be converted to its numeric value and moved to the receiving field.

REPLACE Statement

The REPLACE statement function is similar to that of a COPY... REPLACING except that the REPLACE statement operates on all source program text, not text in libraries. Thus, if one of the new reserved words is used heavily in an existing program, you may want to use a REPLACE statement to change it. For example,

```
REPLACE ==TEST== BY ==TESTT==
```

will replace all subsequent occurrences of TEST by TESTT in the source program until another REPLACE statement, a REPLACE OFF statement, or the end of the source program.

Optional FILLER

The word FILLER is now optional for data items which will not be referenced.

```
01 A.  
05 B PIC X(5).  
05 PIC X(5) VALUE "NAME:".
```

New USAGE data item formats

The USAGE clause may specify USAGE IS BINARY or USAGE IS PACKED-DECIMAL. USAGE IS BINARY indicates a radix of 2. USAGE IS PACKED-DECIMAL indicates a radix of 10 and that each digit occupies the minimum possible configuration computer storage. USAGE IS BINARY corresponds to USAGE IS COMPUTATIONAL in COBOL II and USAGE IS PACKED-DECIMAL corresponds to USAGE IS COMPUTATIONAL-3.

Initializing arrays

The VALUE clause may be specified for a data item that contains an OCCURS clause. The effect will be that each item of the table will receive the value specified in the VALUE clause.

```
01 A.  
05 B PIC S999 OCCURS 10 TIMES VALUE 0.
```

INITIAL Attribute

The INITIAL clause in the PROGRAM-ID paragraph indicates that every time the program is called, the internal data is initialized. This function is the same as the \$CONTROL DYNAMIC option on the HP-3000.

PROGRAM-ID. SUB-PROG INITIAL.

EXTERNAL Attribute

The EXTERNAL clause specifies that a data item or file is available to every program in the run unit which describes the data item or file.

FD FILE-1 IS EXTERNAL.

SYMBOLIC CHARACTERS Clause

The SYMBOLIC CHARACTERS clause in the SPECIAL-NAMES paragraph of the Environment Division allows the programmer to equate a name to a specific character. This feature can be useful for unprintable characters. For example,

SYMBOLIC CHARACTERS BELL IS 8, CARRIAGE-RETURN IS 14.

This clause would allow a MOVE statement such as

DISPLAY BELL " JOB COMPLETED ".

CLASS clause

The CLASS clause of the SPECIAL-NAMES paragraph of the Environment Division allows the programmer to specify a class name. This class may be used in a class condition in the PROCEDURE DIVISION. For example, given the following declarations:

SPECIAL-NAMES.

CLASS VALID-GRADE IS "A" "B" "C" "D" "F".

...

WORKING-STORAGE.

01 GRADE-LIST.

05 CLASS-GRADES PIC X OCCURS 5 TIMES.

The following IF statement could be used:

IF GRADE-LIST IS NOT VALID-GRADE THEN
PERFORM ERROR-ROUTINE.

The above statement would perform ERROR-ROUTINE if GRADE-LIST contained any character other than "A", "B", "C", "D", or "F".

ADD Statement Enhancement

Under COBOL '74, the ADD statement allows either a TO or a GIVING format, but a statement of the form

```
ADD A TO B GIVING C
```

is not allowed. The new specifications will allow the TO before the last operand when the GIVING option is used.

Alphabetic Tests

Two new alphabetic class tests have been defined:

1. ALPHABETIC-UPPER will be true if the data item being tested contains only A-Z and spaces.
2. ALPHABETIC-LOWER will be true if the data item being tested contains only a-z and spaces.

SET Statement Enhancements

The SET statement has been enhanced to allow the setting of external switches either on or off and condition-names to true. For example, given the following declarations:

```
ENVIRONMENT DIVISION.  
SPECIAL-NAMES.  
SWO IS SWITCH-1  
...  
WORKING-STORAGE SECTION.  
01 READ-FLAG      PIC 9.  
   88 EOF-FLAG    VALUE 1.
```

The following SET statements could be used:

```
SET SWITCH-1 TO ON.  
SET EOF-FLAG TO TRUE.
```

The second SET statement is equivalent to:

```
MOVE 1 TO READ-FLAG.
```

OBSOLETE CATEGORY

There are some features of the current standard which are scheduled for a phased deletion. Implementations must still support these features in the new standard, but not in the subsequent standard.

The Identification Division paragraphs are included in the obsolete category in favor of the more general comment facility (* in column 7). Part of the reason for this change is the problem with the use of the word COPY in these paragraphs. It is not clear whether COPY in a comment entry is intended to be a COPY statement or is merely part of the comment.

The ALTER statement is being made obsolete because it is widely accepted as a poor programming practice which causes significant program maintenance problems.

The ENTER statement, RERUN clause, and MEMORY SIZE clause are being made obsolete because they are primarily implementor defined functions which are not necessarily meaningful on all systems and are thus not portable. Implementations will still be allowed to support these three features as extensions to the standard.

The LABEL RECORDS, VALUE OF, and DATA RECORDS clauses of the file description entry have been made obsolete. The DATA RECORDS clause was made obsolete because it is redundant information. LABEL RECORDS and DATA RECORDS were made obsolete because their function is considered a part of the operating system and does not belong in a COBOL program.

The segmentation module and the debug module of the standard are being made obsolete because they are based upon a technology that is now obsolete. The debug module facilities were considered to be better served by an interactive debug facility which would need no COBOL source statements. The segmentation module function is provided at the operating system level.

OTHER CHANGES

New status code values for file errors are being defined. These codes will cover error situations but for which no standard status codes were previously defined. Programs which check for specific status codes may need to be changed to check for additional error conditions or to check the more general case. It was felt that the added granularity of the new status codes provided a functionality that justified its incompatibility. For example, trying to open an indexed file in a program which declares it to be a relative file receives a status code 39.

The order of the steps in a multi-conditional PERFORM...VARYING statement has been changed. Under COBOL'74, the statement

```
PERFORM PAR-1 VARYING I FROM 1 BY 1 UNTIL I>10  
AFTER J FROM I BY 1 UNTIL J>10
```

would set I to 1 and vary J from 1 to 10 and then set J to 1, increment I to 2 and vary J until 10. The new specifications will increment I to 2 before setting J to I. Thus, on the second cycle, J will vary from 2 to 10 instead of 1 to 10 as under COBOL'74. The primary reason for this change is because this statement, as currently defined, has caused much confusion because it doesn't do what most people expect and is probably not used very much. Programmers who have attempted to use this statement to do a bubble sort have usually been surprised at the results. Few programs are expected to be impacted.

The new reserved word ALPHABET is required in the alphabet clause of the SPECIAL-NAMES paragraph.

ALPHABET ASCII IS STANDARD-1.

This change was required because of the desire to minimize the portability problems caused by implementor-defined reserved words. Under the COBOL'74 standard, the implementor could reserve the words used for switches, alphabet-names, and output advancing controls. The new standard will not allow these words to be reserved. This change however caused a parsing problem in the SPECIAL-NAMES paragraph because it would not be clear whether a clause such as

```
SWO IS EBCDIC
```

was specifying that EBCDIC was the mnemonic-name for switch SWO or whether SWO was the mnemonic-name for alphabet EBCDIC. By requiring the word ALPHABET in a alphabet-name clause, the ambiguity is resolved.

An implicit EXIT PROGRAM is assumed when there is no next executable statement. This was an undefined case in COBOL'74. It was felt that specifying an action would enhance portability of programs.

When the receiving item of a MOVE is of variable length (eg. the data item contains an OCCURS DEPENDING ON clause) the maximum length will be used. This change was made to avoid loss of data. Only those program which have other data following a variable length table will be affected by this change.

RESERVED WORDS

The following new reserved words have been added:

ALPHABET	END-DELETE	EVALUATE
ALPHABETIC-LOWER	END-DIVIDE	EXTERNAL
ALPHABETIC-UPPER	END-EVALUATE	FALSE
ALPHANUMERIC	END-IF	GLOBAL
ALPHANUMERIC-EDITED	END-MULTIPLY	INITIALIZE
ANY	END-PERFORM	NUMERIC-EDITED
BINARY	END-READ	ORDER
CLASS	END-RECEIVE	OTHER
COMMON	END-RETURN	PACKED-DECIMAL
CONTENT	END-REWRITE	PADDING
CONTINUE	END-SEARCH	PURGE
CONVERTING	END-START	REPLACE
DAY-OF-WEEK	END-STRING	REFERENCE
END-ADD	END-SUBTRACT	STANDARD-2
END-CALL	END-UNSTRING	TEST
END-COMPUTE	END-WRITE	TRUE

PORTABILITY

Primary portability concerns are caused by implementor defined clauses in the DATA division. These are SYNCHRONIZED, USAGE IS INDEX, COMP, COMP-3, BINARY, or PACKED DECIMAL. The only truly portable usage is DISPLAY. All other usages are implementor defined. Programmers need not abandon other usages and SYNC to achieve portability; however, it may be wise to just restrict their use if a program is intended to be used on multiple architectures. For maximum portability implementor defined data should not be placed in a file to be read by another machine. If you are going to do so, be sure to investigate that the definition of these items is the same on the machines you are going to. Also, records or data items declared with these clauses should never be redefined. To redefine an item requires that you know the exact size and starting position of the each data item in the record. This is not constant from machine to machine for implementor defined usages, and synchronizations.

Copies of the draft standard may be obtained for \$25.00 from

CBEMA/X3 Secretariat Staff
1828 L St. N.W.
Washington, D.C. 20036

Individuals or organizations interested in participation on X3J4 may write to:

Anne Vermilion
X3J4 Membership Chairman
Hewlett-Packard Information Technology Group
19447 Pruneridge Ave. M.S. 471z
Cupertino, Ca. 95014

3004. OPPORTUNITIES AND DANGERS OF
FOURTH GENERATION LANGUAGESROBERT REMILLARD
INFOCENTRE LTD.
MISSISSAUGA, ONTARIO, CANADA

SUMMARY

The objective of this paper is to review the direction and definition of Fourth Generation Languages (4GL's) and address the impact that they will have on data processing departments and their users. We will look at several real examples, review the dangers and opportunities of 4GL's, look at their benefits and suggest practical tips on how to put them to work for your company. Along the way, we will look at the changes that are needed in data processing to accept and successfully implement 4GL's.

BACKGROUND

Before we go into the definitions and concepts of 4GL's, let us review the trends that brought them into existence in the first place.

A recent computer industry forecast predicted that the number of installed computers would increase by a factor of ten (10) in the next ten years. That was before the fantastic growth created by the micro-computer. This means that the demand for software would increase considerably.

In a Computerworld survey, it was indicated that the number of computerized applications in existing data processing departments is growing at a minimum rate of 45% per year. Again, this indicates that a lot of new software is going to be required from data processing staff.

Computerworld also estimated that the average cost of one debugged computer instruction is approximately ten dollars (\$10.00). What this means for data processing managers is that their goal should be to reduce the number of lines or instructions required for an application in order to reduce the cost of development.

There is also a very interesting fact about our industry. Last year, North America produced more computers than programmers and science graduates combined. At this rate, we cannot continue to develop software in the same way as before.

As everybody that has attended an HP sales presentation knows very well the cost of hardware keeps going down for the same (or improved) software performance, while the cost of people increases all the time. This indicates that programmer productivity has to increase to keep up with the hardware trends in our industry.

A recent article in Computerworld indicated that the average application backlog in North American DP departments is between three and four years. This is only the visible backlog of documented requests from users. It does not include what James Martin calls the invisible backlog. This is the one where users do not even consider making a request for the applications they need because they know that it will take several years before they receive a response.

In summary what we see right now is that the traditional approach to systems development has to change drastically in order for every DP department to survive in the next decade. People productivity has to increase dramatically if we do not want every human being to be a programmer by 1995.

SOLUTIONS

Fortunately, there are solutions to the development bottleneck experienced today. The first solution that appeared in the marketplace is pre-packaged application software. This is especially true of the HP3000 family since there are many good packages available.

Pre-packaged applications usually offer a good starting point for the most common needs (ie accounting, manufacturing, etc).

Another solution is to let end users create their own solutions with powerful tools. This trend has been accelerated lately with the new type of software on micros, such as spread sheets, file managers and graphics.

Finally, another approach is to have systems analysts create applications, rather than write program specifications. The benefit of this approach is that the time required to develop an application is greatly reduced through prototyping, where the user gets instant feedback on how the system might look and operate. This avoids costly misinterpretations and the generation of applications to meet needs that changed long ago.

No single solution will necessarily be the only solution for your company. DP managers should always evaluate each approach for different requirements and make a decision on the best alternative(s).

HISTORY AND DEFINITION

We are hearing a great deal about Fourth Generation Languages. Let us first review some history and where they come from. The first generation was the machine language (binary). Then came the second generation of "Assembler" type languages. The third generation was that of the supposedly machine independent languages such as Cobol, Basic, Pascal, Fortran, PL/1, RPG, etc. This is the generation that most of us are familiar with. It provide more friendly interface with the machine, but is still driven by programmer logic.

My own definition of a fourth generation language is: "A non procedural data base language that produces results in one-tenth of the time than with Cobol, or less". This is also close to James Martin's definition of 4GL. What it really means is that it should be a language that is action driven instead of logic driven. With a 4GL, one should tell the machine what to do instead of how to do it, and the results should be user friendly. It means that data processing's productivity should be improved by a factor of ten or 1000% to qualify as a 4GL. Also, a language that works exclusively with indexed files instead of data base management systems would not qualify as a 4GL.

SOME CONCEPTS (BUZZWORDS)

There are several categories of software tools that fall in the 4GL family. Let us review them:

- A) Simple Query facilities. They have been around since the first disc based systems. They allow stored records to be printed or displayed in an appropriate format.
- B) Complex Query languages. These are like the Query on HP3000, allowing retrieval and joining of records based on certain qualifications or parameters. Traditionally these languages are poor performers on the machine.
- C) Report generators . These are tools for extracting data from files and formatting it into reports, while allowing substantial arithmetic or logic processing before displaying or printing it. These tools have been around for a while in the HP3000 environment and are the first wave of accepted fourth generation tools.
- D) Dictionaries simply are files in which is stored pertinent information about data and their use by programs.
- E) Systems or applications generators . These represent the latest trend in 4GL's. They contain modules that permit an entire application to be generated.

The best application generators are created with the user in mind and will automatically generate user friendly applications that include menus, online help, error messages, security structure, etc. Systems generators, to be classified as fourth generation tools, should generate a non procedural code and definitely not a third generation code such as Cobol. Tools that produce third generation code are known as code generators and can be classified as third generation productivity tools.

- F) Documentation tools . This concept is now expanding the scope of 4GL's. As we all know, the implementation of an online application is usually only as good as the user documentation manuals that come with it. The better 4GL's offer facilities that automatically generate these manuals.
- G) Parameter-driven packages . The latest packages include parameters that can be changed by the users to make them fit their own environment. The spread sheets on micros are good examples of parameter-driven user applications. MM/3000 is also an example of this generation with the customizer option.

CASE STUDIES

Now that we all know the terminology, definitions and history, let us look at some real situations of companies and institutions that have benefited from using 4GL's.

These cases are documented internally at Infocentre, or have been taken from examples given by James Martin in his book, "Applications Development Without Programmers".

- Case 1 : In the Chase Manhattan Bank an end user department itself created a complex system for online analysis of the Chase's management accounting data in ADMINS II. They accomplished this in 4 months with two people and claim that the DP department had estimated they would take 18 months with 20 people at a cost of \$1.5 million.
- Case 2 : Mark-Hot, a manufacturer in the province of Quebec, had a backlog of applications estimated at 3 years. Using SPEEDWARE on an HP3000, and the same number of people on staff, they got rid of that backlog of applications in 6 months.
- Case 3 : At Infocentre, we have an application called T.O.U.R.S. It was written in VIEW/3000 and Cobol. The application needed six people to support it for a base of four clients. The listing printout was over 10 inches thick when on a table. The application was mostly re-written in SPEEDWARE and the listing is now less than two inches thick. We still have four people to maintain it, but our base of clients has grown to over fifteen sites.
- Case 4 : Corfax Benefits System is a software supplier of pension benefits systems. The software is both sold as a package or rented on a time sharing basis to client companies in Canada. The application is mostly written in SPEEDWARE with some Cobol subroutines. They estimated the increase in the area of maintenance productivity to be over 2000%.
- Case 5 : Mohawk College of Hamilton Ontario had two programmer/analysts do an evaluation of SPEEDWARE. As a test, the management gave to them what was estimated as a one year backlog of applications to play with. The two created all the applications from the one year backlog in three weeks. They are now using and teaching SPEEDWARE.

Case 6 : B.A.S.F., a world wide manufacturer of magnetic media has achieved an increase in programmer productivity estimated at 2500%, using SPEEDWARE instead of V/3000 and Cobol. Moreover, the DP manager tells us of improved morale in his department since users are now very happy with the turnaround time of new projects.

Case 7 : A world wide manufacturer with a division in London, Ontario had a purchasing control application to develop. The estimated development in V/3000 and Cobol was over 7 months for an application without user friendliness, menus and documentation. The same application was developed, tested, documented and implemented in 3.5 weeks, using SPEEDWARE. Moreover, the end result had online help and was controlled by menus.

CHANGE NEEDED IN DP

Hopefully by now we have convinced you that 4GL's can work and are for real. This brings the first major change needed in DP management for 4GL's to be accepted. There has to be an open mind to the new technology and concepts. As vendors of a 4GL, we often find that our single most important problem is that people do not believe that we can do what we do. A lot of companies have decided that all development must be done in Cobol or another "passe" language, deciding automatically against improving productivity. When Hewlett-Packard announces a new microchip that doubles the performance of the hardware, nobody seem to reject the claim (except IBM maybe). However, when a software manufacturer announces a new tool that can double programmer's productivity, most DP managers have serious doubts about it. Their view can be likened to that person in the 1950's, who was asked to get from Montreal to Los Angeles in half the normal time required. He took his car to a high performance shop, changed the engine, boosted the power, improved the suspension, added a bigger tank for fuel. When asked why he did not go by plane instead, he simply answered: "Those things, they will never fly!"

The second change needed in DP management concerns the wall between the user and the application creator. It has to disappear with 4GL's. This wall consists of written specifications that must be frozen, a multi-year backlog, slow programming, lengthy program documentation and the fact that most bugs are in analysis and design. By the time the application finally arrives, the original requirements have changed.

Prototyping is a new concept introduced by 4GL's. Programmer/analysts can now sit with the user and generate a prototype application that will serve as a start up systems design model. When the 4GL has a user documentation generator, the user can then take the documentation back with him/her and write the required enhancements on it. The users do their own analysis. Also, when in doubt with a new application, try a prototype instead of a formal analysis. The cost might be less.

Another change needed in DP is that we have to start managing information instead of code. After all, we are called Managers of Information Systems, not Managers of Code. Our role is to give the users access to the data that they require to perform this task. Our role has to become a guide to the information, by knowing where the information is and then telling the system what we need instead of how to retrieve it.

The new applications created have to be user friendly and flexible enough so as to accommodate more than one need. If the application is friendly, you will not get numerous phone calls from users who do not understand how to use your application. A way to generate flexible applications is to always include substitution parameters in your applications. A report could, for example, ask a user directly for the sort criteria. With one report program, you may be creating in effect 25-50 different outputs that will be defined at run time by the users. The reports become user-customizable and multi-functional.

We also have to avoid being a maintenance department. (see figure 1). The maintenance pitfall is very real when we develop more and more applications that we have to maintain later. 4GL's can help in that area, since the code generated is usually one tenth to one twentieth the amount of Cobol needed for the same application.

Finally, DP managers have to look for existing packages instead of trying to re-invent the wheel all the time. Today, there are a lot of very good application packages that can offer a good fit for 80-90% of the needs for certain applications.

DANGERS OF 4GL's

One of the most serious dangers associated with 4GL's is the power given to users. For example, users can write their own inquiries into large data bases. This can create a situation whereby, a user could read a 200,000 record data set sequentially to find a keyed customer number and then re-read the same data base again sequentially to extract the customer address. Obviously the system performance would suffer tremendous degradation. This can easily be avoided by retaining some level of control at the data processing level.

Another danger of 4GL's is to try to duplicate old systems with the new technology. For example, if you have an old application that was running in a certain manner and you try to redesign it using a 4GL, you should always try to take advantage of the new features and technology instead of duplicating what is there. 4GL's have tremendous benefits when simple guidelines are followed.

Control needs are greatly increased when a data processing department uses 4GL. Since many new applications would be implemented quickly, security measures around the system need to be revamped. There would be an increase in the number of data bases and files on the computer and this should be monitored in order for the data processing department to become an Information Centre.

One of the more unapparent dangers of 4GL's is at the implementation level. Since you will be creating a lot of new systems with this new technology, there will now be a shift in the backlog of application from the data processing shop, to the users. Your shop will now produce and try to implement more systems than your users ever dreamt possible.

What this means is that you have to start streamlining all of the implementation procedures. Although today's fourth generation languages are becoming increasingly powerful, you should not try to do everything with the 4GL. For example, some calculations could possibly be better done and/or simplified by exiting to a Fortran subroutine. With your increasing use of the 4GL tools, you will find those areas of application development that can be developed easier and faster, and performed more efficiently with a third generation call.

The last danger associated with fourth generation languages is their performance. Unfortunately, some 4GL's available today, while offering you a great improvement in the development cycle, are poor performers when put into production. However, this is not a generalized case and it should not divert you from going with a 4GL. It is however, one of the criteria which should be evaluated when choosing a tool for your system.

BENEFITS OF 4GL's

Obviously, the first benefit of 4GL's is the huge savings that can be had by companies that use them. As we have shown in our case studies, the increase in productivity can range from 700% to sometimes 2500% and even 5000%. That represents substantial savings in personnel costs.

With a 4GL, your backlogged applications, for which the development in a 3GL would take at least 10 times as long, can be put in to production a lot faster. This means that the savings generated by those same applications can start happening for the company a lot faster.

Another major benefit of 4GL's is the fact that through code reduction, maintenance is a lot less important. Your staff will now be able to concentrate on satisfying users needs with new projects, instead of maintaining old obsolete applications.

Happier users and increased morale in data processing is also associated with the use of 4GL's. Obviously, if you can serve your users better, they in turn will show to your management how satisfied they are with your services. This will give your people more pride in their work.

One of the most underrated benefits of 4GL's is the automatic standardization that is achieved through their use. As you well know, some companies will pay hundreds of thousands of dollars to consultants to get standardized methodologies in their departments. Most 4GL's, or at least the good ones, will provide you with standards at the data entry level, at the reporting level and even sometimes at the documentation level.

This leads to another benefit, namely the computer generated documentation. As we pointed out earlier, the very best fourth generation languages will offer automatic creation of user documentation manuals. This feature enables users to be transparent to the system, by allowing their replacement if need be.

Also, 4GL's tend to generate user-friendly applications from day one. This includes online help, error messages, friendly menu driven user systems, and so on. These features should also automatically be standardized by the tools. The benefit of the data processing professionals is that implementation is greatly facilitated.

Finally, one of the strangest benefits of 4GL's for programmers is that it gives them more time to play computer games. I will not attempt to translate this into a benefit which management could understand.

HOW TO MAKE THEM WORK FOR YOUR COMPANY

The first step in making a 4GL work for your company, is obviously, how to select it. First, you should always evaluate, more than one 4GL. It is no different than evaluating a computer. If possible you should try to get a trial tape of the software, load it onto your system and begin to develop applications. Very soon in the evaluation process, you will feel your own level of confidence with the tools. This trial gives you, a feeling for the support given by the vendor. In this trial process think of your users. Any new generated system should be given to the users for evaluation of the user-friendliness. We often neglect them. Also, during this trial period, try to develop a feeling for the performance of the tools while in production. As we pointed out earlier, fourth generation languages vary greatly in performance, some of them being splendid performers while some of them are dogs on your system.

In order to make a 4GL work, you will have some selling to do. First you have to sell to your high management the increases in productivity, the faster response to their needs, and the overall improvement in your shop. The second step is to sell it to your own DP staff. They will fear the new tool as competition for their jobs. This should be dismissed early in the process if you want to be successful.

Finally, you have to sell your new tool to the users. They are the ones that will benefit from faster turnaround, friendlier systems and improved documentation.

When you implement the new tools in your environment, start with the easiest applications. This will bring early success that will show your management and your users the power of the tools very early in the cycle. It will also familiarize your staff with the new technology.

With a 4GL, you have to learn to take advantage of the new tools. You will have to start to change the way of developing systems, by using prototyping for example. As we mentioned earlier, do not try to duplicate old systems the way they were.

Your next step in a successful implementation of a 4GL, is to change your role to that of a consultant. You will now have tools to give your users access to the data. Your role should now be to guide your users on how to get to the data. You have to end the big mystery surrounding data processing. With a 4GL, you no longer need that mystique to retain your own security in your shop. Your results will now speak for themselves and your status will be further enhanced in the corporation.

The last paragraph described the consultant approach. This also means that you need good control on where the information is stored. You will now have to develop file references, or dictionaries to tell you where the data is stored. Again with more new applications coming online, this is critical.

You will also need to maintain a catalogue of applications and data bases. The reason for this is that you will create so many applications and programs that very often you will find a user requesting something that already exists. Good systems documentation will prevent duplication of systems.

You should avoid unproductive study of systems. When in doubt, prototype the application instead of spending time studying it. If the prototype works, (and you will out find very soon in the cycle), then you try to turn it to production right away. This will save a lot of time and fully take advantage of the new tools.

A last suggestion is to always keep good communication lines open with the vendor of your tool. As a vendor I can tell you that we know a lot about how to make our tools work in companies. Very often we find that our users don't use us enough as a good reference on how to use our tools. The vendor should be able to refer you to another clients who have solved similar problems.

This technology is very new and finally is being accepted. There are very few good reference books, manuals, or training on

how to make them work for your company. So please use your vendor as a direct source of information.

CONCLUSION

Fourth generation languages are now here to stay and those that do not believe this fact will be left behind, like those who continue to use an abacus instead of a calculator.

Although there are some dangers associated with their use, their tremendous benefits make them the avenue to the future. So open your minds and fasten your seatbelts. You are in for one brand new era.

BIOGRAPHY

Robert Remillard has a bachelor of Commerce degree from University of Montreal. He is the Marketing Manager of Infocentre Ltd., and has been associated with data processing since 1976.

3005. THE SEGMENTER

Larry Boyd
Dallas Times Herald
1101 Pacific
Dallas, Texas 75202

Several HP/3000 programmers, analysts, and system managers have questioned the readability of the MPE SEGMENTER manual (including myself). This paper is designed to help define the concept of the SEGMENTER. It is an introduction to the SEGMENTER and SEGMENTER manual. It will describe the four parts of the SEGMENTER, how they interrelate with each other, and the basic use of them.

The four major parts of segmenting are Relocatable Binary Modules, Relocatable Libraries, User Subprogram Libraries, and Segmented Libraries. For simplicity, we will use HP's abbreviations: RBMs, RLs, USLs, and SLs.

Quickly stated, segmenting is used to manage main memory to improve performance. This can include reducing the size of a code segment to decrease the amount of main memory needed at one time, or increasing the size of a code segment to decrease the number of memory swaps. It may be necessary to decrease a code segment because the segment is too large to fit into the segment limit size set by MPE (16K words). Also, Code Segment Table (CST) size problems can change segmenting schemes. These are not all the reasons for segmenting, but they are some of the most common.

For starters let's study the most common situation (See Figure 1). HP gives us several subprograms that are loaded in the system SL. A system manager will add to this SL all the necessary additional subprograms. The application programmer will write a program using no sections, compile it, PREP it, and finally RUN it. If the programmer calls a subprogram (such as an IMAGE routine), it is automatically loaded at run-time from the system SL. The programmer is never aware of the process. If the programmer needs a new subprogram for several applications, it is given to the system manager and he/she loads it in the system SL. With this type of situation, we have several problems that will directly affect performance.

So let's look at the flow of a source program (See Figure 2). After writing a source program, we will compile it. At this time our compiled code is located in a USL file. This code is object code, but it is not in an executable format. If the source was subdivided into sections, it will create multiple

Relocatable Binary Modules within the USL file, otherwise, there will only be one RBM.

Once the source code is in a USL file, the USL file can be prepared into an executable file if it contains a main entry-point (OUTER BLOCK). USL files without an OUTER BLOCK cannot be prepared as stand alone programs. A USL segment without an OUTER BLOCK can be relocated into an SL using the SEGMENTER by the segments name, or an RBM that is not an OUTER BLOCK can be loaded into an RL using the SEGMENTER by the RBM name.

So the options with a source program is first to compile it into a USL file and then either 1) prepare it into an executable stand alone program if it contains an OUTER BLOCK, 2) relocate the segment(s) which do not contain OUTER BLOCKS into an SL file, or 3) relocate the RBM(s) that are not OUTER BLOCK into an RL file.

THE RELOCATABLE BINARY MODULE:

So let's study RBMs. The RBM is a block of compiled code, grouped in a USL by segments. Each RBM in a USL file has at least one entry-point. The RBM will have an entry-point with the same name, and entry-point names stay the same throughout the USL, RL, and SL files. Consider an RBM as a procedure and an entry-point as an address to the procedure.

When RBMs are added to a USL file they are flagged as "active." If an RBM already exists in the file with a duplicate name, the original RBM is de-activated (status - "inactive"). So if an RBM exists in the file with the same name, but not the same code, the original code will be de-activated. Care must be taken not to overlay a valid RBM with a different RBM because they have the same name. When you are adding an RBM to an already existing file, check the file to make sure that there are no conflicting RBM names.

THE USER SUBPROGRAM LIBRARY FILE:

Now let's look at USL files. A USL file contains compiled code with one or more RBMs. Several source files can be compiled into the same USL file, but only one OUTER BLOCK is allowed (although none are required). To prepare a USL file into an executable file the MPE 'PREP' or the SEGMENTER 'PREPARE' commands can be used. If the USL file is prepared using one of these commands, all RBMs that are located in the USL file will be loaded into the executable program file.

To relocate a USL segment into an SL file the SEGMENTER command 'ADDSL' is used (See Figure 3). All of the RBMs in the

segment will be loaded in the SL. The SEGMENTER command 'ADDRL' is used to relocate an RBM from a USL to an RL (See Figure 3).

USL files can be named any valid HP file name. They can be created using the SEGMENTER command 'BUILDUUSL' or by using one of the many HP compilers. If you compile a program without specifying a USL file name, it is given the temporary file name of \$OLDPASS. USL files that build executable programs do not have to be saved, but it is recommended to ALWAYS save the USL files for RLs and SLs. The reason for this is that if an RL or SL runs out of space, you must build a larger one and load all of the segments or RBMs into the new one. The time needed to compile your executable file is used to decide if you need to save it's USL file. If the time is extremely long, save the USL file and re-compile only the source programs that are modified.

When a permanent USL file is used several times it will become cluttered with de-activated RBMs. These USL files must be cleaned. This is done using the 'CLEANUSL' command in the SEGMENTER. Or you can initialize a USL file with the \$CONTROL statement in your source program (consult your language manual on the \$CONTROL statement).

THE RELOCATABLE LIBRARY FILE:

Now let's examine an RL file. A Relocatable Library consist of RBMs as stated above. It must be built using the SEGMENTER, and can be PURGED or RENAMED using MPE. It can be named any HP-valid file name and it consists of compiled code from a USL file. It is added to an executable program at PREP time, using the ";RL=rlfilename" option (See Figure 4). This prepares the RL code into the executable code of the program file, and adds all requested entry-points in the RL to the program. Entry-points are added to and deleted from RLs simple by their entry-point name.

One problem to consider when using an RL is that all requested RBMs from the RL are loaded into one segment. This could cause the RL segment to become too large for the MPE imposed segment limit (which is one reason we use RLs). If you cannot reduce the number of referenced RBMs in the program to the RL file, you may consider loading some of the RBMs into the group or account SL. Or you can move some of the RBMs to the program USL file before preparing it. This will allow you to load the RBMs into the desired segments.

THE SEGMENTED LIBRARY FILE:

So what is an SL? An SL consists of segments that are executable statements with entry-points and returns. The SL is built and maintained using the SEGMENTER. These executable statements are added to the SL as stated earlier by segments, but if an entry-point is to be deleted, it is purged by its name -

not the segment name. Deleting all entry-points deletes all segments. It does not delete the SL file - you now have an empty SL file. If you wish to PURGE or RENAME the SL file, use the appropriate MPE command.

*** NOTE *** When modifying an SL file, it will lock all users trying to load a segment from that SL.

To use an SL file it must be named "SL." This means you can have only one accessible SL in a group. To use an SL, you select it on the RUN command: ";LIB=G", ";LIB=P", or ";LIB=S" (default) (See Figure 4). The entry-points are searched in the SLs using this criteria:

```

;LIB=G          - 1. SL.Group.Account
                  2. SL.PUB.Account
                  3. SL.PUB.SYS

;LIB=P          - 1. SL.PUB.Account
                  2. SL.PUB.SYS

;LIB=S (default) - 1. SL.PUB.SYS

```

If ";LIB=G" is specified, each entry-point not found in the "RUN" filename will be searched for first in the group SL of the program; if not found there, then in PUB of the program account, and if still not found, finally in PUB.SYS. If ";LIB=P" is selected, the search will begin in the SL.PUB of the program account and then go to SL.PUB.SYS. If the default is used, only SL.PUB.SYS is searched. When an entry-point is found, the load does not look for the entry-point again. So the group SL overrides the account SL, and the account SL overrides the system SL. If the load does not find a entry-point, the load is aborted and an error message is displayed.

USES OF THE FILES:

Finally, let's examine the uses for the three files (USL, RL, and SL) in reference to subprograms [Main programs, (OUTER BLOCKS), are described earlier]. When a source subprogram has been compiled into a USL, it is free to be moved anywhere. If the routine is used by everyone, or almost everyone (For example; VIEW, IMAGE, or date routines), it should be loaded into the system SL. This will allow the routine to be automatically used by anyone. If the subroutine is used by one account, it should be loaded into the account SL, which saves space in the system SL. If a routine is only used by one group it should be loaded into the group SL. When using ";LIB=G" or ";LIB=P", it is best to use UDCs so that users do not have to key any options.

The search criteria for an entry-point can be used to replace routines for different applications or users. Or, If you

want to replace a routine that is in an SL, load it into an RL so it can be replaced at compile-time.

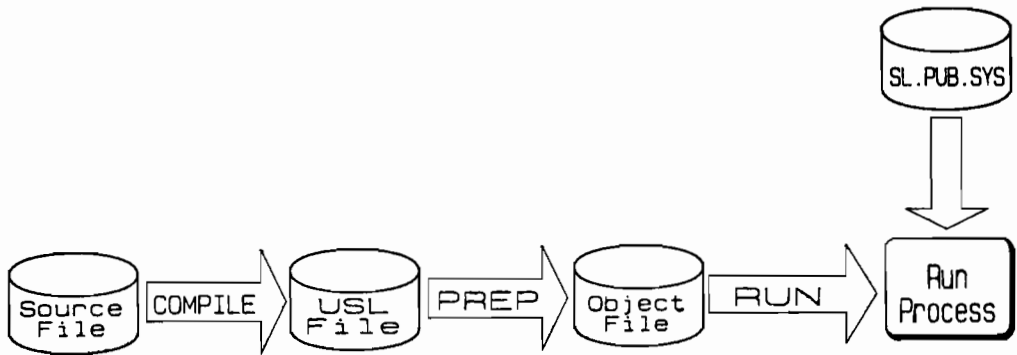
If you have a routine used by only a small percent of static programs (possibly in different groups or accounts), the routine should be loaded into an RL. Also, if a program uses an unacceptable amount of time to load, use an RL to prepare the subroutine right into the program's object code. This will increase PREP time, but decrease load time.

Highly dynamic subroutines should be stored in SLs. This will allow changes to a subroutine without having to re-compile all of the affected executable programs. The one problem with this is that the SL will be locked for a short time.

CONCLUSION

In conclusion, a segmenting scheme should be defined and followed in every installation. But before demanding a scheme at your shop, consult your SEGMENTER manual like a "LOVE LETTER" (as Alfredo puts it). Also, study your in-house and purchased software, hardware, and system configuration. And remember that segmenting in most shops is a NECESSITY, and should be treated as such.

Figure 1



MOST COMMON SITUATION

Figure 2

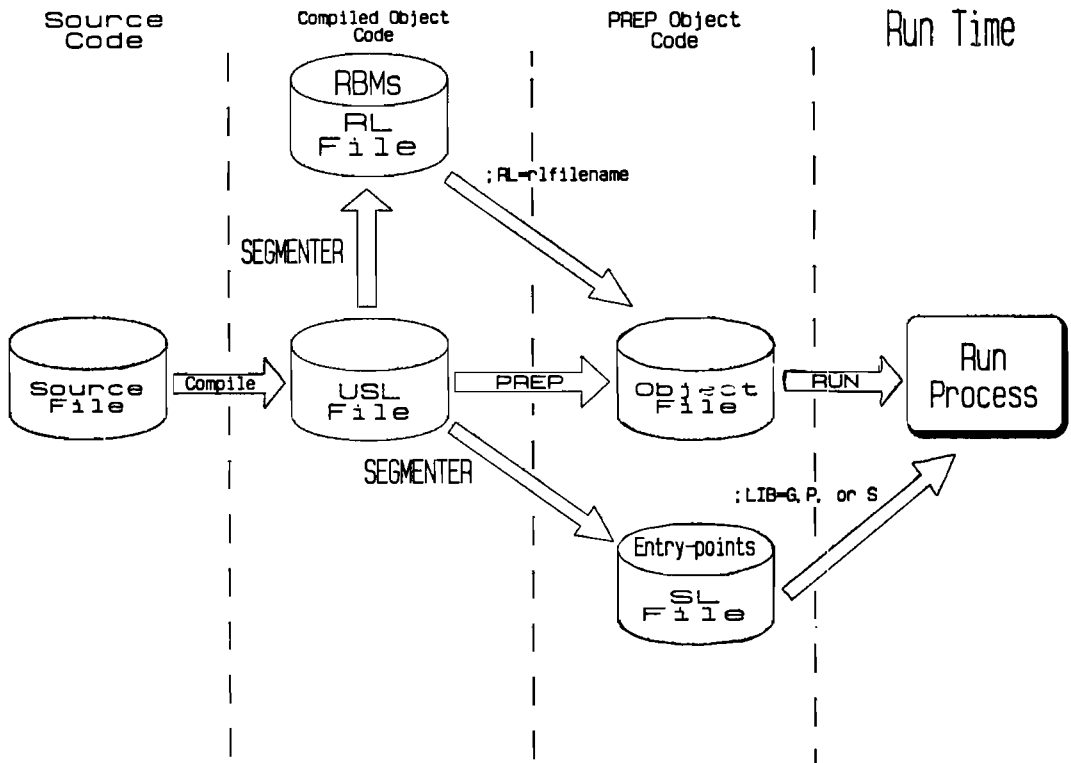


Figure 3

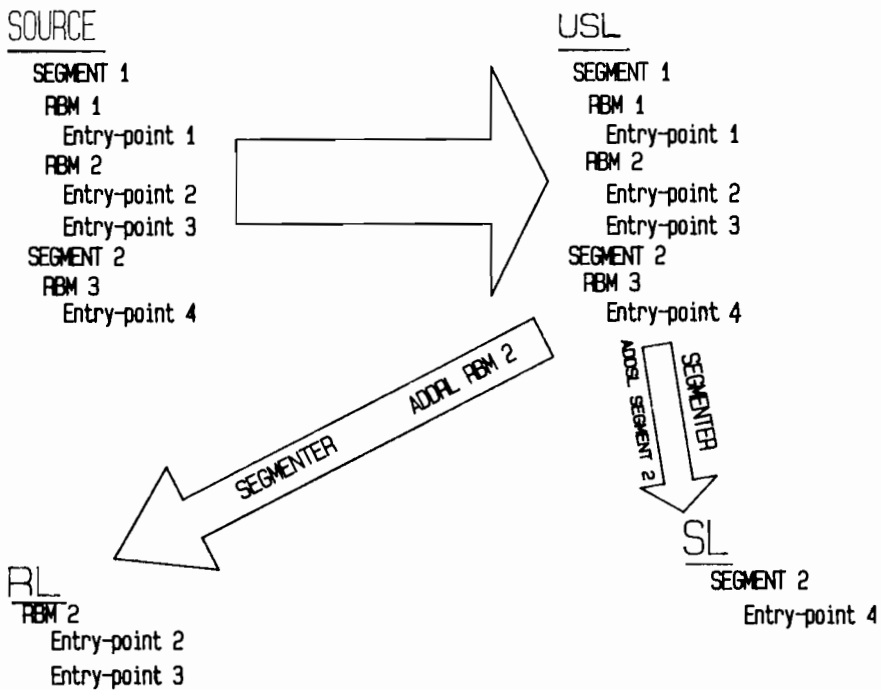


Figure 4

```
:PREP uslfile,runfile;RL=rlfilename      (To access an RL file)

:RUN runfile                               (To access SL.PUB.SYS)
:RUN runfile;LIB=S                          (To access SL.PUB.SYS)
:RUN runfile;LIB=P                          (To access SL.PUB.account
-                                           then SL.PUB.SYS)
:RUN runfile;LIB=G                          (To access SL.group.account
-                                           then SL.PUB.account
-                                           then SL.PUB.SYS)
```

'account' is 'runfile' account
'group' is 'runfile' group

3006. DICTIONARY/3000 WALKTHROUGH -- 2.0

Stephen M. Butler
 Weyerhaeuser
 NAC-2
 Tacoma, Washington 98477

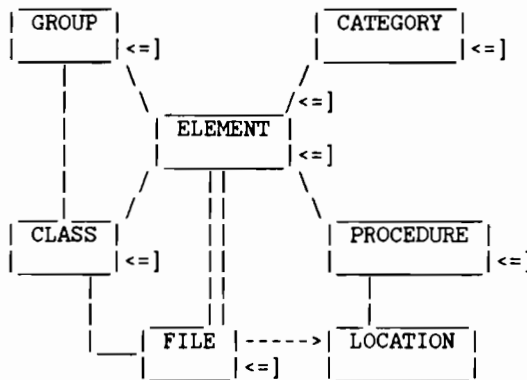
INTRODUCTION

The DICTIONARY/3000 product line consist of several database utilities and data structure generation routines. These all assist the user in maintaining databases. One of the routines enables the user to describe the data elements and sets for IMAGE. That appears to be where the majority of users stop in the utilization of DICTDBM.PUB.SYS. There is so much capability left unused.

This paper describing the internal structure of DICT.PUB (the IMAGE database maintained by DICTDBM) is an effort to encourage the exploitation of the capabilities existing in the dictionary. A previous article (1) was published in the SUPERGROUP magazine; but, the latest release of DICTIONARY has added some datasets, data elements, and more capability. Thus the article is now obsolete.

STRUCTURE OVERVIEW

DICTIONARY has seven (7) entity classifications (CATEGORY, CLASS, ELEMENT, FILE, GROUP, LOCATION, and PROCEDURE) with the relationships:



[] = ENTITY

<=] = ENTITY to ENTITY relationship

line = ENTITY to ENTITY association

FIGURE 1: ENTITY RELATION CHART

Each entity serves as a hub for information regarding the entries within that entity classification. These exist independently of the existence of any other entity. As such, there is no requirement that one entity classification (say ELEMENT) must exist before any others are created. The only requirements of this type are:

1. Two entries of like entity classification must exist before the RELATE xxx command is used. (Where xxx is the entity classification.)
2. Two entries of different entity classification must exist before the ADD xxx command is used. (Where xxx defines the two entity classifications to be associated.)
3. Some DICTDEM commands expect (or can utilize) the relations and/or associations indicated by the two preceding requirements. These commands are described later.

The ELEMENT entity serves as the hub not only for information about elements but also as the terminus for the spokes coming from the other entities. Thus it becomes the hub for the entire DICTIONARY by binding the other entities together. There are some other links used for special purposes; but none are so entwined with the entire DICTIONARY entities. COMMON STRUCTURES

There are a number of common themes running throughout the dictionary structure. The foremost is the structure of each entity. The interpretation as shown at the bottom of Figure 1 is graphically displayed as:

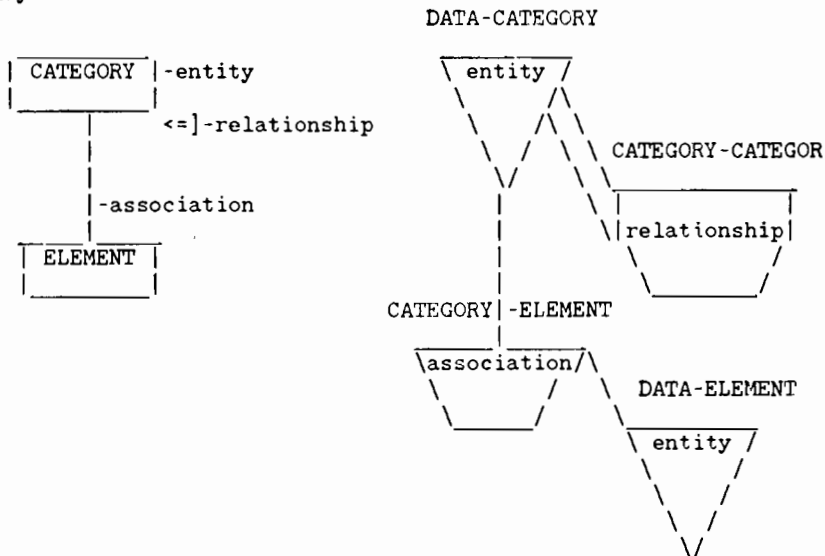


FIGURE 2: CATEGORY ENTITY STRUCTURE

For this case, the box became the manual master DATA-CATEGORY; the '<=]' became the detail CATEGORY-CATEGOR; and the line became the detail CATEGORY-ELEMENT. This same type of conversion is done for the other entities.

The key to the manual master is the entity used as the primary name. The search items for the relation detail dataset (curve line) are the entity-PARENT and entity-CHILD. The search items for the association dataset (straight line) are the names of the two entities involved.

With the exceptions noted, the following correlations between PARENT/CHILD relationship and entity/other'entity apply:

1. A CHILD may also be a PARENT except for the FILE-FILE relationship where the relationship is restricted to two levels. Entries of type VPLS and BASE may be parents while entries of type FORM (for VPLS) and AUTO, MAST, DETL (for BASE) are children.
2. An entry serving as a PARENT cannot be associated to another entity except for PROCEDURE where both PARENT and CHILD may be associated to ELEMENT or LOCATION and except for FILE where the association to CLASS or LOCATION must NOT be a CHILD.

The DICTDBM commands used to manipulate these structures are:

Manual Master of form 'DATA-entity':

- CREATE entity - brings into existence a new entry for the entity.
- RENAME entity - replaces the old name with a new name for a previously existing entry for the entity.
- MODIFY entity - allows the correction of attribute values for a previously existing entry for the entity.
- PURGE entity - gets rid of a previously existing entry for the entity. This will get rid of any other usage of the value for that particular entity.
- LIST entity - produces a display (can be on the printer) of all entity values along with some of the entry attribute values for every entry that meets the selection criteria.
- DISPLAY entity- produces a display of all attribute values and all other usages of a particular entity entry or group of entries. This command would show all information that the PURGE entity command would get rid of.

REPORT entity - lists alphabetically all ELEMENT entity entries that are associated with this entity value or to one or more of this entries children. If no element associations have been made then there is nothing to report. REPORT ELEMENT is an exception in that all ELEMENT entity values are reported. The report is alphabetical and shows the main attribute values for the ELEMENT entries.

Detail relationships of form 'entity-entity':

RELATE entity - brings into existence a new relation between two previously existing entity entries.

REORDER entity- reposition a child entity entry within the other children for the given parent.

CHANGE entity - corrects any attribute values of the relation between the two named entries of the entity.

REMOVE entity - the opposite of RELATE entity. It destroys that relation. Unlike PURGE, REMOVE gets rid of only the one entry that describes the relation between two entries of the entity.

SHOW entity - provides information about the children related to the entity value entered. If elements are associated to the parent or the children or, where applicable, any grandchildren then these are also shown. The attributes presented depend on the entity.

Detail associations of form 'entity-ELEMENT' or
'entityA-entityB':

ADD entity - brings into existence a new association between values of two different entities.

RESEQUENCE entity - reposition the element (or entity) value within the parent entry.

UPDATE entity - corrects attribute values regarding the association between the entity values of the two different entities.

- DELETE entity - similar to REMOVE except it destroys the one entry that describes the association between the values of the two different entities.
- SECURE FILE - works only with type BASE entries within the FILE entity. It provides mass ADD CLASS capability. The CLASS and ACCESS attributes, which the user is prompted for, are the same for every element in the database.
- REPORT entity - see under 'Manual master DATA-entity' above.
- SHOW entity - see under 'Detail entity-entity relationship' above.

DATASET DETAILS

The datasets have been grouped within the entity classifications of ELEMENT, FILE, CLASS, CATEGORY, GROUP, PROCEDURE and LOCATION. The association datasets (those linking two different entities) are arbitrarily placed with the entity classification as indicated by the head of the arrow in Figure 1.

ELEMENT

This consists of the three datasets DATA-ELEMENT, ELEMENT-ELEMENT, and ELEMENT-REFTYPE.

FILE

This consists of the datasets DATA-FILE, FILE-FILE, FILE-ELEMENT, FILE-EL-SECOND, LINK-FILE, FILE-PATH, LINK-ELEMENT, and FILE-SORT. These last four support the chains and sorts for a detail dataset. In addition, LINK-FILE and FILE-PATH are used by the security CLASS to restrict CLASS to a FILE and by INFORM GROUPS to identify which file the element is to come from.

CLASS

This entity is used to set up the security for IMAGE. It also records the security applied to other files/elements. Note that any security set up for IMAGE must be in place prior to running DICTDBC. DICTDBC gets its security matrix information by a reduction of the data in CLASS-ELEMENT and CLASS-FILE.

Any security arrangement that DBSCHEMA supports can be generated by DICTDBC. There are some points to remember though. You may be able to describe security arrangements that DBSCHEMA cannot support. For example: a given security class is allowed to read an item and modify (ie can generate new entries) another item. The end security is such that the Modify changes into an Update because to do otherwise would change the Read into an Update. To do what you wanted would require the use of two security class.

DICTDBC will use the more restrictive capability for a dataset for a given class. This assumes that the dataset security class is derived from the matrix of security for the elements in that dataset. DICTDBC will give precedence to the security information in CLASS-FILE if that information conflicts with the reduction of information from CLASS-ELEMENT.

DICTDBC will not include any security class for which the CLASS-PASSWORD is blank or not defined (null). Thus to temporarily remove a password from a database, one need only ensure that the CLASS-PASSWORD for the associated CLASS is blank for the duration of the DICTDBC run.

CLASS security is made up of five datasets: DATA-CLASS, CLASS-ELEMENT, CLASS-FILE, CLASS-CLASS, and CLASS-GROUP.

The last two provide the INFORM security enhancement released in the last year. Please read the DICTIONARY REFERENCE MANUAL for further understanding in that area.

CATEGORY

CATEGORY is an uncharted section of DICTIONARY as far as HP is concerned. There are many structures that the user may decide to fit within this entity. The usefulness will depend on judicious use of CATEGORY-TYPE and relationship rules of which TYPE is to be related to another in PARENT/CHILD relationships. Ideas of what and how are in the paper "DICTIONARY: Hub of Systems Development and Documentation" presented at the HPIUG ANAHIEM conference.

CATEGORY consists of the datasets DATA-CATEGORY, CATEGORY-CATEGOR, and CATEGORY-ELEMENT.

GROUP

This structure is used by INFORM. The special handling given to GROUP-ELEMENT that identifies the dataset and database for the ELEMENT within a GROUP would have been handy in CATEGORY-ELEMENT and PROCEDURE-ELEMEN. It is best not to include anything in this structure that is not for INFORM usage.

The structure contains the datasets DATA-GROUP, GROUP-GROUP, and GROUP-ELEMENT.

PROCEDURE

This is the area of DICTIONARY where the systems, jobstreams, programs, subprograms, callable routines, and other information regarding the modules of the implemented system is to be recorded. Relations to the ELEMENTS used are also recorded. While there is no direct link to find what files are used, a well designed Data Structure will minimize the duplication of data so that every place the ELEMENT resides will need to be investigated for impact by changes to the module (vice versus, also). An alternate technique is mentioned in "DICTIONARY: Hub of Systems Development and Documentation".

PROCEDURE consists of these datasets: DATA-PROCEDURE, PROCEDURE-PROCED, and PROCEDURE-ELEMEN.

LOCATION

This defines the account structure location for FILE and PROCEDURE entries. Currently HP does not utilize information stored here. Hopefully products like INFORM can be made smart enough to look here and automatically issue file equates (or append the location information on the FOPEN) for the FILES they open.

It is regrettable that DATA-LOCATION does not have a LOCATION-TYPE. If that were the case, then the LOCATIONS could be grouped according to SOURCE (SRCE), OBJECT (OBJT), USL, TEST (database test group), etc. Since this is not the case, the user should set up a standard naming convention for the LOCATIONS.

The datasets are: DATA-LOCATION, FILE-LOCATION, PROCEDURE-LOCATION.

OTHER

There are five (5) other datasets within DICTIONARY. These fall into three groups. The first group supports INFORM recall of where previously stored INFORM programs are located and what their descriptions are. It takes two datasets (DATA-REPORTLOC, AUTO; and REPORT-LIST, DETL).

The second group is the repository of all the description text entered for any entity, relationship or association. This also takes two datasets (LINK-DESCRIPTION, AUTO; and DESCRIPTION-TEXT, DETL).

The third group is the dataset DIC-CONTROL. This has control information to keep all the internal linkages going strong and healthy. This dataset along with the internal linkages are the reason that two or more DICTIONARY databases cannot be merged. You must not even think of using DICTDBL to load a DICTIONARY (unless you abide strictly by the method HP uses).

SUMMARY

DICTIONARY is a complex database and piece of software. There are linkages that the software maintains which, when damaged, may reduce your data to meaningless gibberish. On the other hand, there are a lot of things that can be done with DICTIONARY that most users currently aren't even trying. This should give you an idea of what can be exploited.

Those who want to load the DICTIONARY definitions (as HP supplies them) should do so into a newly created DICTIONARY. This is done by first running DICTINIT to INITIALIZE a new dictionary. At the UPDATE run (DICTINIT.PUB.SYS,UPDATE) do a REINITIALIZATION (R). Proceed as if doing the initialization until it asks whether the store file is on Disk or TAPE. Go to another terminal and unload (using DICTDBU.PUB.SYS) the database MDIC (in logon group and account) into a store file called MDSCH. Then tell DICTINIT (original terminal) 'D' for disk and let the process proceed as usual. When done, you will have a brand new DICTIONARY that contains a fully accurate discription of itself. Now, if only HP would enter descriptions.

BIBLIOGRAPHY

Butler, S. M., "DICTIONARY/3000 Walkthru", SUPERGROUP, May/June 1984, page 18ff.

Butler, S. M., "DICTIONARY: Hub of Systems Development and Documentation", PROCEEDINGS of the HPIUG ANAHEIM Conference, 1984, page 36-1 to 36-15.

Hewlett-Packard, DICTIONARY REFERENCE MANUAL.

APPENDIX

INTERNAL FIELD DESCRIPTION BY ENTITY

LEGEND

Within several of the sets, there are three groups of items that are repeatedly used. We will define them once and attach a label to them:

DCI -- whenever this is used, read:

DATE-CREATE	U6	- date entry was created
DATE-CHANGE	U6	- date entry was modified
IDENTITY-CREATE	U8	- system user who created entry
IDENTITY-CHANGE	U8	- system user who modified entry

DCID -- whenever used, read DCI plus:

DESCRIPTION-KEY	I2	- key number of description for this entry (no description when zero) This forms the implicit path to the DESCRIPTION-TEXT dataset.
-----------------	----	--

DCIP -- whenever used, read DCID plus:

POSITION	K2	- used to maintain sorted sequence within the chain. For the relationship between like entities, this chain is on the entity-PARENT item. For associations between dislike entities, the chain is generally on the item name for the entity pointed to in Figure 1. With the exception of DESCRIPTION-TEXT, no sorted chain is the primary chain (in all cases it should be).
----------	----	---

ELEMENT

DATA-ELEMENT MAST

ELEMENT	U20	- Primary name for the element. This is used to link all the information together regarding the item. This is the KEY item.
---------	-----	---

ELEMENT-NAME U50 - Long descriptive name for this item. This is not the first line of, nor the replacement for, the description.

ELEMENT-TYPE U2 - Format type of the item. The first position is used for the type and the second to indicate if negative values allowed. NOTE: Type '9' is by definition non-negative.

ELEMENT-SIZE I1 - The external position count of the item. The count includes the '.' if any; but not the '+'/'-' sign. For VPLS, this size must be exactly the same as the field size in FORMSPEC minus 1 for the sign position--if applicable.

ELEMENT-DEC I1 - The number of digits to the right of the decimal. If this is non-zero, then the ELEMENT-SIZE is one larger to include the position taken by the '.'. I don't know why--it does seem odd that changing a field from '9999' to '999.9' (same number of total digits) is done by I(4,0)--->I(5,1).

ELEMENT-LENGTH I1 - Number of bytes of internal storage needed to hold the value. DICTDBM makes the standard COBOL defaults but rounding up to a word boundary for IMAGE. This can be overridden to indicate a one byte field, I(1,0,1) or to indicate that the one word integer is to be allowed up to 32767, I(5,0,2).

ELEMENT-COUNT I1 - Equivalent to IMAGE sub-item count. Gives the number of units type(size,decimal,length) contained in ELEMENT.

ELEMENT-UNITS X10 - Unit of measure for the value attribute. The user community needs to standardize the usage for this as HP has not. Any element whose value constitutes some 'measure' based on some unit should have that base unit described here. e.g., US\$M for \$1000 US money.

- ELEMENT-RESP X20 - This can be either the person who is responsible for maintaining the correct information in DICTIONARY regarding this item; or, and I prefer this one, the name of the owner of the data, the one who must keep the data valid, the end user responsible for the values that the item contains. This doesn't need to be a name of a person as much as a means to identify who has the last say. It may be a department, position title, or some other means of identifying the 'responsible party'.
- ELEMENT-HEADING X30 - Used by the RAPID products (at compilation time--if item not defined in either program or DICTIONARY, then used at run time) to provide the default heading for this item. It would be a good documentation tool for other languages in that the programmer could check here when coding his/her program. If not supplied the default is the item name contained in ELEMENT.
- ELEMENT-ENTRY X30 - The default prompt string (RAPID products) to use when prompting the user for a value to place into the item's storage location. If not supplied then the default is the item name contained in ELEMENT.
- ELEMENT-EDIT X30 - The default editing to be done by the RAPID products when DISPLAYing a value from this item. If not supplied then the defaults are left justified, zero suppress (on numerics).
- NOTE: REPORT and TRANSACT have provisions to override the entries in DICTIONARY.
- DCID - See legend at beginning of appendix.
- ELEMENT-SIGN X2 - Flags the SIGN SEPERATE information.
- ELEMENT-BLANK I1 - Flags whether element should be blank when ZERO (for numeric items).
- ELEMENT-JUST I1 - Flags right justification.

ELEMENT-SYNC I1 - Flags synchronization for numeric integer fields.

NOTE: These last four are primarily used when generating PASCAL and COBOL data definitions.

ELEMENT-ELEMENT DETL

ELEMENT-PARENT U20 - Item name whose storage space is being redefined by other items. A COBOL group level if you please. This is a search item chained from DATA-ELEMENT. This chain is sorted by position.

ELEMENT-CHILD U20 - Item name doing the redefinition of the storage space for the ELEMENT-PARENT. This is also a search item chained from DATA-ELEMENT. It is the primary chain and is not sorted.

ELEMENT-POSITION I1 - Location (byte offset--as in SPL) that begins ELEMENT-CHILD within ELEMENT-PARENT. This is needed for the RAPID products. At last check COBRA (from IMACS) ignored this and depended on the parent/child relationships to be in order and non-redundant. Certainly, proper use of DICTIONARY definition for the parent/child relationship would negate the need for this attribute. Nevertheless, it is here and the RAPID products depend on it. A nice compromise might be a tool to run after changing DICTIONARY that would verify (and correct) the ELEMENT-POSITION attribute.

DCIP - See legend at beginning of appendix.

ELEMENT-ALIAS-E X30 - Now you can specify an alias for the CHILD element.

ELEMENT-REFTYPE DETL

ELEMENT X20 - Similar to ELEMENT-PARENT above.

ELEMENT-RTYPE X20 - Similar to ELEMENT-CHILD above.

FILE

DATA-FILE MAST

FILE U20 - Primary name of the file. This is used to link all information regarding this file together. This is the KEY item.

FILE-NAME U50 - Long descriptive name for the file. This is not part of the description--it is a name.

FILE-TYPE U4 - One of the HP defined file types. Note below that only certain file types may have children.

FILE-RESP X20 - Similar to ELEMENT-RESP under ELEMENT above.

DCID ~ See legend at beginning of appendix.

FILE-REC-FORMAT I1 - Not sure what this and the following fields are being used for. Recorded here for completeness.

FILE-DATA-TYPE I1 - See above.

FILE-REC-MODE I1 - (Probably recording mode; but then...)

FILE-REC-SIZE 2I2 - Primary and alternate record sizes.

FILE-BK-FACTOR 3I1 - Three blocking factor values. (But the DICTIONARY REF. MNL. told you that much.)

FILE-DEVICE X8 - Device class name.

FILE-DEV-CLASS I1 - LDEV number or ???

FILE-CCTL I1 - CCTL to be used or not for file.

FILE-FILE DETL

FILE-PARENT U20 - Name of a file of type BASE or VPLS (at one time GRPH was in a pre-release version).

FILE-CHILD U20 - Name of a file of type MAST, AUTO or DETL if the parent is BASE; or, FORM if the parent is VPLS (at one time

CHRT was allowed if the parent was GRPH).

- FILE-ALIAS-F U16 - Local alias name. This will be the real name of the dataset or form. It is the alias for the child. The RAPID products will internally use the FILE primary name but will substitute the alias when calling IMAGE or VPLS. When left blank, the default becomes the primary name. IMAGE uses only the first 16 characters. I don't know if RAPID products correctly truncate the name to 15 characters for VPLS or not.
- FILE-SIZE I2 - Used only if the parent is of type BASE. This is the capacity of the dataset.
- FILE-BLOCK I1 - Used only if the parent is of type BASE. This is the word size of the block to be used for IMAGE. The DBSCHEMA equivalent is \$CONTROL BLOCKMAX. The value must conform to IMAGE restrictions for BLOCKMAX. A BLOCKMAX can be specified for each dataset and they do not have to be identical. When IMAGE carves out the area for the buffers, it will use the largest BLOCKMAX specified. The smaller buffers will not be repacked. For main memory storage, IMAGE puts each block no matter what the size into the area reserved for the largest block.
- DCIP - See legend at beginning of appendix.
- FILE-ELEMENT DETL
- FILE U20 - Primary name of the file of type MPEF, KSAM, FORM, AUTO, MAST, or DETL. This is a search item for the dataset and is chained to DATA-FILE (sorted by POSITION). It is not the primary path.
- ELEMENT U20 - Primary name of the element to be associated with the file. DICTDBM allows CHILD elements to be associated but the other RAPID products WILL NOT allow that

construct. Therefore, it is best that ELEMENT also be the PARENT not a CHILD. This is the primary chain, unsorted, and linked to DATA-ELEMENT.

- ELEMENT-ALIAS U20 - The local alias for this element within this file. The RAPID products will use the primary name (ELEMENT) within the programs but any calls that require the name of the element being passed to VPLS or IMAGE will result in this alias name being passed. The default is ELEMENT. There have been bugs with DICTDBC and other RAPID products not truncating this to the correct length for IMAGE or VPLS. It is best to keep the primary name within the constraints also.
- FILE-KEY I2 - This has meaning only if FILE is a dataset or KSAM file. A '-1' means the element is the key (MAST and AUTO) or one of the keys (KSAM) while '0' means this is not a key, not a search item, just one of the place holders. A '+n' value means that ELEMENT is a search item (DETL file only) and FILE-KEY is the id number via FILE-PATH (and LINK-FILE) of the master dataset being chained to (see below for FILE-PATH and LINK-FILE). This is one of the implicit paths found in DICIONARY/3000.
- ELEMENT-KEY I2 - This has meaning only if FILE is type DETL. It is used only when FILE-KEY is '+n'. A '0' is default in the above cases. When FILE-KEY is '+n' then a '0' means the chain is not sorted. A '+m' is the id of the element within this file that sorts the chain. Another implicit path to LINK-ELEMENT and FILE-SORT. One last restriction: the element indicated by ELEMENT-KEY cannot be the same name as ELEMENT otherwise you would be sorting by the very element that determines the chain. In that case, the element serving as the sort would be identical for every entry in the chain--which is why we have chains to begin with.

ELEMENT-PRIMARY I1 - This has meaning only if FILE is type DETL and FILE-KEY is '+n'. This indicates which of the chains should serve as the primary chain ('+1'). All other cases have '0'. It is possible to have no primary path indicated. In that case IMAGE will pick a path according to its published algorithm. It appears that the paths in DICTIONARY were picked in that manner as most primary paths are on the chain that has the single entry rather than on the chain that could benefit during a reload of the database.

DCIP - See legend at beginning of appendix.

FILE-POSITION I1 - Have not deduced this one yet. Suspect it is related to KSAM, and MPE files.

KEY-DUPLICATES I1 - Duplicates allowed or not for KSAM.

FILE-FIELD-NO I1 - VPLUS field number within FORM screen.

FILE-EL-SECOND DETL

FILE U20 - File name as for above file. Sorted by POSITION.

ELEMENT U20 - Element name. Primary path.

DCIP - See legend at beginning of appendix.

LINK-FILE AUTO

FILE-KEY I2 - The key item (only item--that's the meaning of AUTO) that is the terminus of those implicit paths leading to FILE-PATH--see next. There are four (4) of these implicit paths--those labeled A in figure 4; ie, FILE-ELEMENT (noted above) and DATA-CLASS each refer once, GROUP-ELEMENT makes two references (FILE-KEY and FILE-PARENT-KEY).

FILE-PATH DETL

FILE U20 - The file name that serves as the IMAGE chain head for FILE-ELEMENT entries. It has other functions for DATA-CLASS and GROUP-ELEMENT (look at those dataset walk throughs). This is the primary path and is linked to DATA-FILE (to ensure that the file stays around while it is referred to). This chain may have more than one entry but is not sorted.

FILE-KEY I2 - AHA! This is the entry that's being pointed to by all those FILE-KEY implicit paths. This gives the relationship to the FILE. This item is linked to LINK-FILE. There should be only one entry per chain.

LINK-ELEMENT AUTO

ELEMENT-KEY I2 - The key item that links to FILE-SORT. See next. There is only one implicit path (from FILE-ELEMENT) to this.

FILE-SORT DETL

ELEMENT U20 - The name of the element that serves as the sort item for the related ELEMENT-KEY. This is the primary path and is linked to DATA-ELEMENT.

ELEMENT-KEY I2 - This is linked to LINK-ELEMENT and is the terminus of the implicit path from FILE-ELEMENT.

CLASS

DATA-CLASS MAST

CLASS I1 - A non-negative number indicating the class level (0-9999). Note that DICTINIT will set up negative class numbers corresponding to the passwords entered for the different types of dictionary users. The password is encrypted using a very trivial encryption method. DICTDBM uses the negative classes to find out what level of access the user is

allowed to have; MANAGER through REPORT.

CLASS-NAME	U50	- Long name to identify externally this class. Note that DICTIONARY uses CLASS to maintain the linkage.
CLASS-TYPE	U4	- Set up for the user to organize the the security classes. The LIST CLASS command in DICTIONARY will allow the user to restrict the classes listed to those of a certain CLASS-TYPE. This is completely managed by the user and the values do not affect the function of any DICTIONARY/RAPID routines.
CLASS-PASSWORD	X8	- The eight character password or file lockword. If this is blank or null then DICTDBC will not include it in the generation of a database. For negative value CLASS this field is encrypted by treating each two bytes as a single word integer. Each of the four integers are then negated.
CLASS-RESP	X20	- The person or unit that is to maintain the information regarding this security class.
FILE-KEY	I2	- An implicit path to LINK-FILE and thence to FILE-PATH. If zero then there is no pathway. This is used to record the response to which file this class is to be restricted--if any. DICTDBC will check that the FILE pointed to is the same as the BASE being created. If not the same then the security class is removed from the schema being generated. Note that CLASS could be associated with an ELEMENT that does not belong to the restricted file (which must be a BASE). CLASS may even be associated with a FILE other than the one being restricted to. DICTDBM does not check for those occurrences. DICTDBC will check while the schema is being generated.
DCID		- See legend at beginning of appendix.
CLASS-CLASS	DETL	

CLASS-PARENT I1 - Linked to DATA-CLASS and sorted by POSITION. Relates the IMAGE class to the appropriate INFORM classes.

CLASS-CHILD I1 - Linked to DATA-CLASS and is the Primary path.

DCIP - See legend at beginning of appendix.

CLASS-ELEMENT DETL

CLASS I1 - The security class for this association. This is sorted by POSITION and is linked to DATA-CLASS. The chain is not primary.

ELEMENT U20 - The element for this association. This is the primary chain and is linked to DATA-ELEMENT.

ELEMENT-ACCESS U2 - The access capability for the security class against the element. R-Read, U-Update existing, M-Modify(Read/Update/Write), X-Null. For understanding of the null list read the IMAGE REFERENCE MANUAL. Note that U implies Read at set and Write at item levels.

DCIP - See legend at beginning of appendix.

CLASS-FILE DETL

CLASS I1 - Security class for this association. Sorted by POSITION, linked to DATA-CLASS and is not the primary chain. There is no command to resequence the FILE within a class. Since that is the case, I don't know why they bother to sort by POSITION.

FILE U20 - File for this association. Unsorted, primary chain, linked to DATA-FILE

FILE-ACCESS U2 - Access capability for the security class against this file. R-Read, W-Write, X-Null. DICTDBC will use the access capabilities defined in CLASS-FILE to override the matrix developed from the access capabilities of the ELEMENTs within the FILE.

DCIP - See legend at beginning of appendix.

CLASS-GROUP DETL

CLASS I1 - Linked to DATA-CLASS and sorted by POSITION.

GROUP U20 - Linked to DATA-GROUP and is the Primary path. This ties an INFORM GROUP to and INFORM CLASS

DCIP - See legend at beginning of appendix.

CATEGORY

DATA-CATEGORY MAST

CATEGORY U20 - Primary name for the entry. This will link the information together for this CATEGORY.

CATEGORY-NAME U50 - Long descriptive name for the category. This is not part of the description.

CATEGORY-TYPE U4 - User definable TYPE to classify the CATEGORY according to some user chosen scheme. The above cited article gives some ideas.

CATEGORY-RESP X20 - Who or what is to maintain the information regarding this CATEGORY.

DCID - See legend at beginning of appendix.

CATEGORY-CATEGORY DETL

CATEGORY-PARENT U20 - CATEGORY entry that will serve as the PARENT for this relationship. This chain is sorted by POSITION, linked to DATA-CATEGORY; but is not the primary chain.

CATEGORY-CHILD U20 - CATEGORY entry that will serve as the CHILD for this relationship. This chain is unsorted, linked to DATA-CATEGORY and is the primary.

DCIP - See legend at beginning of appendix.

CATEGORY-ELEMENT DETL

CATEGORY U20 - CATEGORY entry that is associated to the ELEMENT. This chain is sorted by POSITION, linked to DATA-CATEGORY; but is not the primary chain.

ELEMENT U20 - ELEMENT entry that is associated to the CATEGORY. This chain is unsorted, linked to DATA-ELEMENT and is the primary chain.

ELEMENT-ALIAS-C U60 - The local alias for the ELEMENT when used within CATEGORY.

DCIP - See legend at beginning of appendix.

GROUP

DATA-GROUP MAST

GROUP U20 - Primary name of an INFORM GROUP. This group may either have children or be associated to elements. GROUP is the link that binds all the information together.

GROUP-NAME U50 - Long name for the GROUP. This is not part of the description but should be descriptive. Note that this information is not displayed to the INFORM user.

GROUP-TYPE U4 - Up to the user to define. It would be best to set up a standard classification so the DBA person can select entries base on TYPE.

GROUP-RESP X20 - Who/what maintains the group information. This might be the name of the end user for whom the group was put in place. That is the person who ultimately controls how the information is to be shown by INFORM.

DCID - See legend at beginning of appendix.

GROUP-GROUP DETL

GROUP-PARENT U20 - GROUP entry that serves as the PARENT. For those first line INFORM GROUPS the parent is \$MENU. \$MENU is the GROUP that INFORM starts with to present a menu of GROUPS to the user. The chain is sorted by POSITION, linked to DATA-GROUP; but is not the primary even though its average length is longer than the chain by GROUP-CHILD.

GROUP-CHILD U20 - The CHILD group for this relationship. This is a primary name and is the name that INFORM presents to the user on the menu of INFORM GROUPS. This chain is unsorted, primary, and linked to DATA-GROUP.

DCIP - See legend at beginning of appendix.

GROUP-ELEMENT DETL

GROUP U20 - Group entry that is to contain elements. The chain is sorted by POSITION, not primary, and linked to DATA-GROUP.

ELEMENT U20 - Primary element name to be in this INFORM GROUP. This is the primary chain (even though it is by far shorter than the previous) linked to DATA-ELEMENT. It is not sorted.

ELEMENT-ALIAS U20 - The local name for the element. This field will be shown to the user. If left blank then ELEMENT is shown. By specifying the alias, the DBA person makes INFORM more user friendly. INFORM still uses the primary element name to locate its file alias (stored in FILE-ELEMENT).

FILE-KEY I2 - Implicit path pointer to FILE-PATH (via LINK-FILE) that gives the file name the element is to be retrieved from. If zero then the DBA has not specified a file and INFORM is free to choose the best file to get the information from.

FILE-PARENT-KEY I2 - Implicit path pointer to FILE-PATH (via LINK-FILE) that gives the base name that the file (FILE-KEY

immediately above) is a dataset within. The user is asked for the base name only if the dataset is in more than one base. If FILE-KEY was not a dataset then this is left as 0 (also if FILE-KEY is 0). Note that INFORM uses a reduction algorithm to deduce ELEMENT/FILE pairs for those elements selected. If the FILE-KEY is not zero then this reduction is limited to pairing this element with the indicated file. Otherwise INFORM looks at the FILE-ELEMENT dataset using the ELEMENT chain. The order of information in that chain (not controllable by the DBA person) will influence the files that INFORM will use to retrieve the data. This is a problem only if the ELEMENT name is used in more than one dataset with different meanings or containing different information.

- LINK-VALUE I1 - Value indicating the linkage potential this element has to tie information together. A -1 indicates no linkage whatsoever, a 0 means use as a link only in case of last resort. The values 1..n are an ordered list (1 highest) of the linkages to use. If this LINK-VALUE is a 3 then it will be used only after elements that have a link value of 1 or 2 have been exhausted. All positive link valued elements within the GROUP are evaluated (DICTIONARY p. D-8).
- ELEMENT-DISPLAY I1 - Value of -1 means don't display (then it should have a positive link value or its presence does nothing). Value of 0 means display this entry.
- DCIP - See legend at beginning of appendix.

PROCEDURE

DATA-PROCEDURE MAST

- PROCEDURE U20 - Primary name of the procedure or module. This is the key item that

links the information regarding this procedure together.

- PROCEDURE-LANG U10 - User standardized name for the language the procedure source is written in. The user needs to define and enforce his/her own standard responses.
- PROCEDURE-NAME U50 - Long descriptive NAME for the PROCEDURE. This line is not part of the description.
- PROCEDURE-RESP X20 - Who is responsible for the code. This might be the name of the programmer who is to maintain the code, or it might be the name of the user who must coordinate change requests for it.
- PROCEDURE-TYPE U4 - User standardized code for the TYPE of module being recorded in PROCEDURE. The user needs to define and enforce his/her own standards for this field.
- DCID - See legend at beginning of appendix.

PROCEDURE-PROCED DETL

- PROCEDURE-PARENT U20 - Name of PROCEDURE that is the parent in this relationship. This field is sorted by POSITION (ie, the CHILDREN within PARENT are ordered by POSITION), is linked to DATA-PROCEDURE but, unfortunately, is not the primary chain.
- PROCEDURE-CHILD U20 - Name of PROCEDURE that is the child in this relationship. This field is the primary chain (generally only one entry in the chain--so?), is linked to DATA-PROCEDURE and is unsorted.
- DCIP - See legend at beginning of appendix.

PROCEDURE-ELEMEN DETL

- PROCEDURE U20 - Name of PROCEDURE that is associated to the named ELEMENT. This field is sorted by POSITION (to allow the ELEMENTS to be ordered within PROCEDURE), linked to DATA-PROCEDURE;

but is not the primary chain (even though performance wise it should be).

- ELEMENT U20 - Name of ELEMENT that is associated to the PROCEDURE. This field is unsorted, linked to DATA-ELEMENT, and is the primary chain.
- ELEMENT-ALIAS-P U30 - Local name within PROCEDURE for ELEMENT. The RAPID products (INFORM, REPORT, TRANSACT) do not make use of this field. Hopefully, COBOL and PASCAL via HPTOOLSET will.
- DCIP - See legend at beginning of appendix.

LOCATION

DATA-LOCATION MAST

- LOCATION U20 - Primary name for a location. This is the key that links the FILE and PROCEDURE information to where they are located.
- LOCATION-NAME U50 - Long descriptive name for the LOCATION. This is not a part of the description.
- LOCATION-GROUP U8 - The MPE GROUP name that the location defines. This is site/cpu specific information.
- LOCATION-ACCOUNT U8 - The MPE ACCOUNT name that the location defines. This too is site/cpu specific.
- LOCATION-CPU U8 - The CPU name (or other ID) that the location defines. Early on (circa 1981) I hear rumors that HP had plans for some sort of distributed network support for RFA (including database) via the DICTIONARY LOCATION information. I heard it only once--but it sure sounded like an interesting concept.
- DCID - See legend at beginning of appendix.

FILE-LOCATION DETL

LOCATION U20 - Name of LOCATION for this association. This search item is sorted by POSITION (so that FILE may be ordered within a POSITION), linked to DATA-LOCATION, but is not the primary chain. There is no command to resequence the FILES within a LOCATION. With this being the case, I am at a loss why there is a sort on POSITION (if fact why position is used at all).

FILE U20 - Name of FILE assigned to this LOCATION. This search item is unsorted, primary, and linked to DATA-FILE. FILES of type DETL, MAST, AUTO, and FORM cannot be associated with a LOCATION.

FILE-ALIAS U8 - The local name (for this LOCATION) of the file. It defaults to the first 8 characters of FILE. A database located in different LOCATIONS could have a different name (alias) at each location.

FILE-SIZE I2 - Asked only if the FILE is of TYPE MPEF/KSAM. This is the number of records to be within the file. Default is 0.

DCIP - See legend at beginning of appendix.

PROCEDURE-LOCATI DETL

LOCATION U20 - LOCATION name where the below named PROCEDURE is located. This is a search key (non-primary) sorted by POSITION and linked to DATA-LOCATION. There is no command to resequence the PROCEDURES within a LOCATION. With this being the case, I am at a loss why there is a sort on POSITION (if fact why position is used at all).

PROCEDURE U20 - Primary name of the PROCEDURE to be associated with this location. This is the primary search key (unsorted path) linked to DATA-PROCEDURE.

PROCEDURE-ALIAS U8 - Local file name for the PROCEDURE when at this location. This defaults to the first 8 bytes of PROCEDURE.

INFORM REPORT CATALOG

DATA-REPORTLOC AUTO

REPORT-LOC U20 - Contains the group.account the INFORM programs are stored in. INFORM gets this directly from the file system at the time a program is saved. Future users getting a catalog are asked to time a computer group name. This is that name. It is the group name followed by a "." followed by the account name with no embedded blanks. INFORM is smart enough to detect if no account was specified and to default to the logon account. Also if no group is specified it will default to the logon group.

REPORT-LIST DETL

REPORT-LOC U20 - Uhm. Yes, its the guy from above. This will link all the report names located in a common group.account.

REPORT U6 - The report name (remember INFORM puts an 'II' in front for the actual code file name.

REPORT-NAME U50 - This is the long descriptive name for the report. No other description is allowed.

DATE-CREATE U6 - Date this report was created within INFORM and saved (added to the catalog).

IDENTITY-CREATE U8 - User name (MPE) of person who created the report entry. INFORM will compare this with the current user's name and call it 'USER' if the names match.

DESCRIPTION REPOSITORY

LINK-DESCRIPTION AUTO

DESCRIPTION-KEY I2 - The terminus of all the description implicit paths--datasets that have DCIP or DCID (labeled C in figure 4). This forms an IMAGE link with DESCRIPTION-TEXT.

DESCRIPTION-TEXT DETL

DESCRIPTION-KEY I2 - The linkage sorted by POSITION.

DESCRIPTION-LINE X50 - Lines upon lines of description. Each line is 50 characters long. There is no practical limit (32 bits logical for the line number-- see POSITION) as to the number of lines for a given DESCRIPTION-KEY.

POSITION K2 - The line counter (logical so IMAGE will sort it; 2 words so that there is no practical limit--32 bits logical means the sign bit is considered part of the number and not a sign). This is normally incremented by 1000 (as are the editors' line numbers) so that lines may be inserted at a later time.

CONTROL

DIC-CONTROL MAST

CONTROL-KEY I1 - This is the id for the control record within DIC-CONTROL. So far there is only one record within the dataset. This value is '0'.

DESCRIPTION-KEY I2 - No, there is no description associated with the control record. This contains the value of the last DESCRIPTION-KEY assigned to the corresponding implicit path. Most problems with the DICTIONARY description system getting fouled up include a DICTDBL merge from another DICTIONARY or some other means

whereby this field is set to some value much lower than current usage indicates; which, for DICTIONARY, is a fate worse than death.

- FILE-KEY I2 - Last FILE-KEY value assigned to the corresponding implicit path. This may have similar problems to the description implicit paths if DICTDBL or QUERY is used.
- ELEMENT-KEY I2 - Last ELEMENT-KEY value assigned to the corresponding implicit path. Similar care must be taken as to that with DESCRIPTION-KEY.
- DCI - See legend at beginning of appendix.

3007. GROWTH ORIENTED ARCHITECTURE
FOR
SUCCESSFUL MULTI-VENDOR NETWORKING

Dennis McGinn
Group Marketing Manager
Information Networks Group
HEWLETT-PACKARD



INTRODUCTION

As computer manufacturers rush to furnish methods of networking their products to each other and other vendor's systems, it becomes critically important for both vendors and buyers to establish networking strategies based on standard protocols.

For those companies who have tried to network computers in their different departments, there remain problem areas in implementing efficient networks and thus reaping the benefits of connectivity and flexibility among multi-vendor systems.

By taking advantage of existing standards as well as emerging networking protocols, computer manufacturers can provide flexible capabilities required for this decade and years to come, while avoiding haphazard inter-connect schemes.

In October of 1981, Hewlett-Packard announced its support for interconnection of multi-vendor computer systems through compatibility with the Open Systems Interconnection (OSI) Reference Model of the International Standards Organization (ISO).

Given today's diverse communications environment and tomorrow's expected changes, the task of developing a comprehensive networking strategy can be extremely challenging both for users and vendors.

TODAY'S MULTI-VENDOR ENVIRONMENT

Today's typical customer desires to interconnect systems manufactured by different vendors to support various applications on a variety of links.

The communications software required to meet today's needs is quite complex; tomorrow it is likely to be even more so. Perhaps the main reason for increased complexity anticipated for the future is the need to support both "old" and "new" protocols. "Old" protocols here refers to those currently in use, be they proprietary, de facto standards (e.g., SNO), or international standards (e.g., X.25). "New" protocols refers primarily to the emerging industry, national, and international standards, as well

as to any proprietary protocols that customers or vendors might be inventing.

Often articles on protocols seem to imply that somehow old protocols will merely vanish when the new standards do finally emerge. Unfortunately, that is an unlikely -- if not mythical -- scenario. "Old" software neither dies nor fades away. Some customers will not buy new hardware or software unless it is "backward compatible" with the equipment they already own. And they do not always believe that new is necessarily better. This points to the conclusion that to accommodate future growth and changes, users should define a networking strategy that will be based on

an architecture that will allow them to support both their current protocols and new protocols that might evolve.

Islands of Automation

The early evolution of computers led to computerization at the departmental level of typical Fortune 1000 companies. With the occurrence of a typical 30% per year reduction in the price of hardware, departments were able to purchase more of their own customized systems. Joining these systems to share information then required the installation of a network. The proprietary networking schemes met the requirements of the 70's, but because they linked only computers from a single vendor (and often only a specific model) this created "islands of automation." The problem was made more severe by the proliferation of personal computers as productivity tools for small workgroups. Within the organization, the risk of system incompatibility and disintegration became significant. These proprietary networking schemes developed in the 70's created problems for the 80's. Lack of integration between departments became a barrier to further productivity gains. In the 80's, the new strategic requirement is to join these islands of automation to enhance organizational productivity. Any realistic strategy for a major corporation must therefore provide for efficient multi-vendor networking.

A successful networking strategy is one that takes into consideration a company's long-term business strategy and installed base of equipment, and then identifies a networking architecture that can grow with the company, adapting to changes in technology.

ARCHITECTURE

Let us discuss in more detail the objectives of a growth-oriented networking architecture:

OSI Reference Model

This seven-layer model is a framework for all the standards that are being developed, therefore, a cornerstone for all the other objectives. (See Figure 1.)

Flexibility

Today's networks utilize a wide variety of protocols, which will remain the case for the foreseeable future. Implementations based on Xerox XNS, DARPA TCP/IP, Ethernet and X.25 exist now, with newer protocols like 802.3, 802.4 being added. 802.5 and ISDN are envisioned for the future. Different protocols serve different environments. Some protocols provide comprehensive services while others provide performance. This implies that the ideal networking architecture will need to support various protocols at each layer. One monolithic implementation of a specific set of protocols will not satisfy this need (See Figure 2). protocols will not satisfy this need (See Figure 2).

As an example, a manufacturing application network might use protocols for 802.4 at the link level. In the same facility, however, a customer also might be using 802.3 with TCP/IP for its engineering workstations and a PBX for connecting terminals to general office computers. This raises the question of whether an architecture should use one protocol out of many, and force the other environments into using it, or allow coexistence of many protocols. The current HP AdvanceNet architecture provides for the use of multiple protocols within any single implementation.

Standard Protocols

We've mentioned the importance of standards. But why standards? Or should each vendor develop a proprietary network? Primarily, industry standards provide several important customer benefits. With standards, customers are more assured of multi-vendor connectivity. This allows them to choose the best equipment from a variety of vendors to serve their application needs. Standards also significantly reduce the cost of connectivity, providing for easier communication among various types of equipment (PBX, CPU, workstations, etc.) by encouraging the use of VLSI components from multiple vendors. Another benefit that standards provide is that customers can leverage their investment in network software and hardware. Since standards are relatively stable, customers can depend on the upgrade/expansion capabilities of a standards-based network.

Standards implemented within the OSI model allow replacing an individual layer without affecting the application. Hardware investments also can be leveraged because the same transport and network services can be used over different physical links. This commitment to standards is, however, no small task on the part of a computer vendor. It implies major challenges in the testing and support of multi-vendor networks. Today, certain standards

are established or emerging for communications networks. At the link level, standards have been reasonably well defined (E.G., IEEE 802.3, X.25). However, at the higher ISO levels, standards are less defined which has been an impediment to full multi-vendor compatibility.

There is, however, progress being made in attempts to stabilize protocols at higher levels. Excellent examples of this progress are the Manufacturing Automation Protocol (MAP) being proposed by a large number of manufacturing companies and the Digital Multiplexed Interconnect (DMI) Standard which is an ISDN-based proposal for standardizing PBX-to-Processor interfaces.

The lack of a good solution for document interchange between multi-vendor equipment creates yet another problem area. IBM's DIA/DCA (Document Interchange Architecture/Document Content Architecture) has received some support, but it is currently limited to text-only formatting, ignoring the rapidly growing need for merging text, data and graphics in multi-vendor environments.

Growth Path

The network architecture should provide customers with the capability to integrate new products (with protocols) on existing networks. Or, optionally, to upgrade existing networks without changing the interface to existing application software.

Let's look at Hewlett-Packard's experience. In 1977 HP introduced its Distributed System Network (DSN) which was based on a proprietary layered structure supporting standards only at the link level. Since the introduction, more than 19,000 minicomputer-based nodes have been installed on networks using DSN. In 1984, HP introduced its new OSI-based HP AdvanceNet architecture. This architecture is being implemented on most of the company's CPUs (HP 1000s, HP 3000s, HP 9000s, and personal computers) providing both an upgrade capability for this large installed base as well as a "co-existence" capability for "old" and "new" nodes.

Ease of Use

Ease of use can be achieved as a function of:

- Network services
- Transparent user access
- Network management

Network services provide ease of use. Even when different systems are successfully linked together, networks still require software to do productive work. This software called "Network

Services" provides the tools for data access and transfer across a network (see Figure 1).

Transparent access means that to access a network resource a user needs to know only its name not a physical location. Network directories provide users with this capability. Directories can reside either on a central system or in a network node. A higher level of service can be achieved when using network dictionaries that describe the data contents of each node.

Network management can contribute to ease of use by managing the network configuration for the users. An end user should not need to reconfigure his or her view of the network whenever the network changes and a new node or new protocol is added.

SUMMARY

We believe that networking will be used to a greater and more sophisticated extent in the future and that multi-vendor systems will continue to be a major part of every organization. Networks containing thousands of nodes and more efficient protocols will evolve.

A rigid architecture, one set of protocols, and a monolithic hard coded implementation of this set is not the right answer to the problem of multi-vendor networking. Migration is and will continue to be a major issue. In order to maintain the value of the investment in existing nodes, a good migration path must be provided.

The flexibility to use any one of several protocols within the framework of OSI architecture is a solid strategy for ensuring connectivity, software compatibility, and lasting value in multi-vendor networks.

Mr. McGinn is presently Group Marketing Manager for the Information Networks Group at Hewlett-Packard in Cupertino, California. In this capacity, he is responsible for overall network activity including strategic planning, architecture and products. Currently HP's more than 200 facilities around the world are networked with many sites currently linking office, engineering and manufacturing applications. Dennis joined HP in 1969. He holds a B.S. degree in engineering physics and an MBA degree from the University of Arizona.

Figure 1.

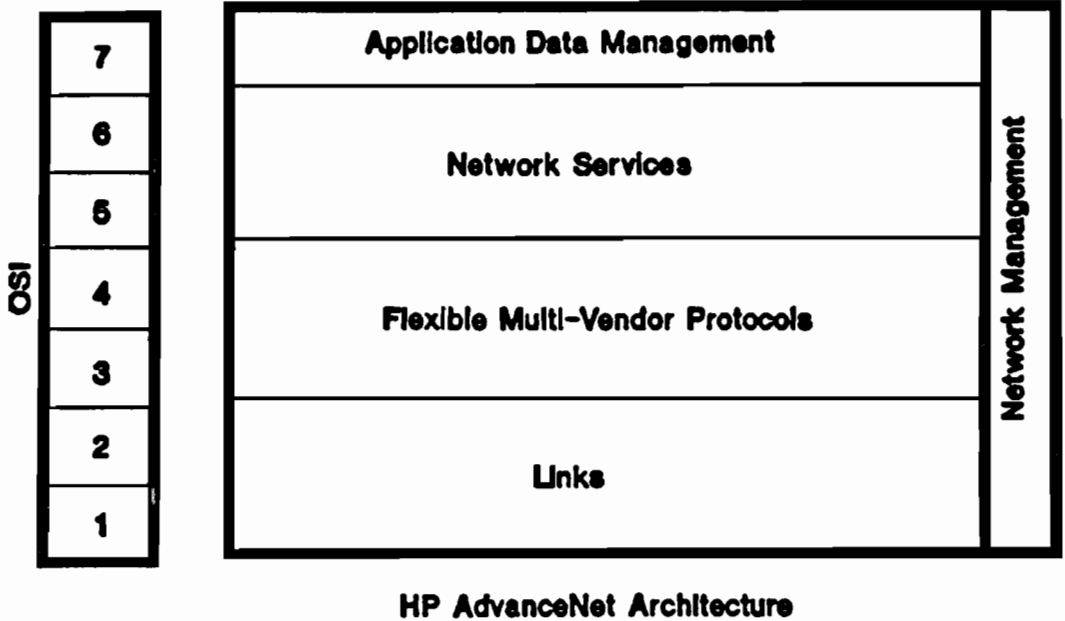
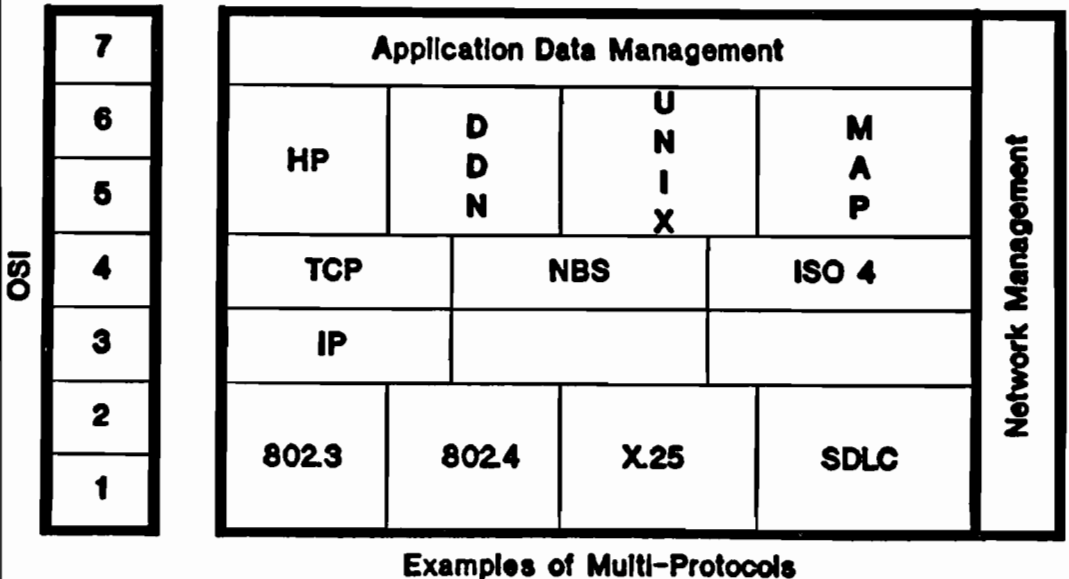


Figure 2.



3009. SOFTWARE TECHNOLOGY for the 80s
(Understanding Key Current and Future Technologies)

Bill Franklin
Hewlett-Packard
19447 Pruneridge Ave.
Cupertino, California 95014

OVERVIEW:

Software is an important technology that must be understood at a sufficient level of detail in order to develop an integrated strategy. Software technologies span multiple operating environments and are key in linking multiple environments together. In the following paper I explore current and future technologies in operating systems, office automation, data processing, database, data communications, and application software. Trends are discussed, insights into the future are explained, effects on the organization are covered, and some key technological considerations are addressed. After discussing the technologies a brief overview is presented on software planning including a guide to effective strategy development. With an understanding of software technologies, decision makers will be able to properly integrate them into their own strategy.

INTRODUCTION:

In John Jewkes's book, *The Source of Invention* he noted that "Peering into the future is a popular and agreeable pastime which so long as it is not taken seriously, is comparatively innocuous". However, it is important that we understand what is taking place in the computer industry to appropriately plan for the future. In Arthur Anderson's book *Trends in Information Technology: 1985* it was noted, "In the days of rapidly advancing technology no executive can afford to fall behind. You must know where technology is going, in particular, where data processing, office automation, and communications are headed." The industry is changing at an accelerating rate and success is dependent upon organizations understanding and accurately anticipating these changes in order to plan effectively.

There are a number of reasons that we must understand and plan for change:

- a. Organizations need lead time to adjust to changes.
- b. Organizations that implement effective strategies are gaining an important business / competitive advantage.
- c. Hardware and especially software are becoming a larger portion of capital investments as well as ongoing operating expenses.
- d. It can be difficult to recover from wrong decisions.

In looking at some of the general trends, it is becomes increasingly clear that the 1980's and beyond will bring a new environment / corporate culture to organizations.

- a. A real need for change is exemplified by surveys indicating that in many organizations less than 1% of all management decisions are being made using on-line interactive systems. Successful progressive companies are aggressively increasing this percentage.
- b. The time is right for change during the 1980s:
- (1) Organizations must address white-collar productivity. Over the past 20 years, automation has contributed to blue-collar productivity increases of around 80% while white-collar productivity has increased by as little as 4%. Productivity and efficiency are becoming key to many organizations today and this means effective software.
 - (2) The Japanese are working on Fifth-Generation systems that will provide what a number of industry watchers believe are some exciting new technologies such as:
 - (a) Expert systems by 1990.
 - (b) Cordic processors (3 dimensional extensions of the 2-dimensional array processors recently popularized) by 1995.
 - (c) Simple Artificial Intelligence by 2010.
 - (d) High Intelligence Quotient Artificial Intelligence Machines by 2030.

New technologies will start being introduced during the 1980's. Successful organizations will need to plan for them. We are constantly finding that unless change is anticipated and planned for, many organizations will not be ready. Software decisions will definitely be affected by the success of these projects.

- c. As a result of developing technologies the cost of hardware and software is changing:
- (1) Hardware costs are decreasing approximately 25-30% per year and technology is approaching the power of what we have previously defined as a mainframe in a personal computer.
 - (2) Software costs are increasing at 10-15% per year.
- d. From a financial commitment standpoint, user organizations as well as vendors are increasing their commitment to software:
- (1) Ten years ago, fewer than one out of every 5 dollars users spent on data processing went to software. Today the split is about 50-50. According to Stephen McClellan, author of The Coming Computer Shakeout by 1990, the ratio will be 4 to 1 in favor of the programs.
 - (2) According to McClellan the market for software supplied by U.S. companies will grow to \$50 billion by 1988 from about \$8 billion in 1983. This is a dramatic increase.

There are some changes that are reshaping our traditional roles and understanding of what information systems and EDP.

- a. Information systems are spanning over multiple departments, operations, and divisions within a company.
- b. The number of people effected by software decisions is increasing.

According to Trends in Information Technology:
1985

the number of electronic keyboard devices (word processors, personal computers, and terminals) in use in U.S. businesses will increase from 7 to every white-collar worker in 1980, to 3 to 1 in 1985, to one to every white-collar worker in 1990. With keyboards software is very important.

- c. Technologies are being developed that increase the number of employees in an organization that use software.
- d. There are some definite decentralization efforts taking place in many organizations.

Software strategies are now affecting more people in organizations than ever before and the complexities are causing organization changes. Arthur Anderson & Co. has noted that chief executive officers and chief operating officers who want to remain competitive will have to incorporate a new title into their management system. In Trends in Information Technology: 1985, they note that the emergence of the Chief Information Officer at the top executive level is essential to survival.

These are just a few of the trends that are pushing us into a new age of information processing. The accumulative effect of these and other trends will have a dramatic effect on software decisions in the 1980s.

Does your organization's software strategy reflect a clear understanding of the future and workable plans to take advantage of projected technological innovations?

The right decisions can be very critical to organizations' success. Successful effective software strategies while difficult to formulate, can make the difference between organizational success and failure. Some decisions are so key that managers are truly "betting their company" on software decisions.

How effective is your software strategy?

What does it include?

Does it include all the elements necessary for success in the 1980's?

Are you knowledgeable about trends in software technologies?

Have you set your organization up for software success?

The objective of this paper is to help provide the decision makers with an understanding of alternatives, insight into some of key technologies, and a framework to develop an effective strategy. It is not to give everybody a

crystal clear software plan. Software strategies must be developed with your organization in mind. We can all say that we want to be on the leading edge of technology, but this is not practical or realistic. With an understanding of trends, organizations can move into the 80s with knowledge and insight. Software strategies must be developed with an understanding of key areas.

Each of the following areas will be discussed:

- I. Operating Systems
- II. Office Automation
- III. Programming / MIS Management
- IV. Database
- V. Data Communications
- VI. Applications
- VII. Framework for Structuring a Software Strategy

I. OPERATING SYSTEMS:

There has been a lot of concern about where operating systems are going in the future. It is really important to understand the trends.

There are some definite battles shaping up in this area. Many users would like to see a common operating system. However it is important to look at the motivations of various hardware vendors who also have a great impact on the operating system. The operating environment that has the greatest chance of becoming a standard is "UNIX". "UNIX" is a U.S. trademark of AT&T Bell Laboratories. From analyzing trends and data we can make educated guesses on what hardware vendors might consider doing with UNIX. The following are thoughts on what might possibly happen in this area (this does not reflect any official HP viewpoint - only insight gained from the author's research): IBM could make UNIX available as an operating system choice on their smaller systems. Long term it seems likely that they will continue to emphasize proprietary systems in UNIX application areas. UNIX could present a threat if it ever becomes a reason not to buy IBM equipment, so they will probably try to diffuse UNIX as an issue. DEC could try and use it as a way to leverage into the commercial market. AT&T will probably view it as a way to catch up on software development. The Japanese could use it as a way to increase their low-end systems share. Since HP believes in supporting standards, HP will probably continue to support the standard AT&T version of UNIX.

The following are reasons that customers prefer UNIX:

- a. They do not want to be locked into buying a system from a particular vendor. They see UNIX as a way to get a commodity operating system.
- b. Universities and research institutes have endorsed UNIX - 1/3 of the computer specialists in the U.S. have experience with UNIX according to the Gartner Group.
- c. Those who want a common operating environment on heterogeneous machines
- including different vendors.
- d. They want to reduce development, support and training costs for special purpose systems.

Users who really want to have a common operating system need to track this trend and buy the standard AT&T UNIX. There are also some perceived UNIX deficiencies that need to be addressed such as security, back-up, and standard file structures. Watching how these areas are addressed will be interesting.

It will be very important to watch the industry and see if standards for UNIX can be developed and accepted by the vendors. A group has been working for over 3 years now. Watching how IBM and AT&T work together will be key. UNIX sales are projected to increase at least through 1988 according to various market forecasts.

Another trend in operating systems is that they will become bundled in the vendors firmware. This makes it difficult to implement other vendors operating systems on other vendors hardware as well as provides the hardware vendor with increased efficiency. There are rumors from within the industry that some vendors plan to develop proprietary operating systems in firmware.

The dramatic technological advances being worked on by HF, IBM, DEC and approximately 10 startup companies to develop RISC machines will probably not have any dramatic impact on the operating system environments. It does appear that UNIX could be an alternative.

If we look at the Japanese experiment in 5th Generation systems, they plan to have parallel capabilities that will dictate new operating environments. This will be an interesting trend to track in terms of what we do with operating systems in the 1990s.

Speed of light (optical) computers using light waves instead of electronic circuits performing a trillion operations per second are probably 5 years away from the prototype and estimates are that they are around 10-15 years before they reach the commercial market. According to an April 1985 article they will revolutionize the way information processors are designed and built. However, we are still a number of years away from this concept.

For the next few years, users need to watch UNIX and determine if standards are developed. It is a time to continue to develop code that is not significantly dependent upon special features in the operating system.

In the later 1980s if standards are developed, UNIX implementations could be aggressively pursued with confidence. It will be important to track 5th generation system development and determine how they will migrate existing users into the new operating environment.

II. OFFICE AUTOMATION:

There are a number of important strategic software decisions that are being made now that are very important to the office employee. What is really included in this category are those with less technical experience. Many of those affected by office automation are new software users. Needs in this area are becoming more visible. This can be exemplified by a Ruder Finn & Rotman Inc. study, "People in the Office", based on interviews with Fortune

1300 companies noting it is difficult for them to to keep up with the paperwork demands. Only 9% of those surveyed indicated that they turn out 80% or more of their office paperwork on word processors. The need to expand the use of productive tools is becoming evident.

As a result of these and other surveys of the current environment and needs, organizations are developing plans as noted below:

- a. Nearly 60% of the Fortune 1,000 companies in the U.S. today have established policies and strategies that address the new office automation technology. Implementation are still in process for many.
- b. By 1985, close to 85% of the Fortune 500 industrial and service companies in the U.S. will have working policies in place for evaluating, purchasing, and installing word processors, personal computers, electronic mail systems and other office automation equipment.
- c. Within the next 2 years, half of all the small and medium sized businesses in the U.S. will have committed themselves to the development of practical office automation strategies.

Plans are also being funded. Organizations are committing funds to these efforts as indicated below:

- a. A.D. Little research organization expects the office automation market to grow from approximately \$11 billion today to around \$36 billion by 1988.
- b. Electronic Mail services will grow from \$80 Million in 1983 to more than \$2.1 Billion in 1988. There is an average rate of growth of 107% according to Link Resources Corp, a marketing research firm.
- c. In the same "People in the Office" study referenced above it was estimated that by the year 2000, literally trillions of dollars will be spent on electronic equipment and personnel to run it.

As a result of these expenditures, the office worker / user is changing. Changes have already begun in many organizations and are continuing. The following are some of the trends that have taken place:

- a. End users are becoming more educated and demanding more.
- b. Within the past 3 years end-users have been given direct access to technology. We now have direct-users rather than end-users. More transactions are being collected / processed closer to the user.
- c. During the 1960s (mainframe era), users were dissuaded from asking for applications that they might actually need. Systems analysts decided for them. End or Direct-users are now more in control.

There is momentum for office automation and plans are being put into action. However, the implementation of software is not always easy. There are areas of resistance that must be understood.

Office resistance can be exemplified by the results of an August 1982 Kelly Services survey. In the survey of 527 temporary office workers it was noted that 36% said they were worried about the "electronic invasion in the

office". 1/3 of these were most afraid of being displaced by machines and 1/3 were concerned about where to learn new skills. This emphasizes the need for education. Many of the concerns can be alleviated through proper training (formal and self-paced). HP has a number of classes that can help in this area.

There is also concern about what vendors are calling "user friendly". In order for non technical employees to be able to use office software it must really be user friendly.

Some of the examples that HP has used in order to help satisfy this concern about user friendly is to quantify it. The goal for office products is to provide an environment where the user can be minimally functional competent within one-half hour of use. It can truly be done with a number of HP's products such as HPSLATE, and EZCHART. The integration of HPMENU provides an easy environment to change between various functions. These are a few examples of implementing the "ease of use" objective.

Increased "functionality" is also important in order to help alleviate concerns. With the ability to increase office productivity and perform functions that take time and have low rewards such as looking up words in a dictionary, software can help reduce some of the office worker concerns. The integration of a spelling dictionary into HP products that covers 74,000 words from the American Heritage Dictionary (99% of ones regular vocabulary) has a very positive effect.

Products engineered with ease of use and "functionality" in mind make automation easier by reducing resistance.

There are three technologies that I believe have the power to move office automation into a new age. They are:

- (1) electronic mail, a technology that is having a dramatic effect today,
- (2) graphics, an existing technology that has a much greater potential as packages become refined and users understand the advantages, and
- (3) expert systems that have the potential to change the way we get information.

Electronic Mail:

Electronic mail is changing the office environment and will continue during the rest of this decade for many corporations. With the implementation and use of this product, more employees are being introduced to office automation. They are becoming dependent on it for their communication. Along with mail implementations, other products are needed such as various word/text processors to create and edit messages. Office workers are creating and "mailing" words, text, and graphics. The ability to electronically send and receive messages has been a motivation to learn. With the introduction of personal computers and portable computers we are seeing an era where managers are composing letters/messages on air planes and going to a telephone booth and sending their message to one or many people. With new innovations in communication technology they will also be able to read/send mail while traveling in the air in the near future. At HP there are approximately 20,000 users of HPMAIL. This has had a very

positive effect on communications at HP. This is really changing the way we do business.

Graphics:

Graphics is another important software technology. The mind thinks in graphics. Memory specialists convert words into pictures and can demonstrate amazing recall. In order to get information, we as human beings have 5 choices: touch, taste, smell, verbal, and visual. While the information transfer capacity touch, taste, and smell is obviously low, verbal communication is only between 150 - 300 words per minute. The channel with the greatest data capacity is visual. The eye has 150 million light sensors, a million parallel fibers that link the optic nerve with the brain; and the ability to distinguish between 160 different hues. From this and other data we can determine that the visual channel is capable of processing information at 48 - 72 million words per minute. Humans read words at only 600 - 1200 words per minute. Graphics allow us to be able to take full advantage of this channel.

An interesting study was done at the Warton School of University of Pennsylvania showed how powerful graphics can be in getting information to people. The study had the following:

- * 2 speakers each giving a presentation 18 times,
- * 1/2 the time they argued one side and
and the other time the other side,
- * graphics were used by both speakers for an equal number of presentations.

When graphics were used, 2/3 of the audiences voted for that speaker. Graphics is a really powerful tool that gets results.

A survey conducted by Business and Professional Software in Cambridge, Mass. shows that less than 30% of all the visual material prepared for informal business meetings consisted of graphics of numerical data. Why?

Graphics as a concept must be understood and its value internalized. More people are becoming familiar with graphics as a result of the Personal Computer and its software. Lotus 1-2-3 and other packages have had a great impact on introducing people to graphics.

Early this year (1985), HP announced a new graphics package for the HP touchscreen (Graphics Gallery and Executive MemoMaker) that allows users to merge presentation graphics with word processing documents using the ThinkJet and LaserJet printers and plotters. There is also the option of using a Mouse with Drawing Gallery. This product increases the potential use of graphics. There are also many other non HP graphics software packages that run on the touchscreen that have also increased the number of people using graphics.

Graphics software is getting less expensive and easier to use. While in the 1970s it was possible to generate graphs, it was very difficult and expensive. Now graphics terminals are getting less expensive (HP2623A - \$3,250) and personal computers are constantly being reduced. There is a

trend in the market to eventually bundle graphics into the hardware offering due to the growing understanding and realization of its value.

How can organizations effectively plan for this technology?

The answer to this question will depend upon where each organization is in their development. Many organizations need to educate key managers on the value of graphics. Training is really key. Education on general graphics as well as details on how to design effective charts can be helpful. HP has a publication "Steps to Effective Business Graphics" that provides some key guidelines on how to design charts.

Expert Systems:

Expert systems are gaining in importance and becoming a reality. An expert system is a software program that contains general, experiential, and intuitive knowledge of an expert in some application domain. It uses this knowledge to infer new information and solve problems normally associated with human intelligence and not amenable to traditional, algorithmic computer. Expert systems contain a knowledge base, a reasoning mechanism, and a control mechanism. Unlike databases they arrive at a solution.

What expert systems attempt to do is to make decisions. In a experiment documented by Edward Feigenbaum, co-author of the Fifth Generation - Artificial Intelligence and Japan's Computer Challenge to the World, a case in a business class took MBA students 26 hours to find the answer while CEO's took 20 minutes. The difference was that MBAs had to think the problem through, while CEOs provided their best guesses. Expert systems won't work unless experts describe how they make decisions. While understanding / reasoning is difficult to quantify, information scientists are working at these tasks.

Psychologists over the last 50 years have shown that the number of pieces of data the conscious mind can most comfortably handle at any given moment is only about 4. Therefore with computers, this can greatly be improved and will help increase the demand for expert systems and artificial intelligence.

While expert systems are in their early stages of development, it is an interesting trend to track.

A prediction about the effects of expert systems was made by Feigenbaum noting that number crunching will look very small by the end of the decade and that programmers will be displaced by computer-based intelligent software development workstations. He believes that programmers will need to be retrained as "knowledge engineers" codifying the knowledge gained from the experts.

Gene Amdahl in referring to expert systems noted that "it can be done with 4th generation equipment, although it might be cheaper to do on 5th". There are some "Expert systems" that run on the HP3000s today. However, there is

still a lot of development that must take place to move them from their youth into maturity.

Chess machines can now perform at championship levels in playing chess better than 99% of the population. Medical systems have been developed and are being used that allow the diagnosis of diseases. We need to understand that this technology will be a reality in the future and include it in those key areas that are important to your organization. It is important that preliminary work be done on how decisions are made dealing with your organization's concerns.

We need to realize that access to information will improve. It is important to gather information and establish the databases that can be used in future decisions.

These trends are just a few that will dramatically change the office environment.

A real software challenge is to integrate office software with current systems as well as develop a software support plan.

It is important to understand current options, technology that will be becoming available, where office software is going, and how to integrate office automation into the organization (both organizationally and from a people standpoint). Training needs to be a key factor in the strategy.

III. PROGRAMMER / MIS MANAGEMENT:

Where is this role evolving?

How should staffing be developed for this function?

What type of expertise is needed?

What software planning should be done?

There are some definite trends that are taking place that focus on the need for a re-thinking of how we view this area.

James Martin noted that by 1990 computers will increase in number by 25%, be 10 times as powerful, require 93.1 times as much code to operate resulting in a need for 27.9 million programmers in 1990 (up from approximately 300,000 in 1980). This explosion assumes no increase in productivity which is not realistic. He noted that COBOL and PL/1 will not be the language of the future. EDP Managers loose or quit their jobs on the average of about every 2 1/2 years. The current pressures of trying to satisfy a demanding user community is a definite contributing factor. Inadequate software development tools and techniques contribute. The EDP backlog is still growing. Daniel McCracken noted that 2 to 3 year backlogs are becoming the norm and some organizations have 5 year backlogs. This is a definite problem and a trend that can not be allowed to continue. As a result of the backlog, many programs are being written without concern to efficiency, coding techniques, or flexibility. It is normally said that it takes approximately twice as long to write a flexible application as it does to write an inflexible one that only addresses the actual needs of the user.

As a result of technology, the role of the programmer and MIS manager is changing. The following technologies are key:

- a. 4th generation programming languages are gaining acceptance. Languages such as HP's TRANSACT allow 1 line of code to take the place of 30 - 50 lines of COBOL code.
- b. Tools such as 4th generation programming languages are making the concept of "Prototyping" a reality and a much more feasible techniques to system / program development.
- c. Application generators are being developed that can generate a program from having the user answer questions. HP's RAPID products as well as third party tools can contribute to this area.
- d. Systems for "System Development" are also being developed. AT&T coined the term "Workbench" that provide aids in system development. HP's Toolset helps with program development.
- e. Dictionaries are reducing the amount of code and data structures that need to be maintained.

These trends from both a technological and user-demand view point, will cause a change in this area during the 80s in many organizations.

In order to respond to these trends it appears that the MIS function will need to undergo some dramatic changes from both a programming and organizational standpoint. Some of these changes are:

- a. Programmers will need to learn new 4th generation languages. While there is the tendency for many programmers to resist learning new languages that are not standard throughout the industry, they will need to learn them or find fewer job opportunities.
- b. Prototyping as a concept will need to be learned and internalized into certain business situations. It has great value when users do not know exactly what they want and can not verbalize it or when the objective is to get them involved.
- c. Programmers and EDP management must work at developing and buying tools for users to be able to do their own work. They need to provide the guidance in helping users become more productive on such tools as HP's INFORM.
- d. Steering committees will lose their importance since end-users will be able to do more of their own work. Rather than have users provide guidance on how to allocate a limited resource, MIS will be helping to train users on how to address their own needs.
- e. Organizations will need to look at developing guidelines on when to develop applications using new techniques and languages vs. the current standard used techniques.
- f. Organizations need to determine how to implement end-user computing. It will be a real challenge to determine how to integrate it into organizations in an effective manner.

Personal computers are starting to be used in program development as well as being incorporated into software application strategies.

Languages are evolving from 1st Generation Machine Language to 2nd Generation Assembler to 3rd Generation COBOL, BASIC, FORTRAN, etc. to 4th Generation Languages today. While it is easy to say all applications should

be developed using 4th Generation languages, it is more complex decision than this. There are some reasons that various languages should/could be used in particular organizations.

Language selection is key. There has been many languages that are being used for different reasons in organizations today. Some basic guidelines are:

- a. COBOL is the most widely used business programming language today. Initial testing of COBOL-80 has shown productivity increases of 20% - 25% over COBOL-74 due to some new key verbs such as EVALUATE that reduces the number of nested "IFs". While it is not a standard yet this is a language that should be tracked.
- b. ADA is being supported by the U.S. Defense department and some believe will be widely accepted in the future - especially by those companies that want to do business with the U.S. government. An ADA compiler for the HP Series 200 was introduced in 1984.
- c. Pascal is being used in many schools. The College Entrance Examination Board chose Pascal as the programming language underlining its Advanced Placement Computer Science Examination in Computer Science.
- d. BASIC is used in many secondary schools today and widely used on personal computers.

While this is only a few languages, there are over 100 languages in use today and more are being developed. The trend is toward higher languages.

The future will probably show COBOL maintaining a position in current commercial applications but 4th generation languages gaining. The biggest problem is that it will take a long time for standard 4th generation languages to evolve. 4th Generation language users need to realize that they use more resources.

Organizations need to continue to track new tools in this area and evaluate how to use them. Training and adjustment to new environments will be key challenges.

IV. DATABASE:

Database technology has been very important over the past few years. The basic trend in databases is that they are becoming easier to implement and more flexible.

There are basically 3 types of databases today:

- a. Network - (ex. IMAGE)
- b. Hierarchical - (ex. IDS)
- c. Relational - (ex. INGRESS)

It is important to understand the definition of a database since this is becoming confusing due to the increase of "databases" on personal computers. A database can be defined as "being composed of more than one file and has the ability through commands to change multiple records with one command." This eliminates many so called databases (especially many of those on personal computers) from the formal definition.

There are many hybrids of these databases but new technology (hardware and software) will provide us with some new guidelines into how we use databases. We will be seeing data residing within a file structure and being able to be accessed in many different ways. The same data could be accessed for example using either relational or network database methods.

In selecting a database it is important to evaluate their ability to be modified to fit on other systems. The separation of functions as well as "scalability" will be important in the future. The separation of functions will allow the user interface functions such as SQL, the relational data base method that is becoming a "de facto" standard, to be easily incorporated into the language. If others become more functional and universally used, then they could become incorporated into the database due to modularity.

While relational databases are viewed as easy to use / modify, technology is just now being developed to provide fast effective access. Since there is a need for extensive memory for access, there is a problem of providing for power failures. What happens to the data and pointers when memory loses power?

David Hsiao, chairman of the computer science department at the Naval Post Graduate School in Monterey and a noted database engine expert, noted that only 10% of the raw data relates to a users query and that only 10% of that relates specifically to the answer. In order to get the correct data significant I/O resources could be used.

Database engines are a technology that can be implemented in one of 2 flavors (hardware or software). While a software implementation is possible, Hsiao believes that there should be only a temporary solution. Britton-Lee shipped its first database engine in 1981 for the HP3000. They have increased the capabilities to increase support from 128 users on 24 computers and 4 disc drives to 4000 users on 64 computers and 16 disc drives. This database alternative has potential when you want to optimize the amount of data that you want to get in a timely manner.

It is important to evaluate database engines when access needs are significant and a separate processor is needed. While in the short term, database engines must be carefully evaluated to determine their viability in a strategy, long term they are expected to be a integral part of Japan's 5th Generation computer system. These systems are expected to have different computers performing symbolic manipulation (arrays), numeric computation, and database access.

Another trend in database technology that is important to be aware of is the move of having the same database run on a variety of systems from the large corporate system to the personal computer. While HP's IMAGE runs on HP1000,

MP9000, and HP3000s, the current HP version does not run on the personal computer. Recently a 3rd party developed a package that looks like IMAGE and runs on the personal computer. This helps makes the entire product line compatible.

Today an effective strategy is to use network or hierarchical databases to extract that approximately 10% of the data needed to evaluate and then use relational database technology on personal computers or dedicated mini computers.

As hardware technological innovations are developed that increase processing speed, database engines and relational databases could be an effective solution in handling large data bases.

In the future "truly" relational database technology will be implemented that will be efficient and allow data sets for the same database to reside on multiple systems. We will have truly distributed data processing.

Database technology is a very important area to track. It must be understood, planned for, and relevant data collected. There will probably be conversions between today's database technology and future database implementations by successful vendors.

V. DATA COMMUNICATIONS:

According to Stephen McClellan "As software gains ascendancy, a new word will arise to characterize the activities of the industry. The key words will no longer be customization, specialization, and market niche. The new word will be integration. All computer equipment will need to communicate."

Data communication speeds are increasing and having an effect on how we plan for software. Copper wire can be used at transmission speeds of 5000 characters per second, satellites at 100,000 characters per second, and fiber optics at 3,000,000 characters per second. This means that the entire Encyclopedia Britannica can be transmitted between 2 points in less than 2 minutes.

Global communications is expected to increase around 20% over the next decade. Along with this are some interesting developments such as a project among 17 different telecommunications agencies in the Pacific regarding the possibility of the first transpacific optical-fiber cable. The cable would go from California to Hawaii to Guam and then branch off to possibly Japan, the Philippines, or Taiwan. The cable would be capable of carrying approximately 37,500 simultaneous voice conversations.

While there are some very exciting developments taking place in this area, it would be impossible to cover all of them in this paper. Speed will not be as much of a problem in the future. There will also be more communications alternatives. Software must consider them. The following are some areas that are important to understand in planning software strategy:

Standards are being developed. As the ISO (International Standards Organization) OSI (Open Systems Interconnect) layered model open-system architecture becomes more formalized between all 7 layers from the physical to the application, communication will be possible between all hardware vendors. The standards committees have a long way to go but progress is being made. Another "de facto" industry standard is IBM's SNA. While there still appears to be some debate if this will be there long term strategy, it is today their recommended model. Recognizing that there are such "de facto" industry standards such as SNA, HP is also committed to maintain "de facto" industry standards. The reality is that there are some potential conflicts of interest in that it will be more difficult for vendors to protect their hardware base if all systems can be integrated. It is important for users to follow the developments in standards and encourage hardware vendors to follow the standards.

Security is becoming more and more of a concern with networks. There are adequate ways of planning for an effective security system on networks when it is understood and properly implemented. This can be done using a combination of hardware and software. Software techniques on the HP3000 can be effective when properly implemented.

January 23, 1984 issue of Business Week noted that 200 U.S. companies have experimented with some form of tele-commuting, and more than 30 of them have formal tele-commuting programs. Unions have expressed concern about employers taking advantage of their employee and expecting more hours. Business Week expects that tele-commuting will grow rapidly over the next decade, because employers are finding that it generates major productivity increases.

Interfacing between other vendors systems is becoming more of a concern and products are being developed to address this need. There are products that allow the IBM PC and the Wang PC to interface with the HP3000 by running a data communication program that provides HP2622 block mode terminal emulation. Also products are being developed to convert documents between HP WORD and Wang OIS WP, and an HPDESK to IBM PROFS text mailing capability. For communications with DEC products, HP currently supports the UNIX operating software on the HP9000 that provides terminal emulation and file transfer to DEC and other computers running UNIX operating systems. HP is working on a file transfer capability between HP and DEC systems through an IEEE 802.3 local-area network. These are only a few of the interfaces that are available now or will probably be available in the near future. This area will be solved by vendors or 3rd parties because there is a market for products like these.

The integration of personal computers into networks is becoming more of a challenge. Local area networks as well as software to integrate personal computers to central processors will increase in importance during the 80s. HP's AdvanceLink can bring the HP Personal computer user into the HP3000 and extract data and transmit it to the HP personal computer. HPMESSAGE enhances communication between the IBM PC and the HP personal computers. HP supports standards such as IBM's DIA/DCA and DISOSS which helps in multi-vendor communication.

Just like there has been a trend in organizations over the past few years to establish "Database Managers", a new trend is to establish "Network Managers". With the significant advances taking place in both communications hardware and software as well as the increased emphasis on distributed data processing, the need for a "Network Manager" is becoming a real necessity in successful organizations. This person needs to have both technical and managerial skills to appropriately address software needs that span complex organizational structures.

Advances in data communications are one of the most important trends to track. Successful organizations will definitely need to incorporate these advances into their organizational strategy.

VI. APPLICATIONS:

Yasuo Ishii, of Fujitsu, Ltd. noted at the 1984 Telecommunications Conference in January that "Ninety percent of applications can be handled by 10% of the developed software; but 90% of the software must be developed for the remaining 10% of applications.

This sector of the software industry is becoming more important than ever before according to Stephen McClellan. He noted that 70% of the projected \$50 billion sales in 1988 will be application packages (up from \$8 billion in 1983 to \$35 billion in 1988). Buying packaged software is a trend that has accelerated in the past few years.

It is becoming increasingly difficult to determine what applications to buy when there are over 100 being developed per month (this includes the Personal computer applications).

With the number of external databases being developed, companies need to start thinking about tying into other companies' databases such as is being done with the airline reservation systems.

Some of the trends in this area that decisions makers must be aware of are:

- a. Vendors are trying to protect their software. There is a trend to restrict source code.
- b. Main stream solutions are being developed by computer manufactures that take advantage of all the systems' capabilities including micro-code.
According to Stephen McClellan "IBM has been micro-coding or hardwiring more of its systems control program instructions into its mainframes, making it more difficult for third-party vendors." This could have an impact on the software business.
- c. On-line applications are increasing. More data can now be kept on-line with the cost per megabyte on large disc/disks being decreased by around

20% per year. The cost per megabyte on HP7933H is around \$63 / megabyte.

- d. Batch systems are being converted to on-line.
- e. Central applications are being distributed. Following the trend to move systems closer to the users.
- f. The smaller the organization the more likely that they will buy a software package and the less likely that they will significantly modify it.

The flexibility of the organizations's user community and the funds that are available to get an exact requirement match, will be key factors in the determination of how packages fit into an organization. Package decisions should be made in favor of those that contribute to protecting an organization's software investment.

VII. FRAMEWORK FOR STRUCTURING A SOFTWARE STRATEGY:

Basically we must ask ourselves what constitutes high quality software and then look to develop a future software strategy around these elements based on their importance to your particular organization.

Some of the key characteristics are:

- a. reliability
- b. flexibility
- c. friendly
- d. complete
- e. supported
- f. easy to maintain
- g. easy to learn
- h. integrated
- i. good performance
- j. hardware compatibility

Before we focus on detailed planning there are some general guidelines that could be worth considering. These could help provide insight into the objective establishment phase of the plan.

These general guidelines are:

- a. In developing a software strategy it is important to keep in mind Pareto's "80/20 rule" as it applies to software. Werner Frank, a software consultant, applied the rule to software noted that
 - (1) "Eighty percent of the computer user's requirements associated with a given application are satisfied by 20% of that application's usual features and functions."
 - (2) "Eighty percent of the overall user population can satisfy 100% of

- their needs with 20% of the typical software application's capability".
- (3) "...users employ 20% of a word processing product's capability during 80% of the software's execution time; during the remaining 20% of the time some of the less used, more obscure features are employed.

- b. More's Law that "Information retrieval systems tend not to be used when it becomes more painful to get information than to live without it" is also interesting. It is important to make sure that software is designed / procured that will be used and solve needs - Not contribute to pain.
- c. Software ergonomics focuses on designing and procuring solutions with the user in mind. Ergonomics comes from the Greek word "ergon" for "work" and "nomos" for "law". It is the science of adapting the work-place to the worker focusing on safety, comfort, and efficiency. It is important to apply this to software. Strategies need to have software ergonomics incorporated into them.

In developing a plan there appears to be some basic steps that can help contribute to success. By following the steps in the outline below and addressing the questions, a plan will evolve.

SOFTWARE PLANNING STEPS:

- a. Requirements Definition:
- The first step is to understand your current environment. Before any steps can be made to determine where you want to go, it is important to know where you are today. Some of the key areas that need to be addressed are:
- (1) What does my current environment look like?
 - (2) What systems do we have in place?
 - (3) How much support is the data processing department getting?
 - (4) How committed is top management to automation?
 - (5) To what degree do users want customized applications:
 - (5a) How much are they willing to compromise on their exact desires to fit a package?
 - (5b) How much are they willing to pay for an exact fit?
 - (6) What languages are being used?
 - (7) What is currently being supported?
 - (8) Competitive pressures - to what degree are competitors automated?
 - (9) How great is the demand for information?
 - (10) How great is the current backlog? (2 years?. . .5 years?)
 - (11) How technically sophisticated and interested are the users?
 - (12) What software security is in place?

b. Objectives:

From an understanding of the current environment, realistic objectives for your organization can be established. Some of the key questions that can help determine the objectives for your software strategy are:

- (1) Budget potential? (is money a problem for justified applications?)
- (2) What is the expected pay-back period?
- (3) How urgent are the systems perceived to be?
- (4) Projected EDP support expenditures - What % of sales, etc.?
 - (4a) Higher - more reliance on EDP and tools for programmer.
 - (4b) Lower - more reliance on user and tools to support user.
- (5) Degree of state of the art desired?
- (6) Emphasize technology . . .make it a competitive advantage?
- (7) How fast does top management want to automate?
- (8) How many vendors are on your short list?

c. Plan:

As a result of answering the above questions a plan can be formulated. Some of the key elements of the plan that must be addressed:

- (1) Training required to obtain objectives addressing the needs of users, programmers, technical staff, MIS management, and others.;
- (2) The plan should be a "phased in" process with explicit list of milestones that will help in accomplishing the objective.;
- (3) Critical applications - what needs to be addressed and when.;
- (4) Languages wanting to phase into and when they should be used.;
- (5) Under what conditions should organizations use 4th generation programming languages vs. 3rd generation vs. buying packages?;
- (6) Criteria for evaluating software packages.

d. Implementation:

The following are some general guidelines that should be in an implementation plan:

- (1) Phased in software plan.
- (2) Start with critical applications.
- (3) Do not try to automate all areas overnight.
- (4) Follow a plan and remember the adage "When you fail to plan, you plan to fail".

e. Review:

Plans and implementations need monitoring. There needs to be a periodic review process incorporated into the strategy.

The following are some key points to consider:

- (1) Need to be done at least once per year.
- (2) See where we have been and where we are going.
- (3) Is progress being made toward the objective?

CONCLUSION:

It is becoming evident that software is key and planning for it is important. There are a number of factors that must be considered.

Developing an understanding of trends, technology, and terminology will help assure success. While it is impossible to discuss all the factors and trends that organizations need to understand, hopefully some insight was gained from this paper that will help you design your software strategy / plan.

It is always helpful to have your plan reviewed by a number of other individuals. The key to developing a sound strategy that can be shared with others and discussed from a sound business standpoint is to formally document your software environment and organizational objectives. From this, consultants who understand software can help tailor a plan for your organization.

Good luck.

NOTES / ACKNOWLEDGMENTS:

The data for this paper came from a number of sources including articles from Computerworld, Interact, the HP Chronicle, Software News, Business Week, unpublished studies, as well as 3 books that can be interesting and valuable to those interested in developing a strategy.

1. Trends in Information Technology: 1985 published by Arthur Anderson.
2. Fifth-Generation Computing by Edward A. Fiegenbaum and Pamela McCorduck.
3. The Coming Computer Shakeout by 1990 by Stephen McClellan.

BIOGRAPHY:

Bill Franklin, B.S., M.B.A., C.D.P has worked on both the technical and managerial level within data processing for 16 years serving in many functions including programmer, project leader, manager, professor, and various functions in computer marketing. He is currently a Program Manager for Hewlett-Packard's Worldwide Major Accounts Program covering all products (computers, instruments, and components). He is an experienced speaker and has published other data processing papers.

3010. COMPUTER INTEGRATED MANUFACTURING (CIM):
NOT A SOFTWARE PACKAGE OR A MAGIC WIRE

Terry H. Floyd
ASK Computer Systems
730 Distrel Dr.
Los Altos, California 94022

Large manufacturing companies have been "computer-izing" their operations since the 1950's. Smaller companies have become involved during the last 15 years. Usually, the movement to automate the factory has been "user driven," that is, it has occurred from the bottom up. The result has been a proliferation of unconnected computers and systems, each of which addresses a small piece of the total manufacturing operation. Some companies have as many as 50 independent systems, each with its own hardware and software. A jungle of abbreviations and acronyms has evolved, sometimes without agreement on their meaning.

The various systems relating to manufacturing applications and now being called CIM are also frequently referred to as "islands of automation." For instance, engineers design products with the help of Computer Assisted (or Aided) Engineering (CAE); systems which juggle the many variables of the physical environment (chemical, mechanical, electrical, etc) with the properties of a company's products to solve a customer's problems.

Draftsmen now use powerful graphics systems (CAD) to relate what engineers have designed to the workers who must build the products. Sometimes engineers do the CAD work themselves, eliminating draftspeople.

Industrial and manufacturing engineers design whole manufacturing systems (factories or production lines) with similar tools (CAM). They now use intelligent machines and robots in their designs (CNC and FMS).

Material handling systems range from intelligent scales that count parts based on their weight to automatic storage and retrieval systems that deliver raw materials, components, and subassemblies to assembly lines and finished goods to warehouses.

Flexible Machine Systems (FMS) and flexible machine centers are computer controlled groups of machine tools (numerically controlled lathes, drills, etc) that can perform different operations or make different parts simultaneously. They may also schedule material and the priority of tasks to be performed.

Quality Control has been automated by automatic test equipment (ATE) - computers that collect failure data for feedback to process improvement functions. Other data capture/collection devices being used on the shop floor include optical character recognition (OCR) and bar coding. Vision input and voice output are making robots a more common site in factories.

Everyone who works in an office is aware of the spread of word processors and micro or personal computers into the workplace. It's happening on the factory floor, as well, not only for data collection, but for computation as well. Other forms of automation like phones, copiers, and even postage machines are being integrated into the CIM system.

Applications like Payroll, Accounts Payable, General Ledger, Accounts Receivable, and Fixed Assets, the standard accounting applications, have been automated in most manufacturing companies for at least a decade. Many manufacturers became interested in the accounting aspects of their factories in the 60's or early 70's. The attempts to manage inventories and costs evolved into Materials Requirements Planning (MRP).

MRP is an attempt to integrate numerous manufacturing functions - Inventory Control, Bills of Material (BOM), Product Structure, Industrial Routings, Purchasing, Planning, Scheduling, and Shop Floor Control into an integrated package. The idea is to plan the independent demand items (what are the customers going to buy) and let the computer system plan the dependent items (through the BOM and part information). MRP is also the effort to time-phase raw materials and components to their requirements, thereby reducing inventories.

MRP II (Manufacturing Resource Planning) was the next step - another integration of existing modules. This time MRP was integrated to the traditional accounting functions which were already in place. The purpose was to provide feedback on the attainment of business plans and goals and streamline procedures. Quotes and Orders could feed Master Production Scheduling (MPS); invoicing could relieve inventory and voucher sales agents' commissions; costing of inventory could bring details and accuracy to margin analysis. Part receipts through purchasing could aid Accounts Payable functions; General Ledger postings could be fed from the Factory Ledger which permeates the MRP system; capital assets and projects for R&D or process improvement could be closely monitored; direct and indirect labor could feed payroll and labor distribution systems.

What has been happening, then, is the development of independent programs to address localized problems, then the

eventual integration of these "islands" into working, co-operating, integrated systems.

The trend developed in the past (the integration of small, specific purpose systems) is continuing in the manufacturing environment and has a final, all encompassing name: CIM.

The end result of CIM may be a century or more away: computers will accept orders directly from customers or brokers (like today's travel agents), design products, order parts, receive/store/route components and subassemblies, build/assemble/fabricate to order, ship and invoice, receive payment, and produce financial statements; all without human intervention. Obviously this is a blue sky dream, but most of the small pieces are in place today, awaiting integration. The number of people needed at each phase is already decreasing. The size of orders is decreasing, too, an indication that inventory reduction is on the minds of customers as well as vendors. Of course, this dream doesn't have to come true in the next 100 years; all manufacturing may just move to Asia instead.

How is this integration going to proceed? As MRP grew from the subsystems within the engineering, production control, purchasing, inventory control, and operations departments of manufacturing companies; and MRP II came from tying MRP to accounting subsystems; CIM will spring from the marriage of MRP II to the numerous other systems in the manufacturing company. In fact, MRP II will be the hub and controller of the manufacturing integration effort.

CIM is not a "package" to be bought off the shelf. Neither is it just a system, a project, a program, or a product available from a vendor. Take only one element, engineering drafting - there are at least 40 vendors of hardware/software. Each of them requires similar information from the users, but no two have exactly the same data items (otherwise there would be no competitive advantage). Even if two of the packages had the same elements, one would have 20 character part numbers and 48 character descriptions and the other would have something different; one would use database, and the other KSAM; one would be COBOL, the other FORTRAN; one would have a parallel printer interface and RS-232 serial input, the other would be completely IEEE-488. Get the picture?

All of the CIM "islands" are this fragmented. Some standards will eventually be accepted, but what can be done in the meantime? Several Fortune 500 companies have begun implementing CIM. The costs of such projects are staggering - complete revamps of existing systems; the standardization of the unintegrated subsystems across the entire organization. It takes a big commitment and lots of dollars and only a few companies have tried to go it alone.

Software vendors attempting to supply CIM are not waiting for standards to be set either. They have no choice but to select one or two vendors for the target functional area and work together to make that "island" talk to their MRP II system.

This is not to say that standardization is impossible or impractical. Several efforts are in progress. Foremost among them is the General Motors MAP (Manufacturing Automation Protocol) task force, which is rapidly developing and had its first Users Group meeting in Spring 1985. MAP addresses the physical connectivity layers with International Standards Organization (ISO) compatible Local Area Network (LAN) specifications. This factory LAN is not to be confused with the baseband LAN using CSMA/CD as specified in IEEE 802.3. GM's factory link is a token access broadband system per IEEE 802.4. Boeing (which uses the former link) and GM are working together to demonstrate both systems in a simulated factory environment at the AUTOFACT show in Detroit in the Fall of '85.

The International Graphics Exchange System (IGES) standard is attempting to allow geometric data to be sent and received on different CAD/CAM systems. The U.S. Air Force is sponsoring Integrated Computer Aided Manufacturing (ICAM) to standardize the framework for the factory of the future and DBMS interfacing. Finally, the National Bureau of Standards (NBS) has an advanced manufacturing facility in Gaithersburg, MD to test and evaluate the integration of planning and control systems (MRP) with automated manufacturing.

Companies involved with CIM today include: General Motors, Intel Corporation, Westinghouse Electric, Caterpillar Tractor, Borg-Warner, Ingersoll-Rand, Hughes Tool, Deere and Co., Cummins Engine, LTV, General Electric, Ungermann-Bass Inc, Allen-Bradley, IBM, Boeing Computer Services, Eastman Kodak, Ford, Chrysler, FMC, Cincinnati Milacron, White-Sunstrand, Cross and Trecker Corporation, Computervision, Autotrol, and Litton. There are, of course, others involved, particularly vendors interested in associating their names with CIM.

The breakdown of the dollar expenditures for the fragments of CIM were like this in 1984: 65% shop hardware (CNC, AS/RS, robots, ATE, data collection); 15% planning and control systems (MRP, MRP II, accounting systems); 20% product/process information (CAD/CAM, CAE, BOM's). A typical manufacturer today might use IBM hardware with IMS data management; TSO or CICS and IBM operating systems with in-house developed business applications in COBOL; DEC VAX, CDC Cyber, Data General, or HP technical computers for CAE; Computervision, IBM, Unigraphics, or Autotrol for CAD/CAM; DG or IBM for data collection; DEC or HP for test/quality/process control; DEC, IBM, or WANG for office automation; Litton, Eaton, or White for AS/RS; plus several voice and data communications networks. None or few of these systems are integrated.

The ultimate goal of CIM is to create the "Custom Shop" - a hybrid of the make-to-order or job shop and the make-to-stock or process shop. The products of a custom shop are made to order, but the factory used to produce them is optimized like a continuous flow or repetitive producer. Only the heavy use of computers can make this possible. The use of FMS cells which are like mini-factories or production centers allow the flexibility required. The networking of suppliers/vendors with assemblers allows further tuning of the timing of receipts - also known as KANBAN (Japanese for just-in-time). High quality is the key to the networks - parts received from a vendor were 100% inspected at the source.

To become a custom shop entails three (3) goals: (1) become a low-cost producer (minimize overhead and labor while maximizing inventory turns and ROI on fixed assets), (2) expanding market share while minimizing inventories through the flexible use of capacity, and (3) expand the product mix and lower the lead time. These sometimes conflicting goals cannot be achieved together without robots, Flexible Machine Systems, Automatic Storage and Retrieval Systems, and Computer Numerically Controlled machines. Most manufacturing companies are aware of the need for these systems (or have them already), but few are thinking of integrating them.

There are no shortcuts to CIM, however. Many predictions were made in the late '70's about how rapidly robots would replace workers in American factories. It has turned out that installing robots and other factory floor equipment should be one of the last steps in a long term policy. According to a Computerworld article of May 27, 1985, "The critical ingredient in an effort to improve manufacturing efficiency is a commitment to enhancing management's knowledge and control over an organization's entire manufacturing process... robots should, in many instances, follow the implementation of manufacturing resource planning systems and the installation of devices that enhance control and communication throughout the factory."

In fact, the state of product and process improvement information in most manufacturing companies is in a mess. The process is not documented; it is in the heads of key people. Drawings are not indexed and are seldom up-to-date. Technical data and specifications, Bills of Material, and configuration data are inaccurate and obsolete. In short, the most vital data in the manufacturing company is unorganized and causing frustration in the company.

The engineer designs a product on one machine, then gives a printout of the parts required to a BOM department, where it is keyed into the MRP system on the main-frame. When a customer orders the new product, a sales order is entered into the Marketing departments order entry system. A work order for the part number is entered and a "kit list" is printed.

Someone keys the part numbers from the kit list into the AS/RS system, which pulls the parts and prints a list of shortages. Someone must key the short parts into the work order's MRP system. When the parts arrive from the vendor, someone on the receiving dock or in data entry keys in the fact that the parts have arrived and are available. An inventory control clerk relieves the shortage. When the vendor's invoice arrives, an A/P clerk matches the PO to the receipt traveler, then keys in the part number, quantity, and cost. Finally, an accountant keys the inventory, WIP, COGS, Sales, A/R, cash, A/P, variances, discounts, payroll, and commissions account totals from the various subledgers into the General Ledger to see if the company made a profit. This is not necessarily a worst case example; some are better and some are worse, but none are automatic. It is not unusual for 10 or more people to key the same part number. There must be a better way!

Solving the physical connectivity problem will not bring the dream of CIM to reality. It is possible to phone Tokyo from Washington, but once the connection is established, the problem of communicating begins - the transfer of information is a lot more than a wire or cable; what is required is a common language and consistent definitions. Therefore, CIM is not a hardware or software (technical) implementation. Unless a company currently has no computers (is there such a company?) or is willing to start all over from scratch, it is not possible today to buy even a few of the pieces of CIM. CIM is a management issue, not a machine. It is a commitment to a consistent plan over a long period of time. It is the definition of a management style which establishes a framework for CIM projects to try to guarantee technical and data content standardization.

Even for General Motors, CIM is a few years away. Their Saginaw Steering Gear plant is the first division to begin implementing CIM with about 5% of the factory floor going into a pilot using the MAP system in the fall of 1985. Think how far away CIM is for the 40,000 or so small manufacturers in the United States. It is the 4000 manufacturing companies with 500 or more employees and the Fortune 500 companies who need to plan now.

As Dan Appleton, President of Dacom, Inc, says, "You cannot buy CIM; what you can buy are islands of automation, integrated by a logo." He means that only a handful of vendors are trying to sell all of the pieces of CIM, and none are succeeding. The best that can be done now is to choose subsystems with a plan to integrate them later - choose vendors of both hardware and software who are committed to CIM.

3011. TRAINING: A KEY TO SUCCESS WITH MICROCOMPUTERS

Ellie East
Media General
Richmond, Virginia

Personal computers, until recently, were bought mainly for homes and small offices. Now, however, due to availability of good business software and advertising campaigns targeted toward big business, corporate purchasing of microcomputers is in full swing.

Unfortunately, one of the key elements to successful use of personal computers is being neglected. That element is user training.

The omission of training has already cost corporations dearly. Many companies have purchased computers, only to have them gather dust because the users lacked the proper background and computing knowledge to use the computers effectively. The novice computer user understands that he will need to learn to use certain software packages. But he does not expect to have to deal with operating systems, file management and design, and, at the least, some elementary systems analysis. He needs training not only on software packages, but also on the basics of data processing.

Too frequently, the users' only training sources are hardware and software manuals, which are often poorly written, and sometimes even contain erroneous information. Digging through manuals is a time-consuming, frustrating process that does not provide appropriate training for the new user. Lack of proper training results in low morale and poor computer use, and in negative investment return for the company.

User training expenditures are vital to successful use of microcomputers, and should be included in corporate budgets as a matter of course.

TRAINING OPTIONS

Any training offered the computer user should be hands-on. Given this assumption, there are several training options open to an organization.

- The first is to designate a person within the company to conduct one-on-one training for each person receiving a computer. Though better than nothing, this type of training is undesirable, for the following reasons.

The training is usually conducted in the office of the user, and will be continually interrupted by phone calls, walk-in traffic, and "emergencies" that the user has to deal with.

Because he is in his own office, his mind will be on his work - not on learning. Another disadvantage to this option is inefficient use of the trainer's time, since he could train as many as ten or twelve people simultaneously in a formal classroom session.

- The second option is sending the user to remote training. There are many training sessions available, usually within a reasonable distance of the user. This is preferable to the one-on-one option, because it is conducted out of the user's office and is of a formal nature, but there are many disadvantages, the most obvious being the expense and inconvenience of travel. Another drawback is that the computers used in these sessions may be different from the one the user has in his office. The third negative factor is that the classes are global by definition, and therefore cannot be geared to individual needs. Lastly, but possibly most importantly, there is no assurance of quality training.

- The third, and most desirable alternative, is formal, in-house, classroom training. It has one disadvantage, and many advantages.

The disadvantage is the cost, in dollars and personnel, to facilitate a formal training program.

The advantages are many. First, the company has control of the training, with accompanying quality assurance. Second, training can be geared directly to the company's needs. Third, formal classroom training on company premises removes the expense of travel, but provides a private place away from phones and other distractions. Fourth, training is done on the same computer the user has, and the software versions are the same. Thus, when the user completes the training course, he can immediately apply what he has learned. The fifth advantage is the benefit of a company trainer who is available after classes for followup assistance.

SETTING UP AN IN-HOUSE TRAINING PROGRAM

In order to implement a training program within a company, there are some questions that must be answered.

1) Who are the potential students?

The most obvious potential students are those who are slated to receive computers. Realistically, however, due to shared computer use, there will probably be about one-and-a-half to two people needing training for each computer purchased.

2) What are the data processing/computing knowledge levels of potential students?

The answer to this question determines the level of basics to be taught. If a person has used a personal computer, or if he is familiar with mainframe end-user computing, he obviously will not

need the same level of training as the person who has had no experience at all. Knowledge levels will likely be mixed, so that the basics level has to be pitched toward the low end.

3) What do people expect from their computers?

The tasks expected of a business computer are usually of two main types: word processing and spreadsheet processing. The types of tasks determine which software packages will be purchased, and therefore which ones must be taught.

4) When will the students receive their computers?

Timing of training sessions is critical. Classes should be repeated at intervals that coincide with expected computer purchases. Two weeks is the maximum time that should elapse between training and access to a computer. Otherwise, the training will have been wasted.

5) What should be taught?

- First, every student should be exposed to the basics of computing, such as definitions of terms (hardware, software, operating systems, discs, files, file structures, random access memory, printers). There should be a brief explanation of how these components work together. This section should also include the care of discs and the computer, an explanation of software copyright laws, and the importance of establishing security measures.

- Second, the student should get plenty of practice with such elementary procedures as inserting and removing discs, turning the computer and printer on and off, and loading the operating system. These procedures should be repeated often during the basic training, because the repetition will help the student later when he is alone with his computer.

- Third, file management procedures should be discussed and demonstrated with hands-on exercises. Because file maintenance is a difficult concept for the novice user, formatting discs, and copying, renaming, and deletion of files should be thoroughly covered.

- Fourth, the computer user should be trained on the software packages he will be using. The shorter, simpler software can be included in the basic training course. More complex software should be offered in a separate course. If the software is very complicated, an advanced course should be offered. The presentation of material should be orderly. A package can be taught by starting out with the simple functions, and then moving to the more advanced ones. Or, if the software lends itself, the course can be divided into logical functions. For example, an integrated package could be divided into spreadsheet, graphing, and database.

- In all cases, the hands-on portion should cover enough that the student can go back to his office and use the software with little trouble. On the other hand, it is easy to overwhelm the beginning user. It is usually undesirable to cover every single command or function of a software package, unless the package is extremely simple.

6) What are the necessary teaching materials?

Teaching manuals should be created. They should contain the important points, plus hands-on exercises that demonstrate the points. They should not contain excess verbiage. The trainer should use the points in the manual as beginnings for more lengthy explanation. The manuals can contain a glossary to be read later, and optionally, exercises that the student can do at his leisure. The student should be able to keep his manual after the class.

The hands-on exercises should be short enough that the student will not become bored, but long enough for him to get the point. Also, the exercises should contain explanations of why the student is doing a particular step. Otherwise, he will merely press keys and push buttons, without understanding the point of the exercise. The exercises must be correct to the last detail. The student should not be expected to second-guess an incorrectly worded exercise. The exercises should be thoroughly tested before they are used in a class.

Most exercises should be accompanied by a disc file that goes with the exercise. The file, previously created by the instructor, should simply demonstrate the points, and should be no more or less complex than necessary. Reading and writing of files should be included with almost every exercise, even if not mandatory under normal operations. Reason: Repetition of file handling commands helps the student later. Also, because students need the experience of creating, saving, and retrieving their own files, these functions should be liberally included in the exercises. If a new user realizes that he can create and retrieve his own files, it will go a long way toward establishing his confidence in the computer system.

7) Who will do the training?

Because of the data processing functions that need to be taught, a strong argument can be made for the trainer, or trainers, to have a data processing background.

The trainer will need certain abilities. The most obvious one is the proper knowledge. Next, the trainer has to know how to present the material in a non-technical manner, and possibly he will have to create his teaching materials. Also, a trainer in this situation is, in many respects, a salesperson. It will be vastly to the company's benefit if students can not only be taught, but can also emerge from the classes excited about what

their computer can do for them. Lastly, the trainer must have a positive attitude toward his students. If the students get the idea that the trainer finds them lacking because they have little data processing knowledge, it will effectively kill the training program.

THE PHYSICAL FACTORS

There are some physical mechanics of a training program that must be considered. They are: the number of people in a given class, classroom equipment, class length, teaming considerations, and scheduling.

1) What equipment is needed in the classroom?

Students work well in teams of two at one computer. Therefore, there should be a computer/printer combination per team. There should be room enough for two people to sit comfortably at a computer/printer, and there must be table space for the teaching manuals. The instructor will need a blackboard or a flipchart, and telephones should be banned from the classroom.

2) How many people can be taught in one class?

In any given class, there will be varying ranges of knowledge and varying abilities to "catch on". Therefore, especially during the exercises, some students finish quickly, while others need more time. If there is only one instructor, the maximum number students should be ten or twelve. If there are two instructors in a class, the number of people can be raised to about twenty. Above this, there is too much variance in abilities and levels.

3) What is the ideal time length for a class?

The total length of a class on a given day should not exceed six hours. Three hours in the morning and afternoon (with a break in each) is the maximum length of time that people seem to concentrate. It is also helpful, since people are on company premises, to give extra time at the lunch break (1 1/2 to 2 hours). If this is not done, students often will take the extra time anyway (sometimes because they must), to check in with their office.

4) How should classes be scheduled?

Schedules should be geared to the computer acquisition schedule, bearing in mind the two-week maximum between training and computer access. Schedules, which should include class descriptions and prerequisites, should be distributed at least two weeks before the classes. Mondays and Fridays should be avoided, as should the first few days of a month if there are accounting people to be trained.

5) Who can work together in a class as a team?

If possible, a democratic approach to classes should be maintained, meaning that private "VIP" training sessions should be resisted. This reduces the number of classes, and therefore the demands on the trainer's time.

TYPICAL TRAINING PROBLEMS

- The biggest problem is that of varying knowledge levels on the part of the students in a given class. The slower students should be given as much attention as possible. Also, the slower person could possibly be teamed with a faster one. This is a situation where a second instructor is of great help.

- Occasionally, a person will be forced by management to attend a class. Sometimes he will resist the class because he feels he is too busy for it, or because he is already familiar with computing, and thinks that the class will be a waste of his time. When this happens, he has a negative starting attitude. Sometimes it can be reversed by paying him extra attention, or by stressing the good points of the computer used in the class, without degrading others.

- The trainer should be prepared for the unexpected classroom situation. At times, there will be an unscheduled, extra student. The trainer should come to the class with at least one extra set of teaching materials, including a set of master discs containing exercise files. Students will occasionally accidentally wipe out a file on a disc. The trainer should be prepared to quickly make copies of destroyed files. Students will sometimes ask questions that do not relate directly to the class, but do relate to their work. The trainer should have enough background to be able to answer these questions, or to find an answer shortly.

AFTER TRAINING

After the training sessions, students will go back to their computers, which are hopefully on their desks, and try to use them. Typically, there will be a period of about two weeks when the students will need a lot of help. Their questions will run the gamut from simple to complex. After this initial period, the combination of training and computer use seems to gel, and the barrage of beginning questions will stop.

Former students will continue to need counseling and advice, however. A person, or a group of people, should be designated to help them. The help needed will vary per individual, but it will probably be needed indefinitely.

The best way to tell how effective training has been is simply to stroll through former students' offices. More than likely, especially if they have been asking questions, their

computers will be in use, and many will want to show off their work. If this happens, the training program has been successful and beneficial to both students and company.

In conclusion, the expenditures for office automation should include outlays not only for hardware and software, but also for formal training. Money spent on an in-house training program pays off in fast, efficient use of computing tools. Money not spent in this way will cost an organization in the long run.

3012. RATIONAL STRUCTURING TECHNIQUES FOR COBOL II/3000 MAINTAINABILITY

James T. McDermott
HP-3000 Independent Consultant
60 Clarendon Avenue
Yonkers, New York 10701
Phone: (914) 964-9225

COBOL is a tedious language to code in. It is also the most widely used one in industry for the last twenty years. This seems paradoxical. The truth of the matter is that even though COBOL may be tedious to write, its reason for success is simply that it uses the English language for syntax and therefore can be "read" by almost anyone with a very minimal amount of data processing background. However, like English, it demands decent "grammar" to be understood. Anyone who just learned English or speaks poor English, may have a difficult time communicating with people in general. This is because either erroneous or incomplete education or just plain sloppy attitudes are at fault. The parallel I am trying to draw is that certain minimal guidelines must be followed to effectively "communicate" your intentions whether in speaking or writing COBOL code. This is especially true with COBOL code because programs can have a life expectancy of possibly up to 10 years. During that time, the code has to be maintained which is the heart of the problem in data processing today. If a different programmer each year was responsible for the maintenance of a complex posting of accounts receivables program there could conceivably be a vast difference in the way that program looked and executed by the time the tenth programmer worked with it. If some reasonable techniques (or standards) were instituted by the programming department at the beginning of programming development, many problems could be eliminated or at least minimized.

Some of the benefits would include:

- readability - reliability - efficient code execution - maintenance made easier - more efficient use of programmers' time - better working relations between programming department and end users

Readability means that the code doesn't overwhelm you when looking at. By this I mean that it has a certain pleasing to the eye nature that a cluttered or poorly formatted program doesn't possess. At least to me, cluttered code usually has the psychological effect of turning me off from a maintenance task or causes me to think about re-writing an entire program.

Reliability means the code does what it was intended to do both before and after any changes were made. Any maintenance changes are clearly visible by use of comments.

Efficient execution of code means less code swapping will occur when a program is properly segmented (sectioned in COBOL). It also means code sizes are kept small (usually less than 8K) for better response time when swapping does occur.

Maintenance is made easier because all programs should look the same and use the same basic constructs which were established during the initial programming development. Also, a programmer's time is spent less on trying to decipher spaghetti garbage written in the stone ages and more on doing a larger quantity of tasks in a shorter time frame with better results.

All of this can indirectly precipitate better working relationships between the programming department and the end users. If there is little or no down time due to bug free applications running, then the programmer/user relationship will benefit. The users will stop blaming the programming department for causing them to work overtime when an application has a bug that takes many hours to locate. The programmers will stop perceiving the users as unknowledgeable about computers and the programmers will better understand what the users job entails.

What are rational structuring techniques. Roget's Thesaurus has many synonym definitions for the word rational. I think the best definition is the word reasonable. Reasonable means not using structured methodologies as a bible with rules and regulations that cannot be relaxed. I worked in the manufacturing and materials systems area of a large aerospace corporation in Palo Alto, California. There, structured methodologies according to one of the most famous proponents in the "science" were forced upon us without mercy. I had to take an intensive 4 week seminar on top down design and structured programming according to the aforementioned proponent. After the course, I was "encouraged" to read a huge 600 page manual/bible which espoused the data processing world according to Since I had previously read books on several other schools of thought on the subject, I came to the conclusion that no one way is the right way. Each "school" has its own quirks about the subject. Structured methodologies are tools, not bibles to adhere to with blind faith. I remember spending many sessions with a senior programmer/analyst walking through some new code I had written. He questioned practically everything in the code concerning technique and structure. Was this or that paragraph functional and does it adhere to the half dozen or so tests for functionality? Should this piece of code be moved up to a higher level since it was making a decision? Is it not better to Perform this function rather than go to a paragraph with the EXIT statement. After 3 or 4 days of this, I started to dread working in an environment where the subjectivity of my code's degree of structure and functionality was scrutinized with a fine tooth comb. Of course, if some other senior programmer/analyst reviewed the same code with me a week later after making all the changes suggested by the first senior programmer/analyst, the second person would start all over again with his/her interpretations of what is and isn't a structured COBOL program.

All of this is clearly IRRATIONAL for one simple reason. It wasted an incredible amount of time and delayed the task at hand, which was to write a workable and maintainable COBOL program in a timely manner. It also caused for poor working relationships within the department in a profession where sometimes huge egos abound. Sure, the code produced under these circumstances was structured and maintainable, but at what price?

Before, I begin to talk about rational structuring techniques for COBOL II/3000, I would like to cover some of the points I consider to be important in relationship to my first benefit, readability. There are quite a few "techniques" I have been using to improve the readability of COBOL code. I will describe them now:

1. Use abundant WHITE SPACE, COMMENTS, and PAGE EJECTS (\$PAGE with a comment). This is especially important in the PROCEDURE DIVISION. In this division, "verbs" (OPEN, CLOSE, READ, WRITE, PERFORM, IF, COMPUTE, MOVE, etc.) can occupy code positions 12 through 23 and data names or literals that are the "object" of the "verbs" can occupy code positions 24 through 72. This is somewhat analogous to any assembler type code where OPCODES and OPERANDS must reside in specific columns.
2. Precede paragraph headers by a line of 32 asterisks. These are used for 2 reasons. The first reason has to do with program development. If you enter a line of 32 asterisks into one of the soft keys on an HP26xx terminal, the soft key can be used to add the asterisks into the code and be used as a "template" to build the paragraph header on the next line. Since a paragraph header can only contain a maximum of 30 characters, the 32 asterisks define the boundaries of the header (1 asterisk in column 7 sets up the line as a comment and the 32nd asterisk designates the position of the period at the end of the largest size header). The second reason is READABILITY. The paragraph headers will jump out at you from the text. Also, leave a blank line after each paragraph header. Here is an example of items 1 and 2:

....+....10...+....20...+....30...+....40...+....50...+....60...+....70.

\$PAGE "PRINT BATCH TOTALS"

C300-PRINT-BATCH-TOTALS.

PERFORM C310-FORMAT-BATCH-TOTAL-LINE.

PERFORM C320-PRINT-BATCH-TOTAL-LINE.

C310-FORMAT-BATCH-TOTAL-LINE.

MOVE	"BATCH"	TO	TOTAL-TYPE.	MOVE
BATCH-TOTAL-DEBIT-AMTS	TO	TOT-DEBIT-AMTS-PRT.	MOVE	
BATCH-TOTAL-DEBIT-ITEMS	TO	TOT-DEBIT-ITEMS-PRT.	MOVE	
BATCH-TOTAL-CREDIT-AMTS	TO	TOT-CREDIT-AMTS-PRT.	MOVE	
BATCH-TOTAL-CREDIT-ITEMS	TO	TOT-CREDIT-ITEMS-PRT.	MOVE	

C320-PRINT-BATCH-TOTAL-LINE.

IF LINE-COUNTR > 58 PERFORM C1XX-PRINT-HEADINGS-RTN.

MOVE	TOTALS-LINE	TO	SUBR-WRITE-RECORD-AREA.
MOVE	2	TO	SUBR-PRINT-CODE-NUM.
PERFORM	C1XX-WRITE-RPT0022-REPORT.		

```

MOVE          SPACES          TO SUBR-WRITE-RECORD-AREA.
MOVE          1                TO SUBR-PRINT-CODE-NUM.
PERFORM       C1XX-WRITE-RPT0022-REPORT.

ADD           3                TO LINE-COUNTR.

```

3. "IF" statements can be nested and indented 4 spaces for up to 3 IF's with corresponding "ELSE" statements (each on its own line), aligned directly below each IF. If you need to nest more than 3 "IF's", there is no problem if they start to occupy space past the 23rd code position. Remember, all of this is for rational and readable code, not iron clad rules and regulations. Also, any code that contains "verbs" should be indented 4 spaces if the code serves as a true condition for an IF statement or is it's false condition after an ELSE statement. Here are some examples:

....+....10...+....20...+....30...+....40...+....50...+....60...+....70.

```

IF          EVENT-CHECK-MM-IN          =          SPACES          IF
EVENT-CHECK-DD-IN = SPACES IF EVENT-CHECK-YY-IN =
SPACES NEXT SENTENCE ELSE MOVE 37          TO
FORMS-RETURN-CODE MOVE "CHECK MONTH & DAY MUST BE ENTERED
IF YOU E - "ENTER A CHECK YEAR!" TO
FORMS-WINDOW-MSG ELSE IF EVENT-CHECK-YY-IN = SPACES MOVE
37          TO FORMS-RETURN-CODE MOVE "CHECK MONTH &
YEAR MUST BE ENTERED IF YOU - "ENTER A
CHECK DAY!" TO FORMS-WINDOW-MSG ELSE MOVE 37
TO FORMS-RETURN-CODE MOVE "CHECK MONTH MUST BE ENTERED IF
YOU ENTER A - "CHECK DAY & YEAR!" TO
FORMS-WINDOW-MSG ELSE IF          EVENT-CHECK-DD-IN = SPACES
IF EVENT-CHECK-YY-IN = SPACES MOVE 38          TO
FORMS-RETURN-CODE MOVE "CHECK DAY & YEAR MUST BE ENTERED
IF YOU EN - "TER A CHECK MONTH!" TO
FORMS-WINDOW-MSG ELSE MOVE 38          TO
FORMS-RETURN-CODE MOVE "CHECK DAY MUST BE ENTERED IF YOU
ENTER A C - "HECK MONTH & YEAR!" TO
FORMS-WINDOW-MSG ELSE IF EVENT-CHECK-YY-IN = SPACES MOVE
40          TO FORMS-RETURN-CODE MOVE "CHECK YEAR MUST
BE ENTERED IF YOU ENTER A - "CHECK
MONTH & DAY!" TO FORMS-WINDOW-MSG.

```

4. Tabulation should be used in each division. In WORKING-STORAGE, for example, the following describes how data can be aligned by setting TABS on an HP26xx terminal in conjunction with the editor of your choice that will eliminate keying in spaces in many instances.

....+....10...+....20...+....30...+....40...+....50...+....60...+....70.

```

01 PAGE-COUNTR          PIC S9(04) COMP. 01
LINE-COUNTR          PIC S9(04) COMP. 01 WS-DATELINE
PIC X(27).

01 HOLD-EXTRACT-DEPT-ID. 05 HOLD-EXTRACT-AFFILIATE          PIC
X(02). 05 HOLD-EXTRACT-CHAPTER          PIC X(02).

```


Notice that level numbers and picture clause repeat factors start with a zero if they are a single digit, and all level numbers, data names, and the PIC sizes are aligned on an even column number boundary. Any PIC that is a signed numeric field of any type is the exception. This makes it easier to see signed fields immediately.

5. I usually separate "logical" groupings (MOVES to a common buffer for example or a routine that does a particular function) within a paragraph by a blank line, followed by a "COMMENT" line of 1 asterisk preceded by 16 hyphens and then another blank line. Here is an example:

```
..+...10...+...20...+...30...+...40...+...50...+...60...+...70.
```

```
*-----
* PRE-1984 FOUNDATION ENTERED SHORT ACCT NOS.
*-----
```

```
IF          SYSTEM-DATE < 19840101 IF          SA-ACCT-NO =
"4202060047" IF (EVENT-DOLLARS-IN > "    600" OR = "
600" ) MOVE SA-ACCT-NO TO EVENT-ACCOUNT-NO-IN GO TO
A115-EXIT ELSE PERFORM Z700-INVALID-SHORT-ACCT-NO-RTN MOVE
"INVALID SHORT ACCT NO: USED ONLY FOR S -
"HARED GIFT GREATER THAN OR = $600" TO FORMS-WINDOW-MSG GO
TO A115-EXIT.
```

```
*-----
```

```
IF          SYSTEM-DATE < 19840101 IF          (SA-ACCT-NO =
"4202060045" OR "4202060046" ) IF EVENT-DOLLARS-IN <
"    600" MOVE SA-ACCT-NO TO EVENT-ACCOUNT-NO-IN GO TO
A115-EXIT ELSE PERFORM Z700-INVALID-SHORT-ACCT-NO-RTN MOVE
"INVALID SHORT ACCT NO: USED ONLY FOR S -
"HARED GIFT LESS THAN $600" TO FORMS-WINDOW-MSG GO TO
A115-EXIT.
```

```
*-----
```

```
IF          SYSTEM-DATE < 19840101 IF          SA-ACCT-NO =
"4201040038" IF (EVENT-DOLLARS-IN > "    600" OR = "
600" ) MOVE SA-ACCT-NO TO EVENT-ACCOUNT-NO-IN GO TO
A115-EXIT ELSE PERFORM Z700-INVALID-SHORT-ACCT-NO-RTN MOVE
"INVALID SHORT ACCT NO: USED ONLY FOR U -
"NSHARED GIFT GREATER THAN OR = $600" TO FORMS-WINDOW-MSG
GO TO A115-EXIT.
```

Now I will describe what I consider to be reasonable alternatives to irrational or archaic coding methodologies:

All COBOL program should contain 5 basic procedural code elements:

- Main Line Logic - Initialization - Central Processing Loop - Termination
- Fatal Error Handling

1. Main Line Logic is the first paragraph in the PROCEDURE DIVISION that PERFORMS the Initialization, PERFORMS the Central Processing Loop, and PERFORMS the TERMINATION. Note, I do not go back to the Main Line Logic if any fatal error occurs for reasons I will explain later. Here is an example:

.....10.....20.....30.....40.....50.....60.....70.

A100-VOTESTAT-MAINLINE-LOGIC.

```

PERFORM      B100-INITIALIZATION-ROUTINE.

PERFORM      C100-PROCESS-MEMBER-XTRCT-FILE      THRU  C100-EXIT
UNTIL  EOF-MEMBER-XTRCT-FILE.

PERFORM      D100-TERMINATION-ROUTINE.

STOP        RUN.

```

2. Initialization is the second paragraph that should open all files, and in WORKING-STORAGE, initialize all numeric fields to zeros and all alphanumeric fields to spaces. It should display some banner or header with the program's name and purpose. It should accept or read in and edit any parameters used by the program that vary each time the program is run, and terminate the program if an erroneous parameter is detected. It is assumed the input parameter is entered as data in line in a stream job. If a sequential file is read as input, an initial read of the file should be done. This is very important so that the WORKING-STORAGE buffer is "primed" with a record that can be processed immediately by the Central Processing Loop.
3. The Central Processing Loop contains one PROCESS AND READ LOOP paragraph that is cycled through repeatedly until a terminating condition occurs. Many other paragraphs can be performed from the PROCESS AND READ LOOP paragraph but control is always returned to it. During this element of the code, certain functions should be accomplished:
 - a) counts should be kept of records read, written, in error, and statistical breakdowns of record categories.
 - b) appropriate error messages (Fatal as well as non-Fatal) should be displayed either on the system console or in the \$STDLIST; they should include the data item(s) causing the error to aid in determining the problem.
 - c) if a program is to execute for a very long time (more than a day), add a counter to display a message at the system console as to how far it has progressed with the number of records processed so far (eg. display a message for every 1000 records processed). This is good for a new program in determining run time statistics.
4. The Termination paragraph closes all files and displays any termination messages with statistics kept during program execution.

5. The Fatal Error Handling paragraph is PERFORMED immediately after an error is detected in the code, followed by a PERFORM of the Termination paragraph and then a STOP RUN. I do not believe there is a need to set a "switch" when an error is detected. The switch would have to be passed upward through the code through possibly numerous other paragraphs, just so it could reach the Main Line Logic paragraph. The routine is going to be performed only once. Why clutter every routine that is PERFORMED up above the routine in which the error occurred, with a check for a switch that detects a fatal error occurrence. It is unimportant that the lower level code is making a decision to terminate the program.

Some of the basic constructs of the PERFORM "verb" that should be used are as follows:

PERFORM paragraph THRU paragraph-exit.

This is a perfectly acceptable form of a top down structure. It has one entry and one exit point. The exit point can also be reached from within the construct (which may contain more than one paragraph) by use of the "dreaded" GO TO statement. This is a reasonable top down structure that is very easy to use. You do not have to keep jumping into another PERFORM or clutter your code with an abnormal number of switches just to avoid using a GO TO statement. Here is an example which includes the concept of the Central Processing Loop described above:

.....10...+.....20...+.....30...+.....40...+.....50...+.....60...+.....70.

```
01 WS-MEMBER-XTRCT-REC          PIC X(80).

01 EOF-MEMBER-XTRCT-FILE-SW      PIC S9(01).      88
   EOF-MEMBER-XTRCT-FILE        VALUE 1.        88
   NOT-EOF-MEMBER-XTRCT-FILE     VALUE 0.
```

```
PERFORM C100-PROCESS-MEMBER-XTRCT-FILE THRU C100-EXIT UNTIL
   EOF-MEMBER-XTRCT-FILE.
```

.....

.....

```
$PAGE "PROCESS MEMBER EXTRACT FILE"
*****
```

```
C100-PROCESS-MEMBER-XTRCT-FILE.
```

```
IF          EXT-MEMB-DEPT-ID = HOLD-EXTRACT-DEPT-ID PERFORM
   C110-ACCUMULATE-TOTALS PERFORM C10X-READ-MEMBER-XTRCT-FILE
GO TO      C100-EXIT.
```

```
*-----
```

```
MOVE          HOLD-EXTRACT-DEPT-ID TO DTL-CHAPTER.
```

*-----

```

IF      EXT-MEMB-AFFILIATE = HOLD-EXTRACT-AFFILIATE MOVE
EXT-MEMB-DEPT-ID      TO      HOLD-EXTRACT-DEPT-ID PERFORM
C200-PRINT-CHAPTER-TOTALS                                PERFORM
C230-INITIALIZE-CHAPTER-ACCUMS                            PERFORM
C110-ACCUMULATE-TOTALS PERFORM C10X-READ-MEMBER-XTRCT-FILE
GO TO   C100-EXIT.

```

*-----

```

MOVE      HOLD-EXTRACT-AFFILIATE TO AFF-SUB-AFFILIATE. MOVE
EXT-MEMB-DEPT-ID      TO
HOLD-EXTRACT-DEPT-ID. PERFORM
C200-PRINT-CHAPTER-TOTALS.
PERFORM
C230-INITIALIZE-CHAPTER-ACCUMS.
PERFORM
C240-PRINT-AFFILIATE-SUBTOTALS.
PERFORM
C270-INITIALIZE-AFFIL-ACCUMS.
PERFORM
C110-ACCUMULATE-TOTALS.
PERFORM
C10X-READ-MEMBER-XTRCT-FILE.

```

C100-EXIT.

EXIT.

C10X-READ-MEMBER-XTRCT-FILE.

```

READ      MEMBER-XTRCT-FILE INTO WS-MEMBER-XTRCT-REC AT END
MOVE 1    TO EOF-MEMBER-XTRCT-FILE-SW.

```

Another construct of the PERFORM "verb" eliminates the explicit incrementing and decrementing of an index, subscript, or other numeric item. It's structure is as follows:

```

PERFORM paragraph THRU paragraph-exit VARYING identifier-1 FROM
identifier-2 BY identifier-3 UNTIL condition(s)

```

Note, that this is the simplest format of the VARYING option because only a single identifier is being varied. The UNTIL causes a loop to execute until a condition such as EOF is met. The condition is checked each time before executing the PERFORM. Also, the conditional test should be constructed as an 88 level switch in WORKING-STORAGE for READABILITY and be initialized in the Initialization element of the code. Suppose, you need to print a report of all the records in a master file of donors who contribute to a charitable organization but only those donors who give more than \$250 in a calendar year. The other criteria is to sample the first 1000 records in the master file. This adds a second condition to be

tested for. The counter for the first 1000 records is automatically augmented by the PERFORM statement. The following code will accomplish this:

```

.....+.....10...+.....20...+.....30...+.....40...+.....50...+.....60...+.....70.

01 WS-DONOR-MASTER-REC          PIC X(80). 01 MF-REC-COUNTER
PIC S9(09) COMP.

01 END-OF-DONOR-MF-SW          PIC S9(01). 88
END-OF-DONOR-MF              VALUE 1. 88
NOT-END-OF-DONOR-MF          VALUE 0.

PERFORM      C10X-READ-MASTER-FILE.

PERFORM      C100-PROCESS-MASTER-FILE THRU C100-EXIT VARYING
MF-REC-COUNTER FROM 1 BY 1 UNTIL MF-REC-COUNTER >
1000 OR END-OF-DONOR-MF.

```

C100-PROCESS-MASTER-FILE.

```

IF      WS-DONOR-AMOUNT      >      250      PERFORM
C110-WRITE-HI-DOL-DONOR-REPORT.

MOVE      .....
.
.
.

PERFORM      C10X-READ-MASTER-FILE.

```

C10X-READ-MASTER-FILE.

```

READ      DONOR-MASTER-FILE INTO WS-DONOR-MASTER-REC AT END
MOVE 1      TO END-OF-DONOR-MF-SW.

```

In closing, I would like to emphasize some miscellaneous items that are of major importance in COBOL on the HP-3000.

- SEGMENTATION (or use of SECTIONS in COBOL) - COPYLIBS

Segmentation is essential on the HP-3000. The object of segmentation is efficient code execution by keeping code segments small (less than 8K) and keeping a segment resident in main memory for as long a time as possible to avoid continuous swapping. The problem is more of a logical one than anything else. Everything in your program, up to and including the Initialization code element should be kept in a SECTION. Next, the Central Processing Loop code element should ideally be kept in a second

SECTION, even though it isn't always possible (especially if it has many paragraphs being PERFORMED). The Termination and Fatal Error Handling code elements should be kept in their own SECTIONS. In the Central Processing Loop, if a serial read is being executed (an IMAGE data set for example), I would definitely place the serial read being PERFORMED in a segment of its own. Common READ and WRITE routines should be PERFORMED and kept at the end of the Central Processing Loop code element.

COPYLIBS are most essential for IMAGE, KSAM, VPLUS/3000 and flat file buffer descriptions so that if any of the buffers increase or decrease in size, only the programs affected will need to be re-compiled rather than laboriously making editor changes to hard coded buffers in WORKING-STORAGE. Also, commonly used PROCEDURE DIVISION code can be added into the COPYLIB.

Although most of my techniques are not necessarily new, they do present a precise and clear method of writing COBOL that makes all programs maintainable because they will all look the same, are easy to follow and therefore are easy to change.

3013. How to Design For the Fourth Generation

Leigh Solland
Cognos Corp.

I. Overview: why be concerned with data design?

Data processing has come a very long way in its relatively brief history. We have, in our own experience, moved from unit record machines and tabulators to on line, interactive, real time application systems. The machines have become faster, smaller, less expensive and more powerful. The programs have become more sophisticated, the programmers more plentiful, and the applications more varied.

Why, then, are we still struggling with systems analysis by "best guess"? Why are we designing data bases which are slower, harder to program for and more complex to understand than we would like? Why is it that any change in the operation of our employer's organization or way of doing business causes major upheaval in the data structures?

In a more positive light, why do one person's database designs seem to be inherently "cleaner" than another's? Why do some designs seem to weather change better than others? Is there some way to generalize and formalize these "lucky" designs, so that they may be applied on a wider scale?

A. More timely application systems

It is a tradition of application systems development that the final product never gets done, and the compromise product is typically late, over budget and under powered. Many of the vendors at this conference are in business because they have discovered some way to help alleviate this problem.

There are many approaches to speeding up application development. Some are managerial, such as the use of development methodologies to ensure that nothing gets missed, and some are technical, such as Fourth Generation Languages (4GL's) and programming utilities.

It is the contention of this paper that simplified, formalized data design early in the development cycle will greatly speed up application development because it:

1. provides a focal point for the systems analysis phase of the project, ensuring that the whole system is understood and that its future use has at least been considered;
2. reduces the complexity of the programming effort by simplifying the access paths, eliminating unnecessary redundant data, and ensuring the presence of required data;

3. lends itself to the heavy use of modern programming techniques and tools such as 4GL's. As well as providing their own inherent gains, these tools lead to smaller programs, which alone will speed up the programming task.

B. More stable application systems

Much of the work done by system development people today is maintenance of existing systems. Estimates range from 50% to over 80% of available resources being spent on maintaining existing systems, rather than building new ones. The maintenance activity is due to three factors:

1. Bugs. These are strictly due to bad programming, and may be addressed by using a 4GL, by structured techniques and by quality control techniques such as walkthroughs, but not directly by data design;
2. New user requirements. These stem from the fertile imaginations of the end users who, no matter how we try to keep them stifled, seem to keep coming up with new ways to use their data;
3. System shortcomings. This may be due to missing something in the analysis phase, but is more likely due to changing external conditions.

A truly flexible data base design can help in this third area, accommodating modifications and additions to the types of data stored by the system. A rigid design necessitates not only periodic major overhauls in the data structures as the world changes, but also frequent changes to the programs accessing that data.

C. More maintainable application systems

Some maintenance of systems is unavoidable, even if your data structures are all perfectly flexible and you have your users writing all their own reports. The problem is that the programmer maintaining the system is probably:

1. not the person who wrote the system in the first place;
2. young and inexperienced;
3. under pressure to get the changes completed immediately so that the payroll can be run!

To help this person satisfy the program maintenance requirements quickly and accurately, the data design must be understandable. Any dependencies should be clear and obvious to avoid the "ripple effect" of applying a change to one place but neglecting to apply it to another.

One of the most important considerations for maintenance is to ensure the independence of the data from the programs. This is more commonly implemented now in an environment of data bases and on line systems than it was with batch oriented systems, where each program had its own data files. However, even with data base technology available, it is up to the designer to use it.

D. More useful application systems

Isolated data is useless. Unobtainable data is useless. Untimely data is useless. In fact, by definition, information is useful data. Therefore, data itself is useless until it is turned into information.

We in the data processing world have two main functions. We provide ways of collecting and storing data, and we provide ways of transforming it into information.

In the past decade, we have learned that the best way to get the information to the users is to let them get it themselves. For this, we need not only the hardware and software to get it to them, but data structures which can be:

1. deciphered;
2. related and re-related in ever changing and unforeseen patterns;
3. utilized by multiple users, each of whom has a different view of the organization, different concerns and different information requirements.

II. Advantages of Fourth Generation technology

Computer programming is undergoing a change now which is as revolutionary and significant at the adoption of Cobol. Fourth generation languages have proven to be the solution to getting more systems up faster and cleaner, with relatively fewer skilled, experienced programmers.

The generations of programming languages may be traced from the first generation, machine code, through the second, mnemonic assembler languages, to the third, the so-called "high level" languages such as Cobol, Fortran, Algol and their offspring. Each generation is characterized by an order of magnitude improvement in productivity. Each has also been characterized by a great division between those who adopted the benefits of the new technology and those who were reluctant to change from the proven solution.

Fourth generation languages have appeared on the market over the past several years. They make their profound productivity gains by using such techniques as nonprocedural code, very high

level syntax and active data dictionaries to store information for the programming languages.

A. Faster development

Use of a 4GL can cut the time for development of a system by a factor of 10 over traditional languages, such as Cobol.

This means that systems are up before the user gets fed up and frustrated with the data processing department, that new systems are running before the external conditions of the world are so changed that the system is obsolete, and that the user who requested the system may get to see it before retiring, changing positions or leaving the organization.

It means that prototyping is a real consideration, instead of a futile dream. A prototype system may be brought up quickly, given to the users to work with in a pilot environment, and modified before final release to the entire community.

Faster development means that not only can the backlog of applications waiting for development be addressed, but that the "invisible backlog", the systems that the users don't even have the nerve to ask for, can begin to be identified. This will do nothing to hurt the credibility of the data processing people in an environment where the users expect (justifiably) that the computer should be able to do everything they need it to do, and to do it NOW!

B. Reduced maintenance

People usually start using 4GL's because of a desire to speed up the initial development of systems. However, the tenfold improvement in development time pales before the gains in program maintenance.

Consider that a program written in a 4GL has a greatly reduced amount of code. It is a lot easier to debug or modify a program when you can see the whole program on one or two pages (or even on one or two screens).

Consider that a 4GL usually handles the complex parts of programming, such as file inputs and outputs, screen handling and record retrieval. This completely removes the most complex, bug prone parts of the code.

Consider that a system written in a 4GL is not nearly as sensitive to data changes as a system written in an older language, especially if the data design has been well thought out and a data dictionary has been used in the more modern system.

All of these serve to reduce the maintenance effort.

C. Increased programmer productivity

When a programmer is working on something interesting, that programmer is more productive than when he (or she) is bored. Faster systems development means the programmer can be on to bigger and better things quickly. Job satisfaction is enhanced.

When a programmer is getting good, positive feedback from satisfied users, he is not only more productive but more likely to stay in the organization. Turnover is reduced.

When a programmer is designing systems or large subsystems, he is more productive than if he is doing maintenance tasks or writing one shot reports for a user who isn't even sure what he is looking for. Career development is ensured.

There are also several specific areas where programmer productivity is improved. It has been shown that, to improve the productivity of programmers, you should:

1. get better programmers;
2. work on smaller programs; and
3. work on less complex programs.

4GL technology automatically gives you smaller and less complex programs, so you will think you have better programmers!

D. "End user" computing

There has been great controversy over the role of the "end user" in computing. It is now technologically feasible for the end user to design, load and run complete systems. Online databases, microcomputers, spreadsheets, and expert systems all provide the end user with the power, to a greater or lesser degree, to address his own computing needs.

Of course, the role of the professional system builder is far from over, and it will be a long time before all the data processing professional will have to do is the backups. However, the end user can be a very reliable resource for the data processing department and can provide two very necessary functions.

First, he should and must provide the conceptual design for the system. After all, only he really understands what he wants to try to accomplish. The data processing professional must synthesize these desires with the other requirements the computer must fulfill and thus generate a working system for the user. In the multiuser, shared time environment, it is probably not a good idea for individual users to be developing their own systems in their entirety, even if they have the tools to do so.

Second, the user should be made responsible for his own ad hoc, one shot reports where possible, with support from the data

processing professionals. It should not be surprising that people with terminals on their desks do have the wherewithal to learn how to use a report writer, and to become very creative in the development of new kinds of information from the available data. This is not to say that they should have free run of the computer. On the contrary, the data processing professional must provide mechanisms to limit and control access.

By making good use of the end user "resource", the data processing department can offload much of the less satisfying work, the user can get better reports sooner, and system enhancements can be identified and developed in a cooperative atmosphere.

III. Approaches to data organization

In terms of discussing data structure and design, it is useful to look at three levels of abstraction. These have been given different names by different people:

Codasyl DBTG (1971)	ANSI (1975)	C.J. Date etc.
1. physical	internal	file level 2.
conceptual	conceptual	data level 3. view
external	semantic level	

The physical level is concerned with the actual storage of the data. The operations used include insert, delete, modify and find.

The conceptual level is more concerned with the organization of the data than with storage. Another important issue at this level is selecting the data to be stored.

The view level is concerned with what the end users of the data structure will see. In this case, interpretations and relationships between data are important.

A. The file (physical) level models

The hierarchical model consists of owner and member data files, which are connected through explicit relationships (usually called sets). Files may be accessed through index structures, calculated keys, or pointers from another file. This is a very restrictive model for general design, but has proven successful in mainframe implementations such as IMS and System 2000.

The network model is much more flexible than the hierarchical. In the network, owners may own more than one set, and members may belong to more than one set. Hewlett Packard's own Image is an example of a network implementation, along with Total, IDMS and ADABAS in other hardware environments.

The major advantage of the physical level data bases is their ability to be made very efficient at the machine level. They may be tailored and tuned for a specific application. This is important for large applications and very large data bases, which may be implemented on dedicated hardware with very little ad hoc user involvement.

There are several disadvantages to designing at this level. In order to get any results at all, a skilled person is required to navigate through the data base. (This is usually more of a problem on the multilevel mainframe data bases than with Image.) Updates are especially complex, requiring a person who is trained to interpret the data structures. The data base will be inherently rigid, and hard to modify, because all the data

relationships are programmed in. Finally, there is some potential risk because the use of pointers throughout these structures results in physically dependent data. That is, moving files around requires that all the pointers be rebuilt, and data relationships may be lost by a hardware problem.

B. The data (conceptual) level model

The relational model used at the conceptual level is much simpler, which is a boon for both the systems builder and the end user. The basic model is that of a table, or of a very simple fixed length file, with one record type per file, consistently located fields in every record, and no repeating fields.

The major advantage of the relational model is its simplicity. This leads to better systems from many angles. It means the systems are better designed in the first place. It means that they are easier to change when change is required. And it means that computer users who are not data processing professionals can play a much larger part in the extraction of their own information.

The primary disadvantage of the relational model is that more analysis is required in the design. Many of us are not yet ready to think about a system before we roll up our sleeves and start coding, in spite of all our lip service to methodologies and controlled development cycles.

Although there are file level implementations of the relational model, they tend to be slow and unwieldy when dealing with commercial volumes of data.

C. Physical vs. Conceptual Approach

As we move from the internal to the external representation, we find ourselves moving from the machine hardware to the end user's more abstract concept of the data. In fact, this transition may also be seen to symbolize moving from data to information.

The point of the exercise in commercial data processing is to provide information which can help an organization run and help its management make better decisions. To transform data (symbolic patterns stored on a passive medium) into information (characterized as useful, cognitively meaningful) is to bridge the gap between the computer and its end users.

An important factor in the conceptual design approach is that the designer can work a level above the physical file. Neither approach is perfect, with the physical design suffering from rigidity and the pure relational design suffering from inefficiency. However, with a relational design at the conceptual level, we may make some compromises for efficiency at the file level during application system implementation. This provides for

the separation of the logical data design from its physical design, producing a combination of flexibility and efficient machine resource utilization.

IV. An introduction to the relational data model

A great mystique seems to have developed around the word "relational". There have been discussions on a large scale as to what it really means...does it mean you can relate files to each other? or does it have something to do with relative addressing? does it mean that the relationships between the files, fields, etc. are somehow changed? The answer to all of these is: it does not matter!

The concepts are called "relational" because they are derived from a part of mathematics called relational mathematics, a branch of set theory.

Relational mathematics is so named because it deals with relations, made up of repeating tuples of non-repeating attributes, which may be manipulated with several operators. This theoretical construction provides the basis for data design and

data manipulation in the relational model, and we shall describe it briefly here.

A. The table structure

The fundamental structure of the relation is the table. It is a simple two-dimensional structure, which may be visualized as a spreadsheet or a data file. It is special because of its enforced simplicity. Every row in the table must be constructed like every other row, and every row must be uniquely identifiable and not a duplicate of any other row.

To translate into more familiar terms, the relational table is like a simple file, where there may be only one record type, and no duplicate records. There are no pointers, index files, or linkages of any sort between files or between records.

In the terminology of relational mathematics, a "record" or row in the table is known as a "tuple". A column in the table is known as an "attribute", and corresponds to a field or data element.

B. The operators of relational math

Because relational mathematics is concerned with theoretical pursuits, and not with updating your employee master file, it has operators which are very different from the read, add, update, and delete operators which are commonly found in file management systems. The fundamental relational operators are:

1. Select: provides the capability of identifying or extracting the tuples you want, based on their values.
2. Project: provides the ability to create a new relation, based on a subset of the attributes in the current relation, and perhaps on a subset of the tuples.
3. Join: provides the ability to create a new relation, by connecting two or more existing relations via equal values in attributes which are found in both relations to be joined.

It is clear that, in the relational mathematics, there is no provision for data entry, key path optimization, report generation or any other data processing concern. However, it does provide us with a sound theoretical basis upon which to build the more practical requirements.

V. Normalization: a practical approach to file design

It is fairly easy to create a relational table on a blackboard or in a textbook, because the data is necessarily simpler than in the real world. However, when one is trying to automate real business problems, the simplification of the structures is not a trivial matter. It has been found that the techniques of normalization are a relatively simple, rigorous method of ensuring that data structures really are relations.

A. 1st normal form

Structures normalized to the first normal form must satisfy the following condition:

"No repeating attributes, no sub-attributes, all tuples unique."

To illustrate this, I will use an example from my own customer files (suitably disguised, of course).

Customer name: ABC Electronics Location: Dallas, TX
 Products: PowerHouse (2 copies), PowerPlan, MultiView GL
 Sales District: 2

First, all attributes must be made atomic. This means that each data item must contain only one piece of data.

Customer Location

ABC Dallas, TX

becomes:

Customer City State

ABC Dallas TX

Second, all attributes must be made non-repeating. This means no arrays or "occurs" clauses are allowed.

An example of a repeating attribute is Products. It must be made into a single valued attribute.

Customer Product

ABC PH,PP,GL

becomes:

Customer Product

ABC PH ABC PP ABC GL

Finally, all tuples must be unique. No duplicate records are allowed. If I want to track copies of product, not just customers, I need more information:

Customer	Product			
ABC	PH ABC	PH ABC	PP ABC	GL

becomes:

Customer	Product	CPU ID		
ABC	PH	1941A00001	ABC	PH
2360A00002	ABC	PP	2360A00002	ABC
2360A00002				GL

The structure has had to be expanded by the addition of another attribute for uniqueness. It is now in first normal form.

B. 2nd normal form

Structures are in the second normal form when they have an additional criterion satisfied:

"In first normal form and all non-key attributes are fully dependent on the key."

In order to illustrate this, let me introduce another customer:

Customer: XYZ Drilling Co. City: Tulsa State: OK Product: PowerHouse CPU: 1800A00003 Sales District: 9

In order to remove the dependency, it is necessary to split our relation into two new relations:

Customer	City	State	Product	CPU ID	District
ABC	Dallas	TX	PH	1941A00001	2 ABC
Dallas	TX	PH		2360A00002	2 ABC
Dallas	TX	PP		2360A00002	2 ABC
Dallas	TX	GL		2360A00002	2 XYZ
Tulsa	OK	PH	1800A00003	9	

becomes:

Customer	City	State	District		
ABC	Dallas	TX	2 XYZ	Tulsa	OK
9					

and:

Customer	Product	CPU ID				
ABC	PH		1941A00001	ABC	PH	
2360A00002	ABC	PP		2360A00002	ABC	GL
2360A00002	XYZ	PH		1800A00003		

The main reason why it is so important to remove this dependency is that, if allowed to remain, it may lead to the introduction of anomalies into the relation. For instance, in the first normal relation, it is possible to concoct something like "ABC - Tulsa - OK - PH - ...". This is clearly incorrect, but the structure would accept it readily. In the second normal form of the relation, this would be impossible.

A second function which may be performed at this point in the data analysis is to remove any attributes which may be calculated reasonably. For instance, if you are storing a quantity and unit price in a relation, you really do not need the extended cost as well. Removing extra attributes saves storage, but more importantly, reduces the complexity and intradependency of the data.

C. 3rd normal form

A relation reaches third normal form when it satisfies this condition:

"It is in second normal form and contains no transitive dependencies."

To illustrate transitive dependencies, imagine what happens if ABC Electronics is bought out by an old-fashioned organization which doesn't believe in computers, and ceases to be a customer (Heaven forbid). By removing that tuple from my relation, I lose the information that Dallas is in Texas and, more to the point, that Dallas is in District 2. It is once again necessary to decompose the relation into more, smaller, simpler relations:

Customer	City	State	District		
ABC	Dallas	TX	2	XYZ	Tulsa OK
9					

becomes:

Customer	City	State		
ABC	Dallas	TX	XYZ	Tulsa OK

and:

District	City	State		
9	Dallas	TX	9	Arlington TX 2
Tulsa	OK			

This allows us not only to retain the district information if a customer is deleted, but to define the sales districts independently of customers, or without even having a customer there yet. In the real world, this keeps us from dealing with an emergency situation every time we have a customer from a new area.

D. Further degrees of normalization

Fourth and fifth normal forms have been defined, as well as several offshoots. In the practical world, normalization to the third level should be sufficient.

The redundancy of city and state in the previous example was intentional. Unfortunately, the people who developed the names for the places of the world had not studied normalization theory and were not concerned enough with uniqueness. Therefore, when we are interfacing with the real world, we sometimes have to make compromises. A theory is supposed to be a guide to be followed, not a dogma to be worshiped.

VI. Mapping to Image and KSAM

Given that we have a nicely normalized data design, and given that Image and KSAM are not relational data base management systems, how do we get from the conceptual design to the physical design at the file level?

As it happens, both Image and KSAM work very well as keyed file structures, and relational theory likes to see keys for its tuples, even though it does not treat their implementation.

A. Candidate key selection

A candidate key is any attribute or combination of attributes which uniquely identifies the tuple all by itself. In our customer system example earlier, there was a relation which contained the customer's name, city and state. In it, the name is clearly the key attribute. However, if a street address were added, it may also uniquely identify the customer. If so, it would also be a candidate key. From the candidate keys, one is selected arbitrarily to be the primary key for the file. In the example above, it would most likely be the customer's name.

Sometimes, a candidate key is made up of more than one attribute. For example, in our relation containing customer, product and CPU identifier, the only thing which makes a tuple unique is the combination of product and CPU, because a customer may have several products on several CPU's, a CPU may be running several products, and a product may be on many CPU's. In this case, we have what is known as a "concatenated" key.

Select the key or keys which will be required for your application from the candidate keys. It may be necessary to use codes, as well as names, for your suppliers, customers, and so on. If so, you may wish to have on line access through both paths.

Key selection should be based on retrieval speed requirements. Every key provides another path by which the record may be accessed on line, in "real time". It also adds to the system's overhead, especially in adding or updating records. Keys should not be spared where needed, but should not be added on speculation "just in case" the users should request such an access.

For access paths used by reports which are not required instantly, keys are not usually required. The selection and sorting can be done in batch, perhaps overnight, eliminating the processing and storage overhead of index files or pointers.

However, if inquiries or reports are often done on specifiable subsets of data, it may be worthwhile to introduce non-unique (repeating) keys to cut out the overhead of selecting and sorting the whole file for each report. This is frequently

encountered in the multiple attribute "concatenated" key situation, where it is desirable to have fast access paths by all or some of the component attributes making up the primary key. The primary key itself may only be used to ensure uniqueness.

The other use for keys is to provide linkage paths for "pseudorelational" joins, especially for 4GL programming. In fact, the keys selected by the processes to this point usually provide exactly the keys needed for 4GL use.

B. A recipe

We now have simple, clean table structures with keys identified for direct access, uniqueness and connectability. It is now necessary to map them into Image and KSAM.

1. Set up the tables as keyed files (KSAM) or as details with automatic masters (Image). The primary key has been identified in the previous design step! (Note that, if part of a concatenated key is required as a repeating key, it will likely be necessary to repeat the data item, especially with Image. This intentional redundancy is unavoidable, and is not a problem as long as it is expected and controlled.)

2. If you are using KSAM, the job is done!

3. Detail sets with only one key (one automatic master) may be made into manual masters. Before changing, consider whether sequential access will be needed. If so, leave the master-detail structure intact.

4. Duplicate master sets indicate that keys are shared between two or more detail sets. They may be consolidated. You may consolidate two or more automatics, or one manual with some automatics, but if you find yourself consolidating two or more manual masters, you have done something wrong!

5. Check for reasonableness. For instance, if you have pulled "State" out of "Address" just to eliminate a transitive dependency, you are probably getting carried away! Sometimes, it even makes sense to use an array in a record, if it is very static and defineable, although this is a "sin" of normalization. It probably doesn't make good sense to have a separate relation for cities and states, so that you don't risk losing the information that Dallas is in Texas! And so on...you know your application by the time you have been through all this design work, so think about it.

C. Storage concerns

In general, it is a good idea to store as little data as possible to get to the information your users need. Not too long ago, it was common practice to carry redundant data on the system for monitoring data integrity, but there are better data validation tools and techniques available now. If the data is

coming in through an intelligent, on line entry system, it should arrive clean. If the other updating processes have been tested, they should not be corrupting existing data.

Do not be overly concerned with any additional redundancy which may have resulted from the normalization process. The gains in flexibility should more than make up for any extra storage requirements. The key item requirements for Image will likely force a certain amount of redundancy into any design, in any case.

D. Performance concerns

Books have been written about optimizing Image's performance, and this paper will not presume to add to that knowledge.

A couple of important things to remember in designing for Image are that you should:

1. keep chains to a minimum, and
2. use the ones you decide to keep.

Too many chains, especially when sorted, very long or both, will cost a lot of machine resources in entry or update situations. Not using the chains which do exist causes an inordinate amount of extra work, as data sets are read serially to find a few records which could have been retrieved randomly.

The use of KSAM should be considered from a performance perspective. Although KSAM is often seen as just a migration path until a system can be rewritten to use Image, it does some things very quickly and easily by its very nature.

KSAM can help when:

1. You are prototyping. Each file can be modified, unloaded, reloaded, and restructured independently, whereas Image data structures must be modified as a whole. The use of a data dictionary minimizes the effort involved in converting to Image later, if Image is required for the production system.
2. Generic keys are needed. Because of the index tree structure, KSAM can retrieve by partial key very naturally and quickly.
3. Ranges of keys are needed. Again, the index tree structure directly supports retrieval by a key range.
4. The application is fairly static, fairly simple, or involves little large scale batch processing.

VII. Alternatives

Because of the wide scale acceptance of the relational model, there are several approaches to making it useful in the real world.

A. Software relational databases

Relational data base management systems are found on every scale of computer, from mainframes (e.g., Nomad and Focus) through minis and superminis (Oracle, Ingres, SQL) to micros (Lotus 1-2-3, Dataease, rBase and a host of others).

The relational DBMS have positioned themselves to be a decision support tool, rather than a data processing tool. They have made use of inverted files and other indexing structures, and provide the ability to create and dismantle indexes dynamically, as required. They often provide means of adding or modifying attributes without unloading the data already in place.

While software relational DBMS have never solved the problems of performance enough to be considered the answer for production data processing, they are useful in their own way. Data can be extracted from larger, more efficient file structures and loaded into a relational DBMS for management reporting and analysis.

Because of their inherent simplicity and flexibility, they have proven very attractive, but have not provided the performance required for large, commercial volumes of data.

B. Database machines

One of the hardware approaches to implementing relational data base management systems is the back end processor, or so-called "database machine". It operates on the same theory as the front end processor which controls communication, offloading much of the work from the main processor.

In database machines such as Britton-Lee's, the user, the user program and the CPU work with a high level model of the data. The back end processor interprets the high level calls, optimizing storage and retrieval, spreading the data around the available storage devices, and taking care of data access.

Often a disk cache, using dedicated memory at the storage device or at the database machine, is used as a part of this process, making data accessible before it is requested, in much the same way as MPE-V's disk cache in main memory.

C. Associative memory

An alternate hardware approach is to use memory which is addressed in two dimensions like the fundamental table structure

of the relation. Current memory operates with one address per location, and so may be viewed as one long vector. With two dimensional addressing, the relational tables may be mapped right into hardware memory. This eliminates the need for any kind of indexing entirely, as the entire relation may be scanned and searched in one clock cycle!

The problem with associative memory is that it is very expensive and very limited in size. It would take a lot of memory to load in a typical order entry transaction file, especially if it were joined with the appropriate master files to create a useful relation. At this time, associative memory is still primarily in use in military applications, which do not have the same concern with cost, and in certain research applications, which do not have to worry about data volume.

VIII. Conclusion: have your cake and eat it too!

What is needed is something which works now. Every data processing shop is striving to provide its users with more information for less effort, to improve the quality of life for those who work there, and to make a contribution to the success of the organization paying the bills.

The use of relational theory at the data design level, current file and database management technology at the file design level, and fourth generation language to develop the algorithmic portion of the application system is a proven,

effective solution. Thousands of shops have adopted 4GL development, and virtually all are experiencing phenomenal improvements in development time and effort, adaptability to change of their systems, and the dreaded program maintenance. They are able to integrate previously isolated systems to better model their organizations. Scarce data processing experts are more fulfilled and better utilized, and better use is being made of people with limited data processing expertise such as rookie programmers and end users.

The future will demand simpler data structures. Multi-user systems, unforeseen applications and uses of data, and further developments in the use of distributed data will all require that data be simple and flexible. Natural query languages are on the horizon, and they need to work on cleanly structured data. Even the fifth generation work lends itself to clean data. Maybe ten years from now we will be standing here discussing tables not \sim ly of numbers and words, but of pictures, sounds, voices or who knows what!

Fourth generation language development, relational design and efficient file management is the winning combination for today.

Biography

Leigh Solland has a degree in Computing Science and Mathematics from the University of Alberta. He has worked for Cognos Corporation since 1979, as a consultant, product specialist and sales representative (with a year's "sabbatical" at Tym labs in 1984). He currently makes his home in Arlington, Texas, and makes his living as the world's most honest salesman, preaching the virtues of 4GL technology to anyone who will listen.

3014. THE SECRETS OF SYSTEM TABLES... REVEALED!

Eugene Volokh,
VESOFT, INC.
7174 Melrose Ave.
Los Angeles, CA 90046
(213) 937-6620

The System Table is an Ornerly Beast
That defies investigation
Unless one first acquaints oneself
With the proper incantations.

Long must one labor o'er
Deep-hidden books of arcane lore
Until one learns the secrets of
EXCHANGEDB, MFDS, and more.

And formats one must understand
What's in word one, what's in word two
What lurks inside the fourteenth bit
And what the sixteenth SIR can do.

A systems programmer's what Wilfred was,
With stolid heart and trusty blade,
And system tables he did read,
And many useful programs made.

Oh, listen to his fearsome tale
Of magics dark and dragons fierce,
And catch a glimpse of mysteries
Not often told upon this earth.

-- "The Saga of Wilfred,"
Norse, ca. 11th century AD.

ABSTRACT

MPE's System Tables contain a large variety of data that can be very useful for any task from performance optimization to writing system management utilities to improving system security and adding power to application programs. Unfortunately, nowhere is it clearly explained (or for that matter, explained any way at all) how one can access all these system tables. The goal of this paper is to show how one can access data segments, file labels, absolute memory locations, disc-resident system tables -- in general, all forms of system tables -- so that armed with the information gleaned from here and a System Tables Manual, a programmer can sit down and start writing programs that manipulate data segments.

INTRODUCTION

Just like any other program, the operating system must keep track of its current state; information on all the things known to it -- files, jobs, programs, everything -- must be kept somewhere. A "System Table" is just a name for a data structure containing such data. System tables may be stored on disc, in memory as a fixed array of memory locations (e.g. locations %1000 to %1377 of bank 0), in memory as a data segment or a portion thereof, in the stack of some system process or even a user process -- any data structure maintained by the operating system can be legitimately called a system table.

There are two problems with using system tables, which account for the fact that few people know how to use them safely -- one is that the format of the tables is little-known, and is usually documented only in the System Tables Manual (not the most readable document known to man) or, worse, only in the source code of the procedures that actually maintain the data structure. Even when the format is documented (e.g. "word 7 contains the flumbrug framastat" is a typical example), various important details (just what is a flumbrug framastat) are omitted. Equally bad is the other major problem -- even if one knows the format of a system table, nowhere does it clearly describe how to access it.

This paper will try to address both these problems. It will give instructions on how to access various system tables, but will also explain what the various system tables are and how they are inter-related. This, coupled with the precise formats given in the System Tables Manual, should allow the interested reader to learn to manipulate much useful system information. If you don't have the System Tables Manual, you can order it from HP as part number 32002-90003 (MPE IV) or 32002-90010 (MPE V).

Note that although all the examples in this paper are given in SPL, there is no reason why you can't write system table-accessing programs in PASCAL, FORTRAN, or even COBOL. All that is necessary is to write simple SPL routines for performing those operations (such as MFDS or TURNOFFTRAPS) that can only be done in SPL, and then put the routines in an RL or an SL. That way, the routines will be callable from your favorite language. System procedures like FLABIO, ATTACHIO, or GETSIR can already be called from other languages.

"WAITER, THERE'S A SYSTEM TABLE IN MY STACK!"

It might come as some surprise to you that one place the system hides a system table is in your very own stack. Your stack is a "data segment," a contiguous chunk of memory no more than 32K words long, in which addressing is done relative to the "data base" (DB) register (not to be confused with IMAGE databases). If you draw your stack (with addresses growing toward the top), it would look something like this (see Section I of SPL Manual):

High memory		
Z >	-----	
	unused space	
S >	-----	
	operands of machine instructions	
	and local variables of the currently	
	executing procedure	
Q >	-----	
	local variables of other procedures	
	and stack markers indicating calls from	
	one procedure to another	
Qi >	-----	positive
	global variables	addresses
DB >	-----	
	"DB negative area," accessible only by	negative
	SPL procedures (like V/3000) -- usable	addresses
	for global storage	v
DL >	-----	v
	The Nether Regions, where mortals may	v
	not stray and non-privileged accessors	v
	are punished with bounds violations	v
	-----	v
Low memory		

At the very top is the Z register (stands for Zounds?), which is the uppermost boundary of the stack data segment (note that in some manuals, HP draws the stack upside down, with DL on top and Z on bottom -- keep this in mind when comparing this picture with others). Then, below it is the S register, which marks the "top of the stack"; parameters to machine instructions are kept at and around S. Below S is the Q register, which indicates the top of the stack at the time the currently-executing procedure was called; the procedure's local variables are allocated above it, with the procedure's parameters and the stack marker (indicating the place from which the procedure was called) immediately below it.

"Qi" is actually not a register, but rather the initial value of the Q register (which changes with every procedure call and every procedure exit). DB is the base register, from which all addressing is done. Word address 6680 is actually "DB+6680"; any pointers that are kept (even if they are pointers to local variables, which are between the Q and S registers) are relative to DB, since DB is the only register that is guaranteed not to change. DL (which - since it, like all registers save for DB itself, is expressed as a DB-relative address - is a negative address) marks the lowest point in the stack that is accessible to user code; it, too, can be moved to dynamically expand or contract the useful DL-DB area, in which SPL procedures (like V/3000) can store global data.

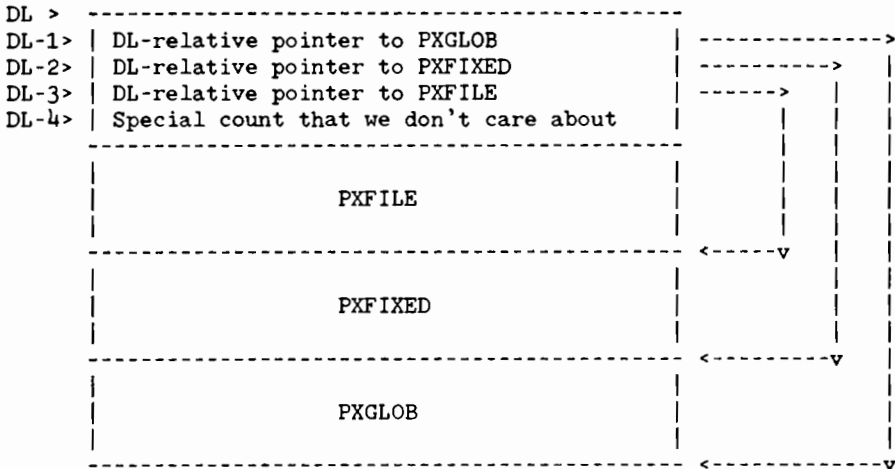
What interests us is the "nether reaches," the area below DL. Your mother may not have told you this, but there is a whole wealth of information (for many programs I've seen, more interesting than the stuff above DL) called the "PCBX" tucked away there. "PCBX" stands for "PCB eXtension" (the PCB being an important system table that you'll

hear more about later), and contains much of the information that the system needs to know about the process -- what files it has open, what session it belongs to, where other tables that describe the session more thoroughly (the JIT and JDT) are, what data segments this process has allocated that must be deleted when it dies, and so on.

The PCBX is divided into four portions (see Ch.7 of Sys.Tables manual):

- PXGLOB, which lists things like the capabilities of the person who is running the process, the DST indexes (aka data segment numbers, which we'll talk more about later) of the Job Information Table (JIT) and Job Directory Table (JDT) pertaining to the process's session, and so on.
- PXFIXED, which describes some other things about the process -- what data segments it has allocated, what session it belongs to, what the values of its registers are (if it is not currently active; if it is currently activate, i.e. being executed by the CPU, the register values are actually stored in the CPU registers), and so on.
- PXFILE, which describes the files that the process has opened.
- The pointer area, which points to the other portions. Unlike most other pointers, these pointers are DL relative and are implied to be negative; i.e. a value of 364 points to DL-364.

They are arranged roughly as follows:



Now, how does one access these system tables? Well, they are kept in your stack, so they can be accessed using simple SPL DB-relative pointers. For instance, to get word 2 of the PXGLOB (which happens to be, in both MPE IV and MPE V, the zeroth word of the user's capability mask, containing information on whether the user has SM capability,

AM, etc. -- all the capabilities save for PM, PH, DS, MR, IA, and BA) you would merely have to do the following:

```

INTEGER POINTER DL;
INTEGER CAPS;
PUSH (DL); << push the address of DL onto the stack >>
@DL:=TOS; << DL now points to the DL register >>
GETPRIVMODE; << all DL-negative addressing must be done in priv >>
CAPS:=DL(
    -DL(-1)+ << -DL(-1) is the DL-relative ptr to PXGLOB >>
    2); << add 2 to get to the caps word >>
GETUSERMODE;

```

Voila! Note that we have to be in privileged mode to access ANY word of the PCBX. After all, if you could do this in user mode, you could change the line "CAPS:=DL(-DL(-1)+2)" to "DL(-DL(-1)+2):=CAPS" and CHANGE the capabilities word instead of just reading it... in fact, a variation of this is exactly what VESOFT's GOD program (which temporarily -- for the duration of the session -- gives the user who ran it all the capabilities and all the ALLOWS) does.

Similarly, one can get at all the other portions of the PCBX. For instance, to get the Active File Table (AFT) entry for file number FNUM, one can do the following:

```

image 7 INTEGER POINTER DL; INTEGER ARRAY COPY'OF'AFT'ENTRY(0:5); PUSH
(DL); @DL:=TOS; GETPRIVMODE; MOVE
COPY'OF'AFT'ENTRY:=DL(-6*FNUM-4),(6); GETUSERMODE;

```

Of course, you must first know that the AFT entry for file FNUM is 6 words long and always starts at location DL-6*FNUM-4 (in MFE V only -- this is different from MPE IV). Also note that the AFT is actually known as either the Active File Table or the Available File Table (depending on which chapter of the System Tables Manual you believe). You can look up its format in chapters 6 and 7 of the manual.

If you want a brief exercise in using this access method, whip out your trusty System Tables Manual and write a small program to print out your current capability word (in octal) and your current job/session number (HINT: it's in the PXFIXED). While you're at it, also change your current capability word to give yourself SM capability (make sure you do not already have it) and do a "LISTUSER MANAGER.SYS" using the COMMAND intrinsic before and after this change. You don't need to change it back since it is valid for the duration of the process only, and when the process dies, the user is left with the same capabilities as before. The solution to this problem (which we'll call Problem 1) is in Appendix 1.

Of course, when you're learning to use these access methods, be sure to run all your test programs during off hours or on a non-production computer. I've crashed my share of computers when I was starting to work with system tables, and you'll probably crash some, too. Fortunately, once you get acquainted enough with the techniques involved, and especially if you use some of the safety devices I'll

describe later, your programs will probably be far more reliable. In the meantime, expect some foul-ups.

Another useful trick is using privileged mode DEBUG to ensure that your program is working right. Say your program doesn't produce the correct result -- it could either be because you're trying to get the wrong value (e.g. PXGLOB location 3 doesn't really contain what you think it contains) or that you're going for the right value, but your method of getting it is wrong or you're outputting it wrong. To find out what the problem is, you can use DEBUG to look at the value manually -- if it's the expected value, then the bug is in your retrieval or output algorithms; if it's not the expected value, you're trying to retrieve from the wrong place.

The way to get at values in PXGLOB using privileged mode DEBUG (you must be a privileged user to do this!) is to type a command of the type "DDL-address,length". Don't forget that DEBUG by default assumes octal input and output; to input a decimal number, prefix it with a "#"; to make output come out in decimal, type "DDL-address,length,I". Thus, to see the value of DL-1 (the PXGLOB pointer), type "DDL-1,1,I", or just "DDL-1,I" since 1 is the default length. To find the value of PXGLOB word 3, you can either first do a "DDL-1,I", and do "DDL-#xxx+3,I" (where xxx is the value that the DDL-1 gave you), or you can say "DDL-'DL-1'+3,I" -- the "'DL-1'" means "the value stored at DL-1".

One important note: although the access method I just described works well enough, I favor a different approach that I'll discuss when I get to system tables kept in arbitrary data segments and the MFDS instruction.

THERE'S DATA IN THEM THAR DATA SEGMENTS

If you peek into your friendly neighborhood System Tables manual (which at last count had 23 chapters), you will find that about half of chapter 7 is dedicated to tables that reside in stacks' DL-negative areas, and you might start to wonder what the remaining 22 1/2 chapters are about. Well, most of the system tables described in the manual are stored in operating system Data Segments, and if you thought that DL-negative tables were tough to access, just wait until you see these.

It doesn't make sense to keep data in fixed locations in memory, e.g. putting the table of currently active jobs in locations %10400 through %10777, a process's stack in locations %533000 through %534377, and so on. For one, the very concept of "virtual memory" demands that the operating system be able to swap data out from disk and then bring it back into a different location in memory; furthermore, the HP 3000 is still a 16-bit machine (yes, Virginia, there are still 16-bit machines out there, and you're using one of them), and it would have trouble supporting addresses that are more than 16 bits long.

Rather, the data is broken up into many "segments". Within each segment, all addresses are relative to the base of the segment; if one segment has to point to another, it would have both the segment number and the offset within the segment. This way, a segment can be conveniently swapped in and out of memory, since the physical address of the segment is kept in only one table that is managed by the swapper; also, since most addresses are segment-internal, we can usually get away with quick 16-bit addressing.

Consider for instance your own process stack -- it, too, is a data segment. Addresses within it are relative to the process's DB register (a slight difference from other segments, in which the addresses are relative to the segment's base); the stack contains all your data plus the tables in the DL-negative area that we talked about. In case you should wish to access another process's stack, you can -- all you need is its "data segment number." Similarly, the table of all the currently active jobs (called the "JMAT," for Job MASTER Table) is also kept in its own data segment, as is the table describing all currently active processes (the PCB -- Process Control Block). Any such table can be accessed if you only know its data segment number.

Incidentally, let me mention a certain documentation inconsistencies pertaining to data segments. What I call the "data segment number" is sometimes called:

A "DST index." All existing data segments are described in a system table (which itself is in a data segment) called the Data Segment Table, aka DST. The number of the data segment is thus the index into that Data Segment Table.

A "DST number," a corruption of "DST index." Doesn't make much sense -- Data Segment Table number?

A "DST," a further corruption of "DST index." Although this is actually both ambiguous and technically improper, lots of people use it (including yours truly). It's all HP's fault that they didn't call it something decent like "data segment number" in the first place.

I shall make a daring attempt to CONSISTENTLY call these critters data segment numbers throughout this document, but don't count on it.

ACCESSING DATA SEGMENTS -- ONE APPROACH

We have established that most of the data worth knowing is kept in data segments, and that a piece of data can be accessed by specifying the data segment number and the offset within the data segment, without caring about the physical memory address, which anyway will probably change when the segment is swapped in and out. Now, the question becomes: how do you get to it?

Well, thought some unknown HP designer when he was faced with making all this up back in '72 when the 3000 was being built, we can already access arbitrary DB-relative addresses in our stacks -- why don't we

just provide some way of switching the DB register to point to an arbitrary data segment? So, say that someone wants to get location 3 of data segment 55 -- he'll just have to switch to segment 55, grab location 3 (note that he'll do it just like he'd grab location 3 of his own stack, except that now DB points to segment 55), and switch back to his stack, the value of the desired location in hand.

Thus, our program would look something like:

```
<< ATTENTION: People who're just looking at the pictures:
THIS PROGRAM WON'T WORK! DON'T EVEN TRY IT! Read the text...
>>
$OPTION PRIVILEGED << PM or no dice! >>
BEGIN
INTEGER
  DUMMY, << location DB+0 >>
  I; << location DB+1 >>

<< This is a system procedure that switches DB to point to the
specified data segment, and returns the number of the
data segment to which it now points.
If the data segment number is 0, switches to the process's
stack. This is NOT the same as the documented privileged
procedure SWITCHDB -- THEY CAN NOT BE USED INTERCHANGEABLY!
SWITCHDB is only intended for accessing your own data segments
that were allocated by your program using GETDSEG. >>
INTEGER PROCEDURE EXCHANGEDB (DSEG'NUMBER);
VALUE DSEG'NUMBER;
INTEGER DSEG'NUMBER;
OPTION EXTERNAL;

DUMMY:=EXCHANGEDB (55);
ASSEMBLE (LOAD DB+3); << push the value at DB+3 onto the stack >>
I:=TOS; << pop it from the stack -- I now contains the value
of the 3rd location in segment 55 >>
DUMMY:=EXCHANGEDB (0); << switch back to the stack >>
<< now, do whatever we want to do to I >>
END.
```

What did we do? We set our DB register to point to data segment 55, we got the value at location DB+3 -- now location 3 of segment 55 -- and moved it into I, and then we switched back to our stack. Note that we did not use the value returned by the first EXCHANGEDB, which is the data segment number of the segment to which DB pointed to before -- namely, our stack -- as the data segment number in the second EXCHANGEDB. That's because if we just EXCHANGEDB to our stack's data segment number, DB will point to the BASE of the stack data segment; however, specifying 0 as the data segment number is a special feature that switches back to the process's stack data segment and makes DB point to the right place.

But THIS DOESN'T WORK. What's more, IT FAILS IN A TRULY SPECTACULAR WAY, most likely crashing your system or worse. Why? Because when we do an EXCHANGEDB, ALL THE DB-RELATIVE ADDRESSES NOW POINT INTO THE

DATA SEGMENT, including all of our DB-relative variables -- such as I -- and our arrays, either DB-relative (global), or Q-relative (procedure local)! When we move the top of stack into I, we are moving it to DB+1 -- the location in which I is stored -- and DB+1 now points into the data segment, too! In fact, the only way in which the above could be made to work is by making sure that whenever we use any DB-relative addressing while in "split-stack mode" -- i.e. when we've EXCHANGEDBd to another segment -- we mean addressing relative to the data segment. For instance, we could leave the value of location 3 on the stack, EXCHANGEDB back to our stack data segment, and then safely pick the value up into I; or, we could put this in a procedure, and make I a Q-relative variable. Remember, Q-relative and S-relative addressing does not get changed; however, ALL DB-relative addressing does.

The bottom line is that, for all practical purposes, we can have access to only one segment at a time -- either our stack, in normal mode, or one alternate data segment in split-stack mode. We can't really work with both, except when we're in split-stack mode and we use S-relative and Q-relative addressing. Remember, though that even this is restricted since most arrays, even Q-relative ones, use DB addressing anyway (because they have a pointer that points to the data, and all these pointers are always DB-relative). So, if for the entirety of our process's life we want to be dabbling with data segment 55, we have no problem; however, if we usually want to work with our stack -- remember, data segment 55 probably does not have room for our temporary variables, arrays, and whatever else is usually kept in our stack -- and only sometimes get at data segment 55, we have a problem.

Thus, the rules of the game for split-stack mode operation are:

- * Always be keenly conscious of when you are in split-stack mode. Try to be in split-stack mode as little as possible (I prefer never to be in split-stack mode except when some operating system procedure that I'm calling -- such as DIRESCAN, the directory traversal procedure -- requires it).
- * When in split-stack mode, remember that only:
 - By-value procedure parameters;
 - Simple (non-array) procedure-local variables;
 - Variables explicitly declared to be Q-relative or S-relative (using constructs like "INTEGER XYZZY = S-3;");
 - Local arrays that are declared to be Q-relative (e.g. "INTEGER ARRAY X(0:4)=Q;");

refer to data in your stack; all others refer to data in the data segment to which you've switched.

A CIVILIZED ALTERNATIVE

Fortunately, there is another way of accessing data segments -- two little-known privileged machine instructions called

MFDS (Move From Data Segment)

MTDS (Move To Data Segment)

Their principle of operation is simple -- they move data from the stack to a data segment or from a data segment to the stack. Thus, if you want to read location 3 of data segment 55, you'd issue an MFDS instruction indicating that you want to move 1 word from location 3 of data segment 55 to some buffer in your stack. It's simple, and since you don't do an EXCHANGEDB, you can keep using all of your variables with no problems.

Now, let's be a bit more specific: just how does one tell the instruction where to move from and where to move to? Well, the best place where instructions can take input parameters is from the stack -- not just from your stack data segment, but from the topmost few locations on the stack, just below the S register. You can put parameters onto the stack using the SPL construct

```
TOS:=parameter value;
```

and then issue the instruction by entering

```
ASSEMBLE (MFDS 4);
```

or

```
ASSEMBLE (MTDS 4);
```

The "4" in the ASSEMBLE statement is the so-called "stack decrement" -- it indicates how many of the parameters passed to the instruction should be removed from the stack once the instruction is done. Each instruction takes 4 parameters (which we'll talk about soon), and there's no reason to leave any of them laying around on the stack, so we tell the instruction to get rid of all 4 of them.

Now, each instruction must take as parameters:

The address of the buffer you want to move from or move to. Usually, this is "@BUFFER", where BUFFER is the name of the buffer array. This must be a word address; this usually means that if you specify "@BUFFER", BUFFER must not be a byte array, but should rather be an integer, logical, or double array.

The data segment number you want to move to or move from.

The offset in the data segment at which you want to start the move.

The length (in words) of the data to be moved.

Note that if you specify an incorrect data segment number, invalid offset, invalid buffer address, or invalid length, HP will reward you with a system failure...

The order of the parameters is NOT the same for both instructions. Rather, for each instruction you should put onto the stack the parameters that describe "where to move to", then the parameters that describe "where to move from", and then the length. In other words, the MFDS calling sequence would be

```
TOS:@BUFFER;    << move to >>
TOS:=DSEG'NUMBER; << move from >>
TOS:=OFFSET'IN'DSEG; << move from >>
TOS:=LENGTH;
ASSEMBLE (MFDS 4);
```

and the MTDS calling sequence would be

```
TOS:=DSEG'NUMBER; << move to >>
TOS:=OFFSET'IN'DSEG; << move to >>
TOS:@BUFFER; << move from >>
TOS:=LENGTH;
ASSEMBLE (MTDS 4);
```

Also, don't forget that the instructions must be executed when in privileged mode.

So, let's try a real-live application. Say that you want to write a program that prints out the session limit -- the maximum number of sessions that can be running at one time. Let's see how you could go about doing it.

There's an old joke about a Chinese recipe for broiled rabbit -- the recipe starts with "first, catch the rabbit". We must first find out where the session limit is stored. Well, the session limit is logically job information ("job" in this context pertains to both batch jobs and online sessions), so it would most reasonably be in the "JOB INFORMATION" chapter of the System Tables manual -- Chapter 8. In fact, on page 8-??? is the layout of a table called the JMAT (Job Master Table), and lo and behold, there in word 8 (MPE IV) or in word 10 (MPE V), is the session limit. So far, so good. And, what's more, on that very page (or barring that, in chapter 2) it says that the JMAT is data segment number (they probably say "DST Entry assignments") 25.

So, the session limit is word 8 (MPE IV) or word 10 (MPE V) of data segment number 25. We know that it's one word long, so we can write the following program:

```
$CONTROL NOSOURCE, USLIMIT
<< :PREP me with ;CAP=PM >>
BEGIN
INTRINSIC
  GETPRIVMODE,
```

```

GETUSERMODE,
QUIT;
INTEGER
SESSION'LIMIT;

GETPRIVMODE;
TOS:@SESSION'LIMIT; << Move to the SESSION'LIMIT variable >>
TOS:=25; << Move from data segment 25 >>
TOS:=8; << MPE IV; in MPE V, use 10 >>
TOS:=1; << Move 1 word >>
ASSEMBLE (MFDS 4); << Move From Data Segment >>
GETUSERMODE;
QUIT (SESSION'LIMIT);
END.

```

When you run this program, it'll get the session limit, and call QUIT with the specified parameter. I decided to call QUIT because QUIT will print the parameter and it's easier than calling PRINT and ASCII (I'm actually used to using my own SPL output package, which I described in the "Winning at MPE" column of the DEC 1983 through MAR 1984 Interact magazines; it makes numeric formatting much easier).

Now, you know for sure whether or not this program works because you can do a :SHOWJOB and see the real session limit. However, if you tried to move something from a data segment and wasn't sure whether your program was working right or not, you could use privileged mode DEBUG to check it out. The command you'd use is "DDA" (Display DATA segment), and you'd enter "DDA dsegment+offset,length". Thus, to check out the above program, you can type:

```

:DEBUG

*DEBUG* PRIV.xx.xx
?DDA#25+#8 <<or #10 for MPE V>>,1,I
DA31+10 +xxxxx
?E

```

The "xxx"s stand for the values that depend on your system configuration -- the "+xxxxx" that comes out on the same line as "DA31+10" (or "DA31+12" in MPE V systems) is your session limit. Note that DEBUG echoes "DA31+10" -- 31 is the octal representation of decimal 25, and 10 is the octal for decimal 8.

So that's really all there is to it -- figure out the data segment number to which to move from/to (which is either a constant given in the System Tables manual or a variable kept in some other system table) and the address within the data segment to move from/to, and then issue the appropriate MFDS or MTDS instruction. It's really rather simple, and it's a shame that HP doesn't explain it anywhere except a remote corner of the Machine Instruction Set Manual (and even there, not too well). It would have made sense for HP to describe this -- in fact, describe everything mentioned in this paper -- in the

System Tables Manual or even a more widely distributed document. Oh well, I guess that's just not the "HP way".

However, there is one gigantic problem with the above approach -- even if you're just doing MFDSs, passing an incorrect parameter will crash the system. Even if you write a procedure that does the "TOS="s and the ASSEMBLES, thus avoiding typographical errors, you're still going to have many bugs in your program, and having the system crash for each one of them is not a very appealing prospect (in fact, a very appalling prospect).

This is especially a problem if you're not accessing permanent system tables (like the JMAT) but rather more ephemeral tables like other process's stacks. You could easily get the data segment number of another process's stack, and by the time you try to read it, the process (and the data segment) might be gone.

The solution is to write procedures that get passed the MFDS/MTDS parameters, check them to ensure validity, and only do the MFDS or MTDS if the parameters are OK. That's what I'm going to talk about now, because writing code that uses MFDS and MTDS without having these safeguards (at least while debugging) is, in my opinion, grossly impractical.

What kinds of errors can you have in your choice of parameters? Well one is that you're moving data from or into the wrong place. There's nothing any automatic checking can do about this -- that's a logical error that you'll have to find yourself. I only hope that these logical errors will happen to you mostly on MFDSs and not on MTDSs.

There are several errors, though, that are automatically detectable:

Negative length, data segment number, or offset. I'm not certain about this (and I'm not going to risk a system failure to try to find out), but maybe a negative length would mean "right-to-left" movement (like it does in the MOVE statement -- see the section on the MOVE statement in the SPL Reference Manual). However, since even if this were so, it wouldn't be a very useful option, a negative length, data segment number, or offset are pretty certainly errors. Note that negative buffer address is NOT an error -- remember, buffers in the DL-to-DB area will have negative addresses. Also, data segment number 0 is an error.

Invalid data segment number. Not all data segment numbers are valid -- they can be up to 1023 in MPE IV and even more in MPE V, and there aren't always going to be that many real, allocated data segments. How could we check for this? Well, we'd have to look this up in another system table, the Data Segment Table (DST); it's described in System Tables Manual Chapter 2, and we'll talk more about it later.

Buffer address out of bounds. This means that the starting address of the buffer is less than the DL register (i.e. the buffer starts in the PCBX or even lower, outside of your stack in somebody else's



chunk of memory) or the ending address of the buffer (@BUFFER+LENGTH-1) is greater than the S register. Remember, this is privileged mode -- no bounds violation checking!

Data segment offset out of bounds. This means that the offset is less than zero (already mentioned above) or offset+length-1 is greater than the current length of the data segment. The length of the data segment is also stored in the DST.

So, our code would look something like:

```

$CONTROL NOSOURCE, SEGMENT=DSEG'IO, SUBPROGRAM, USLIMIT
<< Caller must be :PREPped with ;CAP=PM >>
BEGIN
DEFINE
  TURNOFFTRAPS =
    BEGIN
      PUSH (STATUS);
      TOS.(2:1):=0;
      SET (STATUS);
      END #;

LOGICAL PROCEDURE DSEG'CHECKPARMS
  (DSEG'NUMBER, OFFSET, BUFFER, LENGTH);
VALUE
  DSEG'NUMBER,
  OFFSET,
  LENGTH;
INTEGER
  DSEG'NUMBER,
  OFFSET,
  LENGTH;
ARRAY
  BUFFER;
BEGIN
<< Returns TRUE if all parameters OK. >>
INTRINSIC
  GETPRIVMODE;
INTEGER POINTER
  DST = 2; << I'll talk more about this later >>
INTEGER POINTER
  DL;

<< First, find out the address of DL -- the lowest valid stack
  address -- for use later on. >>
PUSH (DL); << Push the value of the DL register onto the stack >>
@DL:=TOS; << Pop it into our own pointer called "DL" >>

<< Now, get privileged mode for subsequent uses of "DST", the
  system table pointer we defined above (more about this later).
  Privileged mode should NOT be relinquished by this procedure
  (again, more about this later). >>
TURNOFFTRAPS;
GETPRIVMODE;

```



```

<< Now, check all the possible conditions >>
IF
  << Is any of the parameters negative (or DST=0)? >>
  DSEG'NUMBER<=0 OR OFFSET<0 OR LENGTH<0
  OR
  << Is DSEG'NUMBER greater than the maximum data segment #? >>
  DSEG'NUMBER>=DST(0)
  OR
  << Is DSEG'NUMBER invalid (indicated by having the length
  in its DST entry be 0)? >>
  4*DST(4*DSEG'NUMBER).(3:13)=0
  OR
  << Is the last data segment address to be moved
  (OFFSET+LENGTH-1) greater than the last valid data segment
  address (which is (data segment length - 1))? >>
  OFFSET+LENGTH-1>DST(4*DSEG'NUMBER).(3:13)*4-1
  OR
  << Is the starting address of the buffer below DL? >>
  @BUFFER<@DL
  OR
  << Is the ending address of the buffer (@BUFFER(LENGTH-1)) above
  the greatest possible address, which is just one word below
  the first parameter to this procedure? >>
  @BUFFER(LENGTH-1)>DSEG'NUMBER-1
  THEN
  DSEG'CHECKPARMS:=FALSE
ELSE
  DSEG'CHECKPARMS:=TRUE;
END; << the procedure exit gets you back to user mode >>

PROCEDURE DSEGREAD
  (DSEG'NUMBER, OFFSET, BUFFER, LENGTH);
VALUE
  DSEG'NUMBER,
  OFFSET,
  LENGTH;
INTEGER
  DSEG'NUMBER,
  OFFSET,
  LENGTH;
ARRAY
  BUFFER;
BEGIN
  << Reads LENGTH words from offset OFFSET of data segment #
  DSEG'NUMBER into the array BUFFER.
  Calls QUIT if any parameter is invalid. >>
INTRINSIC
  GETPRIVMODE,
  QUIT;
IF NOT DSEG'CHECKPARMS (DSEG'NUMBER, OFFSET, BUFFER, LENGTH) THEN
  QUIT (1717)
ELSE
  BEGIN
  TOS:=@BUFFER;

```

```

TOS:=DSEG'NUMBER;
TOS:=OFFSET;
TOS:=LENGTH;
TURNOFFTRAPS;
GETPRIVMODE;
ASSEMBLE (MFDS 4);
END;
END; << the procedure exit gets you back to user mode >>

```

```

PROCEDURE DSEGWRITE
(DSEG'NUMBER, OFFSET, BUFFER, LENGTH);
VALUE
DSEG'NUMBER,
OFFSET,
LENGTH;
INTEGER
DSEG'NUMBER,
OFFSET,
LENGTH;
ARRAY
BUFFER;
BEGIN
<< Writes LENGTH words from the array BUFFER to offset OFFSET of
data segment # DSEG'NUMBER.
Calls QUIT if any parameter is invalid. >>
INTRINSIC
GETPRIVMODE,
QUIT;
IF NOT DSEG'CHECKPARMS (DSEG'NUMBER, OFFSET, BUFFER, LENGTH) THEN
QUIT (1718)
ELSE
BEGIN
TOS:=DSEG'NUMBER;
TOS:=OFFSET;
TOS:@BUFFER;
TOS:=LENGTH;
TURNOFFTRAPS;
GETPRIVMODE;
ASSEMBLE (MTDS 4);
END;
END; << the procedure exit gets you back to user mode >>
END.

```

I'll be the first to admit -- that's a lot of code. But with it, I am able to confidently test all my programs that read data segments (those that write other data segments can, of course, cause other nasty problems) on a production computer during working hours with no fear at all. In fact, I don't recall a single system failure in my past year of development, in which I've been writing and maintaining many programs like VESOFT's MPEX, LOGOFF, and others, all of which do heavy system tables work. When I did the conversion of these programs from MPE IV to MPE V, the first thing I did was to ensure that DSEGREAD works. Once that was done, despite the fact that there were

bugs in the programs that it took several hours to iron out, there were no system failures.

Now, a couple of comments about the above code are in order:

- * First, note that the procedures abort whenever an incorrect parameter is passed -- why? Well, an incorrect parameter is almost guaranteed to be caused by a program bug (by definition). What's the use of going on, doing other things, when you know that due to a problem bug, your program couldn't read or write data that it may be relying on? You could, of course, have the procedure return a logical flag, but then you'd have to check the flag every time you call it and probably wind up calling QUIT anyway. A good idea, however, is to somehow get the procedure to indicate the place they aborted and the cause of their abort -- negative parameter, invalid data segment number, bad offset, etc.
- * Second, note that all three procedures get privileged mode AND NEVER GET USER MODE! This is because whenever a procedure is exited, the mode is always reset to the mode of the caller; so, we don't need to get user mode explicitly. What's more, if we try to, it could actually hurt because it will cause an abort if the calling procedure itself is privileged. MPE does not permit code executing in user mode to exit to code executing in privileged mode. So, getting user mode explicitly is both unnecessary and undesirable.
- * Third, what's this "TURNOFFTRAPS" nonsense? Well, once upon a time (a long, long time ago, in a galaxy far, far away but really in Cupertino), I was having problems with my privileged code aborting. Well, a friendly HP person suggested that I turn off arithmetic traps (things like INTEGER OVERFLOW and others) before going into privileged mode, and lo and behold! all was well. Apparently, some privileged operations require arithmetic traps to be off (although why I'll never know), so to be on the safe side, I always turn them off before going into privileged mode. Like privileged mode, an exit from a procedure resets the setting of the arithmetic traps to what it was in the calling procedure. Note that some people (including Stan Sieler, an ex-HPer now at Allegro Consultants, who's likely to know about these things) believe that you only need to TURNOFFTRAPS before calling a system internal procedure (ATTACHIO, EXCHANGEDB, GETSIR, etc.) and not when you're just executing a privileged instruction (MFDS, LST, etc.). They may be right -- try it both ways and see.
- * Finally, let me just caution you that the above checks are NOT foolproof. Not only will they allow you to write incorrect data into data segments -- they'll make certain that the data segment number's correct but there's no way they could check the data -- but also, if you read or write a segment that is still around but is being deleted (or contracted), the procedure might check the segment while it's still around, see that all's well, and then do the MFDS/MTDS after the segment is gone. In this case, the system will indeed crash. I have NEVER gotten this to happen, but it's

possible -- the only way to completely avoid it is to make sure that there are no bugs in your programs. Good luck...

USING MFDS/MTDS TO ACCESS THE PCBX AREA

When I was talking about accessing the DL-negative area, I mentioned that there was a way of doing it that I liked better than the one I presented. Well, here it is.

Remember, your stack is a data segment like any other segment. You can access it using MFDS just as easily (or as difficultly) as, say, the PCB, or somebody else's stack. All you have to do is find its data segment number, which is not very hard, since it is in your process's PCB (that's Process Control Block -- a very useful system table) entry. Then, to read the PXGLOB, just do a DSEGREAD from your stack segment, location 0, length 8 (MPE IV) or 12 (MPE V). To get the PXFIXED, just do a DSEGREAD from your stack segment, with a starting location of 8 (MPE IV) or 12 (MPE V) -- just above the PXGLOB. To get the PXFILE, you have to get the PXGLOB, get the address of DL relative to the start of the data segment (that's PXGLOB cell 0), and then get the PXFILE address from DL-3.

So, just write the following procedure:

```

INTEGER PROCEDURE MYSTACK;
BEGIN
  << Returns the data segment number of the process's stack >>
  INTRINSIC
    GETPRIVMODE;
  INTEGER POINTER  << see below for more info on this construct >>
    PCB = 3;

  << SL procedure that returns the process's PIN >>
  INTEGER PROCEDURE MYPIN;  OPTION EXTERNAL;

  TURNOFFTRAPS;
  GETPRIVMODE;
  MYSTACK:=
    << MPE IV: >>  PCB (16*MYPIN+3).(1:10);
    << MPE V: >>  PCB (21*MYPIN+3).(2:14);
  END;

```

Note the use of the MYPIN procedure -- that's how you get your Process Identification Number (PIN), which is also the number of your PCB entry.

Now, you can just call DSEGREAD, pass to it MYSTACK, and presto! you're reading your own stack.

"Now, Eugene," you must be saying, "why would you want do a damn fool thing like that? There you have the DL negative area, easy as pie to access using pointers, and you insist on using an entirely different strategy. Why bother?" Well, the answer is really quite simple -- I'm

lazy. I don't want to have to write and remember two sets of procedures -- DSEGREAD/DSEGWRITE and PCBXREAD/PCBXWRITE. I'd rather have one set that I would use to read either a data segment or the PCBX, and one way of looking at things -- everything's just another data segment, including the PCBX. System tables are complicated things, and any bit of conceptual simplification helps.

AN EXAMPLE -- GETTING YOUR OWN SESSION NAME

The perfect example of using MFDSs (or actually, DSEGREAD -- you should always call DSEGREAD, and never have an MFDS or MTDS anywhere else in your code) is getting your own job/session name. This is a useful piece of information that no HP-supplied intrinsic gives you -- you can use it to identify users (e.g. there's one user called CLERK.PAYROLL, with session names JOE, SUSAN, etc. -- a technique we support and encourage with our VESoft SECURITY/3000 product), provide accounting information, and whatever else.

First, we have to find out where it is. It comes under the broad category of "JOB INFORMATION", and if you look into the System Tables Manual, Chapter 8, you'll find it as a field of the Job Information Table (also known as the JIT). But, the JIT isn't like other tables, like the PCB, DST, or JMAT, which have constant data segment numbers -- each session has its own JIT. Well, it turns out (and this is NOT easy to find out) that the data segment number of the JIT is stored in the PXGLOB. So, what you'd do is:

- * Get the PXGLOB (using either of the methods shown above of accessing your PCBX).
- * With the JIT data segment number from the PXGLOB, get the JIT.
- * Extract the session name from the JIT.

To see the actual program, look at program 2 in Appendix 1.

ACCESSING MEMORY-RESIDENT SYSTEM TABLES

As if EXCHANGEDB and MFDS/MTDS weren't confusing enough, there's yet another way of accessing some system tables.

Certain system tables (e.g. the PCB, DST, CST [Code Segment Table], etc.) are memory-resident -- they are always in main memory and are never swapped out to disc. These tables can be accessed using two special machine instructions called LST (Load from System Table) and SST (Store into System Table). This would be a moot point if not for the fact that SPL has an feature that uses these instructions to allow you to easily access memory-resident system tables. (Note that this feature has only been documented in the FEB 84 version of the SPL reference manual, p. 2-13, 7-18, and 7-19; all prior versions of the reference manual do not mention it.)

In SPL, if you say "INTEGER POINTER name = number;", all subsequent references to "name(index)" will access the index'th word of the memory-resident table indicated by "number". Thus, if you declare "INTEGER POINTER TAB'PCB = 3;", saying "I:=2+TAB'PCB(1);" will set I to 2 plus the value of word 1 of the PCB. Or, if you say "TAB'PCB(SIZE'PCB*PIN+7):=24;", it will move 24 to the SIZE'PCB*PIN+7th word of the PCB.

This means that to access these tables, you need not do an MFDS or MTDS -- which are somewhat slower and also more cumbersome to call -- but rather just treat the table as if it were an array that you could index just like an ordinary array.

Several things must be noted about these constructs:

- * MEMORY-RESIDENT TABLE IDENTIFIERS (the numbers that you put after the "=" in an INTEGER POINTER declaration) ARE NOT THE SAME AS DATA SEGMENT NUMBERS! In the case of the PCB, the data segment number and the memory-resident table identifier happen to be the same (3). For other tables, this may not be the case. The way to determine a table's memory-resident table identifier is to look into Chapter 1 of the System Tables Manual where the System Global (SYSGLOB) area is described. If the Nth word is indicated as containing the address of (or pointer to) a system table, then N is that table's identifying number.
- * Memory-resident tables can only be accessed in privileged mode. That means that whenever you want to use a memory table pointer (declared using the "INTEGER POINTER name = number" construct), you must be in privileged mode.
- * Memory-resident table pointers can only be used when indexed, and then only in expressions or in assignment statements. They can not be used in MOVE or SCAN statements, as by-reference parameters in procedure calls, or without an index. If you want to move several words from a system table, use either an MFDS or memory-resident table pointers and a FOR loop.
- * I haven't the foggiest notion of what would happen if you specify an invalid index when using a memory-resident table pointer (i.e. a negative index or one that is greater than the size of the table). I'd guess that if you pass an index greater than the table size, it would just get you data from whatever happens to be in memory at the system table base + the index, but if you specify a negative index or an index that would cause the effective address (the system table base + the index) to be outside of the memory bank in which the system table is located, be prepared for a system failure.
- * Remember that relatively few system tables are accessible in this way.

Personally, I do not often use this method of system table access. For one, as I said before, I like to consistently use one approach, and

not confuse myself with many different ones -- my favorite is MFDS/MTDS. Furthermore, since I have allocated a special array for each system table, and have DEFINES that access that array, I usually end up having to copy an entire table anyway (not just a single word); however, if you use the alternative approach described above (i.e. having DEFINES specify only the index into the entry without the array name), you wouldn't have to move the entire table entry.

Sometimes, I do use memory-resident table pointers, primarily when speed is of the essence -- I've been told that LST and SST are faster than MFDS and MTDS. It's mostly a matter of personal preference -- use whichever approach you find most appropriate.

An example of this approach could be the following procedure that determines whether it's running on MPE IV or MPE V:

```
LOGICAL PROCEDURE MPE'V;
BEGIN
<< result := TRUE if MPE V, FALSE if MPE IV >>
  INTRINSIC GETPRIVMODE;
  INTEGER POINTER TAB'PCB = 3;
  DEFINE PCB'ENTRY'LEN = TAB'PCB (1) #;
  EQUATE MPE'V'ENTRY'LEN = %25;
  TURNOFFTRAPS;
  GETPRIVMODE;
  IF PCB'ENTRY'LEN=MPE'V'ENTRY'LEN THEN
    MPE'V:=TRUE
  ELSE
    MPE'V:=FALSE;
END;
```

Note that the PCB table entry size changed from %20 in MPE IV to %25 in MPE V. Since the entry size is stored (in both MPE IV and MPE V) in word 1 of the PCB table, we can just look at that word and see if it's %25 or not.

This procedure is exceptionally useful for writing programs that work on either MPE IV or MPE V, or at least abort if they're run on a version of MPE other than the one they were written for. If your program is written for MPE IV, it's much better to abort with a nice error message when run on MPE V rather than crashing the system.

Note however that if you wanted to, you could perform the same task using MFDS.

ABSOLUTE MEMORY LOCATIONS

Some data is stored not in data segment-relative locations, but rather absolute memory addresses. For instance, there is an area in memory called "SYSGLOB" (which stands for SYStem GLOBal), which contains certain interesting pieces of information, like the current console device, the global allow mask, and lots of other goodies. It happens to be stored at a fixed address -- it goes from locations %1000 to %1377 in memory bank 0, and can be accessed using the "ABSOLUTE" construct of SPL. For instance to determine the system console logical device number (which is stored in location %74 of the SYSGLOB), you'd do something like:

```

TURNOFFTRAPS;
GETPRIVMODE;
CONSOLE'LDEV:=ABSOLUTE (%1074);
GETUSERMODE;
TURNONTRAPS;

```

Simple -- think of "ABSOLUTE" as an integer array whose 0th word is memory address 0 of bank 0. To store a value into this location, just say "ABSOLUTE (%1074) := NEW'CONSOLE'LDEV;". Note however that unlike a normal array, you can only read and write individual absolute locations; you can't do a MOVE or SCAN with an absolute address, nor can you pass it by reference to a procedure. In this respect, it's like the system table pointers discussed above.

To confirm your results, or to access absolute locations without a program, you can use privileged mode DEBUG's "DA" (Display Absolute) command:

```

:DEBUG

*DEBUG* PRIV.xx.xx
?DA1074,I
A1074 +00020
?E

```

Also note that DEBUG has a "DSY" (Display SYstem global) command, with "DSY n" being equivalent to "DA 1000+n" -- it just gets you the nth word of SYSGLOB.

One word of caution: ABSOLUTE can only be used to access memory locations with addresses 0 to 65535 -- those memory locations in the so-called "memory bank 0". HP has recently been on a campaign to reduce the amount of data that must be stored in bank 0 -- tables like the PCB, CST, DST, and others that used to be always in bank 0 in MPE IV are no longer always in bank 0 in MPE V. The upshot of this is that since the only things that can be accessed using ABSOLUTE are the ones that are guaranteed to be stored in bank 0, you should avoid using ABSOLUTE whenever a non-bank 0 dependent access method (e.g. MFDS) is available.

DISC RESIDENT SYSTEM TABLES, PART I -- FILE LABELS

Unlike process information, data segment information, and job information, which are stored in memory, file information -- the size of a file, its file code, lockword, location on disc, etc. -- can not be stored in memory because it has to stay around through system failures.

Most of the important file information, including all of the stuff that all modes of :LISTF show, is stored on disc in "file labels". A file's file label is pointed to by its directory entry and occupies one sector (128 words) on disc. Since it's stored in disc rather than in memory, accessing it is rather different from accessing data segments.

In order to access -- read or write -- a file's file label you must first find out the logical device number (LDEV) of the disc on which the file label resides and the sector number of the file label on that disc. This is by no means a trivial task, and in many instances is a lot more complicated than actually accessing the file label once you have this. Logical device numbers and sector addresses are stored in various system tables, in a variety of formats (which we'll discuss later). However, for the duration of this discussion, we'll assume that you are trying to read or write the file label of a file that you have opened; that way, you can figure out the file label address by calling FGETINFO (or FFILEINFO mode 19).

Now you have the file label address -- a doubleword, containing the LDEV in the 8 most significant bits and the sector number in the 24 least significant bits (a format we'll call type L). All you need to do is to perform the actual I/O.

The fundamental file label I/O procedure is, not surprisingly, FLABIO:

```
INTEGER PROCEDURE FLABIO (LDEV, ADDRESS, CMD, FLAB);
VALUE LDEV, ADDRESS, CMD;
INTEGER LDEV, CMD;
DOUBLE ADDRESS;
INTEGER ARRAY FLAB;
OPTION EXTERNAL, UNCALLABLE; << must be called from PM >>
```

Its operation is quite simple -- you pass to it the LDEV and sector address, CMD=0 to read or CMD=1 to write, and the 128-word array into which the file label is to be read or from which it is to be written. The result returned is 0 if all is OK, 1 if there was a "soft error" doing the I/O (I think that this means that the check sum in the file label being read is wrong), and 2 if there was a "hard error" doing the I/O (I think that this means a physical I/O error). Simple enough, once you have the file label's address.

So, we can write the following procedure:

```
<< Read the file label of the file opened as FNUM into the array
FLAB; return true if all OK, FALSE if failed. >>
LOGICAL PROCEDURE FREADFILELABEL (FNUM, FLAB);
VALUE FNUM;
```

```

INTEGER FNUM;
ARRAY FLAB;
BEGIN
INTRINSIC
  FGETINFO,
  GETPRIVMODE;
INTEGER LDEV;
DOUBLE ADDRESS;
INTEGER ARRAY ADDRESS'I(*)=ADDRESS;

  INTEGER PROCEDURE FLABIO (LDEV, ADDRESS, CMD, FLAB);
  VALUE LDEV, ADDRESS, CMD;
  INTEGER LDEV, CMD;
  DOUBLE ADDRESS;
  INTEGER ARRAY FLAB;
  OPTION EXTERNAL, UNCALLABLE;

```

<< To be really safe, you should have code here that checks to make sure that reading 128 words into FLAB won't cause a bounds violation. For more info on this, see below in the discussion of disc address validity checking. >>

```

FFILEINFO (FNUM, 19, ADDRESS);
IF <> OR ADDRESS=0 THEN
  << either file not opened or not on disc (address=0) >>
  FREADFILELABEL:=FALSE
ELSE
  BEGIN
  LDEV:=ADDRESS'I(0).(0:8); << high-order 8 bits >>
  ADDRESS'I(0).(0:8):=0; << ADDRESS now is sector address >>
  TURNOFFTRAPS; << See above (MFDS) to learn about this >>
  GETPRIVMODE;
  IF FLABIO (LDEV, ADDRESS, 0 << read >>, FLAB) <> 0 THEN
    FREADFILELABEL:=FALSE
  ELSE
    FREADFILELABEL:=TRUE;
  << no GETUSERMODE -- also see above to learn about this >>
  TURNONTRAPS;
  END;
END;

```

Using this procedure, you can now read the file label of any file you have opened, and find out a lot of information that FGETINFO and FFILEINFO won't give you -- things like the file creation date, last access date, last modification date, the file's extent map, lockword, and other useful things. A similar procedure can be written to write out an open file's file label -- just be sure that you don't use to change things that are also kept in memory-resident file tables that exist for open files (like the FCB, ACB, etc. -- see the System Tables Manual).

Similarly, say that you want to read the file label of \$OLDPASS *without* opening it. Why would you want to? Well, in the case of \$OLDPASS you usually wouldn't (except if you want to save time) -- however, you might want to do this for other files that you can't open

so easily, like spool files or other people's temporary files or \$OLDPASSes.

The first thing you'd have to do is to find out where the file label address for \$OLDPASS is located. When you can recite this from memory, you are truly a superior system programmer -- mere mortals would have to scan through the System Tables Manual, make an educated guess or two, and find it in the Job Information Table (chapter 8), words 36 and 37. Looks simple enough -- there is your file label address; all you have to do is extract the LDEV from the 8 high-order bits, the sector address from the remainder, and call FLABIO. Right? Wrong.

The fundamental means of doing the I/O are still the same -- you get the LDEV, get the sector address, and call FLABIO. However, what is stored in the high-order 8 bits of words 36 and 37 of the JIT is NOT (!!!) the logical device number of the disc. The \$OLDPASS address in the JIT is what I call the "type V address" (as opposed to the "type L address" described earlier) -- its high-order 8 bits are the so-called "volume table index" of the disc on which the sector address is contained.

So, you have a problem. You are trying to get the logical device number of the disc on which the file label is located and the file label's sector address. You have the file label's sector address, but instead of the LDEV, you have a volume table index (VTABX).

What you need to do is to convert the VTABX into LDEV, and to do this you use a system internal procedure called LUN:

```
INTEGER PROCEDURE LUN (VTABX, MVTABX);
VALUE VTABX,
      MVTABX;
INTEGER VTABX,
      MVTABX;
OPTION EXTERNAL, UNCALLABLE;
```

What LUN (which I suspect stands for "Logical Unit Number", another name for LDEV) does is take a VTABX and a MVTABX (Mounted Volume Table Index) and return the LDEV which they describe. Where do you get the MVTABX? Well, in this case, it is also (fortunately) stored in the JIT, in word 57.

So, to read \$OLDPASS's file label, you'd do the following:

- * First, read the JIT into an array (you can use the DSEGREAD procedure we described earlier to do this).
- * Then, take the \$OLDPASS type V address (words 36 and 37), and separate out the VTABX (high-order 8 bits) and the sector number (low-order 24 bits).
- * If \$OLDPASS's address is 0, GO NO FURTHER -- it means that no \$OLDPASS file exists.

- * Call LUN, passing to it the VTABX and the MVTABX (from word 57 of the JIT).
- * Make a prayer that you did everything right ("Oh Lord, watch over this humble program and prevent it from crashing the system").
- * Call FLABIO, passing to it the LDEV you got from LUN and the sector address you got from the low-order 24 bits of words 36 and 37 of the JIT.
- * I don't suppose I need to tell you to experiment with CMD=0 (read), not CMD=1 (write)...

Voila! All you ever wanted to know about FLABIOing in ten pages or less...

Incidentally, you're probably wondering what all this VTABX and MVTABX nonsense is about. Well, it was implemented to support Private Volumes, in which the logical device number of a disc is not known until the disc is actually mounted -- thus, the directory on the disc would have to contain not LDEVs, but VTABXs. If you expect your program to NEVER need to be run on Private Volumes, you can forget about MVTABXs, and just call LUN with an MVTABX of 0. However, you still have to call LUN, because even if your system never even smelled a private volume, the VTABXs need not correspond to the LDEVs.

If you intend to use FLABIO often in your programs, I would suggest that you write the following procedures:

- * FLABREAD and FLABWRITE, which take a type L (LDEV and sector) address, and either read into or write from a user-specified array. For consistency's sake, they should return a file system error number (e.g. FLABIO error 1 would map into file system error 108 [INVALID FILE LABEL]; error 2 would map into file system error 47 [I/O ERROR ON FILE LABEL]). Also, since passing an out-of-range LDEV or sector to FLABIO will cause a system failure, this procedure can check the address using some tricks I'll talk about presently.
- * V'TO'L'ADDRESS, which takes a type V (VTABX and sector) address and a MVTABX, and returns the type L address. This would call LUN to convert the VTABX and MVTABX into the LDEV, and would allow you to easily read or write a file label given a type V address:

```
FLABREAD (V'TO'L'ADDRESS (V'ADDRESS, MVTABX), FLAB);
```

- * FNUM'TO'L'ADDRESS, which takes the file number of an open file and returns the type L address. Simply calls FFILEINFO or FGETINFO. Again, makes reading/writing file labels given an FNUM easier:

```
FLABREAD (FNUM'TO'L'ADDRESS (FNUM), FLAB);
```

- * Finally, FILE'TO'L'ADDRESS, which takes a filename and returns the type L address. The simplest way to do this is to FOPEN the file, call FNUM'TO'L'ADDRESS, and FCLOSE the file. A far more difficult approach, which however is faster and doesn't care about file security, lockwords, or whether the file is already opened, is to get the file label address directly from the directory. For this, you'd have to call the DIRECFIND procedure to get the sector address from the directory, call DIRECFIND again to get the directory entry for the group in which the file resides (to get the group's MVTABX), and then call V'TO'L'ADDRESS to convert the type V address that is stored in the directory to a type L address. For the beginning, stick to the simple approach -- the directory is probably worth a whole paper dedicated to it alone, and will not be elaborated further in this document.

The algorithm of checking a type L address for validity is not trivial, but still well worth implementing (unless you want to get a reputation as "Mike 'System Failure' Johnson"):

- * Call the CHECKDISC procedure:

```
PROCEDURE CHECKDISC (LDEV, STATUS);
VALUE LDEV;
INTEGER LDEV;
LOGICAL STATUS;
OPTION EXTERNAL, UNCALLABLE;
```

Pass to it the LDEV, and make sure that the low-order 3 bits (bits .(13:3)) of STATUS are all 0 -- if bit 15 is set, this means the LDEV is out of range; if bit 14 is set, the LDEV is not configured; if bit 13 is set, the LDEV is not a disc. If none of these bits is set, you know you have a good LDEV.

- * Call the DISCSIZE procedure:

```
DOUBLE PROCEDURE DISCSIZE (LDEV);
VALUE LDEV;
INTEGER LDEV;
OPTION EXTERNAL, UNCALLABLE;
```

Pass to it the LDEV, and make sure that the sector address is greater than or equal to 0 and less than the number DISCSIZE returned to you (which is the number of sectors on the device).

- * If you're a general-purpose procedure, check the buffer address passed to you to make sure that it's within bounds, i.e. that its start is at DL or above, and that its 127th word is at Q or below. Remember that bounds violations are not checked for in privileged mode, and trying to write to an out-of-bounds address would cause whatever is at that address (i.e. in someone else's space) to be over-written.
- * If either check failed, abort -- you've got a bad address, which if used in an FLABIO call would crash the system.

If you want to check the results of your programs, or dabble with file labels without writing a program, there are several utilities you can use:

- * For some specific file label retrieval and modification tasks, VESOF's own MPEX, which allows you to easily look at files' creator ids, access/modify/creation dates, etc., and modify many file attributes (like creator id).
- * HP's DISKED2, which allows you to read and write arbitrary disc locations. If you intend to modify file labels, make sure that you are either using the ">FILE" command of DISKED2 to get to the file label's sector (rather than just a >DISC and >MODIFY) or set the file label's checksum (word 34) to 0. If you use >DISC and >MODIFY but do not set the checksum to 0, the file system will not update the checksum and will think that the file label is corrupted.
- * Privileged mode DEBUG, which has a "DV" command (Display Virtual memory, a misnomer -- really displays an arbitrary disc sector). Note that the syntax of this command, especially when the sector address is more than 32768, is non-trivial; see the DEBUG Manual.
- * FLUTIL3, a contributed program which allows you to read and write the file label. It's easier to use than DISKED2, and is also safe for modifying file labels.

Finally, a caveat to the user who knows about and uses ATTACHIO: resist the temptation to do file label I/Os using ATTACHIO instead of FLABIO. On reads, calling FLABIO is better because it checks the file label's check sum, letting you know if the file label appears corrupted. On writes, it is IMPERATIVE that you call FLABIO, because otherwise the check sum will not be updated, and next time the system wants to read the file label, it will think that it's corrupted.

As a final present to all you file label hunters out there, the following table should tell you how to get file addresses:

- * Permanent files -- FOPEN the file and call FGETINFO/FFILEINFO (type L address); if you want to be fancy, call DIRECFIND to get the file's directory entry (type V address).
- * Your session's temporary files -- FOPEN/FGET(FILE)INFO (type L address), or get type V address from your JDT (described in System Tables Manual Chapter 8; pointed to by the PKGLOB, which is also described in Chapter 8).
- * Your session's \$OLDPASS -- FOPEN/FGET(FILE)INFO (type L address), or get type V address from your JDT (described in System Tables Manual Chapter 8; pointed to by the PKGLOB, which is also described in Chapter 8).
- * Other sessions' temporary files and \$OLDPASSes -- type V addresses stored in each session's JIT or JDT.

- * Spool files -- IDD or ODD system tables (System Tables Manual Chapter 14); type L.

Good luck, and happy hunting.

DISC RESIDENT SYSTEM TABLES, PART II -- USING ATTACHIO

Other information besides file labels is also stored on discs -- things like disc labels, defective track tables, the contents of files themselves, and so on. To get at them, you have to use a privileged system internal procedure called ATTACHIO.

ATTACHIO is *the* primitive I/O procedure. You can use it to do any arbitrary operation against any arbitrary device, from reading and writing discs to reading, writing, and doing various control functions to non-disc devices. All file system I/O functions eventually end up as ATTACHIO calls.

The calling sequence for ATTACHIO is:

```
DOUBLE PROCEDURE ATTACHIO (LDEV,QMISC,DSTX,ADDR,FUNC,CNT,P1,P2,FLAG);
VALUE LDEV, QMISC, DSTX, ADDR, FUNC, CNT, P1, P2, FLAG; INTEGER LDEV,
QMISC, DSTX, ADDR, FUNC, CNT, P1, P2, FLAG; OPTION EXTERNAL, UNCALLABLE;
```

This is quite a mouthful (just what *is* a QMISC?). Fortunately, however, for discs these parameters are rather simple:

- * LDEV: This is self-explanatory. I'm sure I need not tell you that passing an incorrect LDEV will cause a system failure (#206, to be precise).
- * QMISC: 0. QMISC could have a lot of different values for non-disc devices, but for disc devices 0 will do quite well.
- * DSTX: Technically, the data segment number of the segment to/from which the read/write is to take place. 0 means your stack, and this is the value you'd use most often.
- * ADDR: The address (within the data segment indicated by DSTX) to/from which the read/write is to take place. When DSTX = 0 (i.e. your stack), this is the ordinary word address of the array which you're using for your I/O. Don't forget to have room for as much as you want to read/write in the array, since like all other privileged internal procedures, this one doesn't do ANY bounds checking. Also note that if you're reading into a stack array, you should be sure to pass the ADDRESS of the array (i.e. "@arrayname"). "ATTACHIO (1,0,0,@FOO,0,10,ADDR'HI,ADDR'LO,1)" will read into the array FOO; if you omit the "@" before the FOO, you will read into the stack address pointed to by the 0th word of FOO (which is probably not what you want).
- * FUNC: Curiously enough, the function. 0 means read, 1 means write. Try something else. I dare you.

- * CNT: If positive, this is the number of words to read or write; if negative, the absolute value of this is the number of bytes to read or write. Thus, 10 means 10 words; -20 means the same thing (20 bytes).
- * P1: The high order word of the sector number for the I/O. Don't forget that all reads and all writes must start at a sector boundary (although CNT need not be a multiple of the sector size).
- * P2: The low order word of the sector number. If you have a double word address D, you can say `INTEGER D0=D, D1=D+1`; D0 will now refer to the high-order word of D and D1 to the low-order word of D. Or you can calculate the high-order word of D on the fly by saying `"INTEGER(D&DLSR(16))"` and the low-order word by saying `"INTEGER(D)"`.
- * FLAG: 1. This means "blocked, wait until request is complete" -- the I/O system's equivalent of ordinary file I/O (as opposed to no-wait I/O).
- * Function returns a double-word:

Word 0 bits 8:5: qualifying status for the I/O; this further describes the result of the operation. The general status (see below) is usually all you need to find out what happened.

Word 0 bits 13:3: general status of the I/O: 0: Pending. I don't think this should ever happen with flags = 1. 1: Normal. 2: End of file. I can't see how this could happen on disc (remember, ATTACHIO knows nothing about disc files). 3: Unusual condition. The qualifying status probably describes this better -- try figuring it out. 4: Irrecoverable error. Again, look at the qualifying status if you can figure it out.

Word 1: number of words (positive) or bytes (negative) transferred.

Incredibly, that's all there is to it. Once you've got an LDEV and a sector address, just plug them into the right parameters, and call away! Of course, remember to enter privileged mode and turn off traps (just as you would with FLABIO or an MFDS/MTDS). Also, remember that many disc addresses (the type V addresses I mentioned in the ACCESSING FILE LABELS CHAPTER) contain not an LDEV, but rather a VTABX which you'll have to convert into an LDEV. But, once you get all these preliminaries out of the way, calling ATTACHIO is just as easy as calling FLABIO.

For instance, say that we want to determine the number of defective tracks on a given ldev.

First, as always, we have to find out where this number is stored. System tables Chapter 3 contains a lot of useful information on disc

format, including the fact that sector 1 of every disc contains the disc's defective track table, and word 0 of the sector contains the actual number of defective tracks.

Thus, our program would look something like:

```

INTEGER PROCEDURE NUM'DEF'TRACKS (LDEV);
VALUE LDEV;
INTEGER LDEV;
BEGIN
  INTRINSIC GETPRIVMODE;
  EQUATE SIZE'DEF'TRK'TAB = 128;
  ARRAY DEF'TRK'TAB (0:SIZE'DEF'TRK'TAB-1);
  DEFINE TAB'NUM'DEF'TRACKS = DEF'TRK'TAB (0) #;
  INTEGER DISC'STATUS;
  DEFINE LDEV'IS'DISC = (DISC'STATUS.(13:3) = 0) #;
  EQUATE FUNC'READ = 0; << ATTACHIO read function >>
  EQUATE DISC'IO'FLAGS = 1; << ATTACHIO flags for a disc I/O >>

  PROCEDURE CHECKDISC (LDEV, STATUS);
  VALUE LDEV;
  INTEGER LDEV;
  LOGICAL STATUS;
  OPTION EXTERNAL, UNCALLABLE;

  DOUBLE PROCEDURE ATTACHIO
    (LDEV, QMISC, DSTX, ADDR, FUNC, CNT, P1, P2, FLAG);
  VALUE LDEV, QMISC, DSTX, ADDR, FUNC, CNT, P1, P2, FLAG;
  INTEGER LDEV, QMISC, DSTX, ADDR, FUNC, CNT, P1, P2, FLAG;
  OPTION EXTERNAL, UNCALLABLE;

  TURNOFFTRAPS; << See MFDS chapter for a discussion of this >>
  GETPRIVMODE;
  CHECKDISC (LDEV, DISC'STATUS);
  IF NOT LDEV'IS'DISC THEN
    NUM'DEF'TRACKS:=-1 << error, not a valid sic >>
  ELSE
    BEGIN
      ATTACHIO (LDEV, <<qmisc>> 0,
        << Amazingly dreadful things will happen if we leave
          off the "@" in "@DEF'TRK'TAB >>
        <<dstx>> 0, <<address>> @DEF'TRK'TAB,
        FUNC'READ, <<cnt>> SIZE'DEF'TRK'TAB,
        <<sector address:>> 0, 1,
        DISC'IO'FLAGS);
      NUM'DEF'TRACKS:=TAB'NUM'DEF'TRACKS;
    END;
  END;

```

That's all. Relatively simple, and because of the CHECKDISC call that ensures the LDEV is in range, configured, and a disc, no danger of system failure.

Of course, this particular ATTACHIO calling sequence is only relevant to discs; terminals, printers, tape drives, and other devices have their own parameters for QMISC, FUNC, P1, P2, and FLAGS, which I will not explain here for lack of space and because they really are not relevant to system table access.

SIRS, AND WHAT HAVE THEY DONE TO GET SUCH DEFERENCE?

Whenever you are accessing (reading or writing) a system table that somebody else might be modifying, you have to beware of something being changed out from under you.

For instance, say that you're reading the ODD -- the Output Device Directory, a rather odd name for the table that contains information on all the output spool files in the system. The ODD is structured as several linked lists of entries, one for each spooled device or device class. The way that you'd read it is that you'd read one entry, process it, get the address of the next entry from the current entry, get the next entry, process it, and so on. However, say that while you're processing this entry, the next entry in the list gets deleted. Then, when you try to get the next entry from the location that is pointed to by the current entry, you'll get garbage.

Even worse, say that you read the entry, process it, make some changes to the copy of the entry you keep in your stack, and then try to write it back out. Then, if the entry is deleted between the time you read it and write it back out, you stand the risk of over-writing whatever new entry might have been put there, probably with disastrous consequences.

This is by no means a new problem -- it arises in file and database systems all the time. The general solution to this is some kind of locking mechanism, and that is precisely what SIRs are for.

SIR stands for System Internal Resource. You may lock it by calling the privileged system internal procedure GETSIR and release it by calling the privileged system internal procedure RELSIR. Since all (in theory) processes that modify a system table must lock its SIR before starting the modification, if you lock the SIR you are guaranteed that no other process will modify the table until you unlock it.

Most system tables that are usually modifiable by more than one process at a time (including things like the PCB, JMAT, ODD, etc., but excluding JITS and JDTs, because these are session-local tables and are thus very rarely concurrently accessed) have a SIR. The SIR assignments are listed in Chapter 5 of the System Tables Manual.

Before using GETSIR and RELSIR, you have to declare them in your program as follows:

```
INTEGER PROCEDURE GETSIR (SIR'NUMBER);  
VALUE SIR'NUMBER;
```

```
INTEGER SIR'NUMBER;  
OPTION EXTERNAL, UNCALLABLE;
```

```
PROCEDURE RELSIR (SIR'NUMBER, GETSIR'RESULT);  
VALUE SIR'NUMBER, GETSIR'RESULT;  
INTEGER SIR'NUMBER, GETSIR'RESULT;  
OPTION EXTERNAL, UNCALLABLE;
```

When you want to get a SIR, you figure out its SIR number and pass it to GETSIR, saving the result returned by GETSIR. Then, to release it you call RELSIR, passing to it the SIR number and the result returned by GETSIR (this is very important!). For instance, say you want to go through the ODD, which is a linked list of entries, without being afraid that somebody might be in the middle of adding or deleting an entry, causing you to encounter a bad link. What you'd do is:

- * Find out the ODD SIR number from the System Tables Manual Chapter 5 (it's 4).
- * Lock the ODD SIR by calling GETSIR, saving the result in some variable -- have a special variable, reserved for this purpose (say, ODD'GETSIR'RESULT), that you are certain will not be accidentally changed -- I and J are definitely out of the question.
- * Do whatever you want to do with the ODD, remembering that if you abort for any reason before you unlock the SIR, THE SYSTEM WILL CRASH. Also, be sure that neither you nor any procedure you might call while you have the SIR tries to get a SIR whose priority number (see the discussion below) is lower than the ODD's.
- * When you're done, call RELSIR, passing to it the ODD SIR number and the value that GETSIR returned.
- * Breathe a sigh of relief that you haven't crashed the system.

Similar procedures are used for locking other SIRs -- however, note that if you have more than one SIR locked at the same time, you must unlock them in the REVERSE ORDER of locking. Also note that you won't get any problems if you try to lock a SIR twice -- say, you lock it, and another procedure you call locks it again; just remember to unlock it twice, too (in the case both you and another procedure locks it, all that is necessary is that both the other procedure and you unlock it, again in the reverse order that you two locked it in).

Watch out, though -- when you're dabbling with SIRs, there are a lot of possible pitfalls involved:

- * Locking a SIR is not just a nice thing to do that'll avoid problems for you. If you modify a table without first locking its SIR, you get everybody else in deep trouble, much like if you were modifying a shared file without first FLOCKing it. Unfortunately, unlike KSAM files and IMAGE databases (but like MPE flat files), no checking is done to ensure that you've gotten the right SIR --

it's your responsibility, and if you don't live up to it, may God have mercy on your system.

- * Like other resources, it's very easy to get into deadlock trouble when you're getting more than one SIR. Say that you get SIR A and then try to get SIR B. SIR B is locked by another process, so you're suspended until the other process unlocks the SIR. Meanwhile, the other process tries to get SIR A, and suspends waiting for you to unlock it! Both of you are waiting for each other to release a SIR, and you'll stay that way until the system is rebooted. What's worse, anyone else who tries to lock either SIR will be suspended, too, causing an ever-growing number of suspended problems, and making the system come to a grinding halt. To avoid this, there is a certain very fixed and unalterable SIR locking sequence that you MUST obey. It is described in Chapter 5 of the MPE V System Tables Manual, where each SIR is given a priority number, and you must never lock a SIR whose priority number is lower than that of one you already have locked. Note that a similar table in Chapter 5 of the MPE IV System Tables Manual is quite incomplete -- try to get the MPE V locking sequence (which, to the best of my knowledge, is the same as that for MPE IV, although it is described much better). When you're debugging SIR code that you're afraid might deadlock, it's a good idea to keep an OPT or a SOO process running on some other terminal. OPT and some versions of SOO have commands that allow you to see who is holding what SIRs -- this can help you find the cause of the deadlock.

- * Not only must you watch out to make sure that you lock all SIRs in the right order, you must also be certain that all your callers and called procedures do too. Thus, if you want to lock the FILE SIR (which must never be locked before the FMAVT [File Multi-Access Vector Table] SIR), you must be sure that you do not call any procedure that tries to lock the FMAVT SIR, since that would be a SIR locking sequence violation. Since virtually all file system intrinsics can under some conditions try to lock the FMAVT SIR, you must either lock neither the FILE SIR nor the FMAVT SIR or both the FILE SIR and the FMAVT SIR before calling a file system intrinsic. That goes for all other procedures you call -- you better know what SIRs they try to lock, and you better make sure that your SIR locking sequences are appropriate. Similarly, if a procedure that you write uses SIRs, you have to be sure that all your callers realize this and make sure that they either don't have any SIRs locked when they call you, or they are certain that your and their locking sequences mesh well.

- * If your process terminates itself for any reason (TERMINATE, QUIT, STACK OVERFLOW, whatever) while you have a SIR locked, you get a System Failure 314. You best be VERY careful. Note that you need only be afraid of your process terminating itself -- while you hold a SIR, you can't be killed from outside (say, by an :ABORTJOB); if someone tries, you'll keep on processing until you release your last SIR, and then you'll die.

- * Always disable break before locking a SIR. This is not because you can be :ABORTed from break -- if someone tries, you'll keep on going until you release your SIRs, and only then will the abort take hold. Rather, if you let a user hit break while you've got a SIR, the user might try to execute an MPE command that tries to get the same SIR, and you'll get a deadlock -- the command waiting for the SIR (which is held by your program) and your program waiting for MPE to :RESUME it (which it can't because it's waiting for a SIR). Deadlocks work in mysterious ways.
- * Never do any terminal I/O while you have a SIR locked. In fact, do not perform any task that you suspect might take a long time to finish -- like issuing a request for a tape, reading a possibly empty message file, etc. The reason for this is simple -- if a process that has a SIR suspends, everybody else who wants the SIR will suspend, too, thus effectively stopping the system until the SIR holder is re-activated. The last thing you want is for the system to come to a grinding halt because someone went on a coffee break while a SIR-holding program was expecting input from him. As I said, this is also relevant for terminal I/O because a control-S on the terminal will suspend the writing program until a control-Q is hit.
- * Always be sure that the GETSIR result you pass to a RELSIR is the result returned by the GETSIR that got the particular SIR involved. If not, three guesses as to what happens...
- * Always be sure that you unlock SIRs in the inverse order of locking them, under penalty of you-know-what.

This is an array of warnings that should make the bravest programmer cringe. Unfortunately, if you want to modify a system table or even read system tables that you fear might change out from under you, you have to lock the SIR. So, how can you do this without fearing a system failure?

- * Try to avoid locking SIRs whenever possible. In MPEX, I used to lock the FMAVT SIR and the FILE SIR (the two SIRs you should lock when doing certain operations on files -- always lock them in that order) when I modify a file's file label. Recently, I've changed it so that I wouldn't modify the file label without first FOPENing the file exclusively -- that way, I'm guaranteed that no-one else (except possibly :STORE/:RESTORE) is dabbling with the file, and I no longer need to lock the SIR. This avoids risk of deadlocks and system failures.
- * When you need to get a SIR, release it as soon as possible. Every statement between a GETSIR and a RELSIR is just one more opportunity for a disastrous (system-crashing) abort to occur.
- * Write three procedures, SIR'GET, SIR'RELEASE, and SIR'RELEASE'ALL.
 - SIR'GET should take a SIR number, turn off break (see above), lock the SIR, and save the SIR number and the GETSIR result in a

global array. This global array should be shared between these three procedures, and should actually be treated as a stack, with new entries being added to the end, and old entries being deleted (by SIR'RELEASE) in a last-in, first-out fashion.

- SIR'RELEASE should take a SIR number, make sure that it's the most recently gotten SIR, get the GETSIR result from the last-added entry in the global array, and call RELSIR.
- SIR'RELEASE'ALL shouldn't take any parameters, but should only release all the SIRs mentioned in the global array maintained by SIR'GET and SIR'RELEASE. It should be called by any procedure that wants to terminate or abort in any fashion. For instance, the DSEGREAD procedure that I talked about above does a MFDS, checking its parameters for validity first, and aborting if the parameters are invalid (to avoid a system failure). If you ever call it while you have a SIR locked, put a call to SIR'RELEASE'ALL into it right before it aborts. That way, if you have a bug and DSEGREAD aborts, it'll release all the SIRs you have locked before aborting, thus preventing a system failure.

I use these procedures exclusively, almost never calling GETSIR and RELSIR outside them, and my MPEX (or any of VESOPT's other privileged mode-using programs) has never crashed the system in production by aborting while holding a SIR (in fact, they've never crashed a production system, period). Sometimes, we get a letter containing a PSCREEN of an abort caused by an MPEX bug (yes, even MPEX has bugs) with the message "DSEGREAD passed bad parameter, SIRs released", and I feel kind of proud -- sure, MPEX may have a bug, but even though it was trying to read a non-existent segment while it had a SIR locked, it just aborted with a nice message instead of crashing the system.

The number one cause of user program-caused system failures is not carelessness, but laziness -- the user didn't spend the time to develop some simple utility procedures that would prevent minor bugs or typos from causing system failures.

A final aside on the topic of SIRs -- sometimes (fortunately, very rarely) you may want to ensure that NOBODY else on the system is doing anything. Essentially, you want to turn off interrupts to make sure that until you turn them back on, you and you alone will use the CPU. This is kind of a "super-SIR" -- you don't just lock some table, you lock the entire system.

I don't like this, and I've never felt the need to do this. For one, you can crash the system this way easy as pie -- from what I understand, any interrupt, including one caused by a request on your part to access a data segment that isn't currently in memory would cause a system failure. Furthermore, this isn't as good as a SIR in at least one way -- when you lock, say, the JMAT SIR, you can be sure that you will wait until anybody who locked the SIR to modify the JMAT releases it; that way, when you get it, you know that nobody else is dabbling with the table. This is not so for turning off interrupts --

the process which is modifying the table you want to modify may have just been swapped out, and the table might be in a temporarily inconsistent state. Finally, when you turn off interrupts, you also turn off clock interrupts, and the system clock is essentially not running until you turn interrupts back on.

As I said, I personally have never needed to do this, and don't foresee myself doing it in the future. However, if you feel up to it, there are two machine instructions that do this -- PSDB (PSeudo-DisaBle) and PSEB (PSeudo-EnaBle).

PSDB disables process dispatching, thus assuring that you'll be the only one to run until you do a PSEB. Unfortunately, this may (or may not -- I'm not sure) mean that the system will crash if you try to access a data segment that is not in memory, or do any other kind of disc I/O. This means that about the only thing you can safely do between a PSDB and PSEB is to access system tables that you know are always memory-resident (e.g. CST, DST, PCB, etc.).

If you have any luck doing this, let me know -- I'd like to see how this should really be done.

SYSTEM TABLES AND SPL PROGRAMMING STYLE

Everybody and his brother have their own opinions about programming. The last thing that I want to do is to enter this controversial and generally thankless field. However, in my experience with using system tables in SPL, I discovered certain useful programming guidelines that I want to mention in passing.

The major problem in working effectively with system tables is that virtually nobody can remember just what the 5th bit of the 53rd word of an entry in data segment number 22 contains. Furthermore, it is rather cumbersome to always flip through the System Tables manual whenever you're writing or reading programs. Comments help some for program readability, but you'd still have to look at the manual while writing, and you also stand the risk of incorrect comments.

What one really wants is record structures (like in PASCAL). You declare a data type PCB'ENTRY'TYPE, declare a data structure of this type called PCB'ENTRY, read the entry into PCB'ENTRY, and then just refer to the PCB entry fields as PCB'ENTRY.FATHER'PIN, PCB'ENTRY.PRIORITY, PCB'ENTRY.CS'QUEUE, etc. If you have another PCB entry you want to simultaneously work on, just put it into another data structure -- ANOTHER'PCB'ENTRY -- and use ANOTHER'PCB'ENTRY.FATHER'PIN, etc.

Unfortunately, SPL does not have these kinds of record structures. However, they can be cleverly emulated.

What I do is that I declare an array called PCB, and I set up a number of DEFINES -- PCB'FATHER'PIN, PCB'PRIORITY, PCB'CS'QUEUE, etc. -- that refer to the appropriate fields of the array. Then, when the array

contains a PCB entry, these DEFINES can refer (on either the left or right side of an assignment statement) to the appropriate fields of the entry stored in the array. Note that these DEFINES can refer to individual bits of the array as well as to entire words. For instance, one such definition of some of the PCB entries might look like:

```
<< Definitions of some fields of MPE IV PCB entries. >>
EQUATE SIZE'PCB = 16;
INTEGER ARRAY PCB(0:SIZE'PCB-1);
DEFINE PCB'FATHER      = PCB( 5).( 0: 8) #,
      PCB'PRIORITY    = PCB(13).( 8: 8) #,
      PCB'CS'QUEUE    = PCB(13).( 2: 1) #;
```

Note that I also have an equate for the PCB size; I'd also have an equate for the PCB data segment number, the PCB System Internal Resource (SIR) number (if it had one), equates or defines for all interesting PCB constants (e.g. the possible values of the "process type" field), etc. What's more, I'd put this all into one \$INCLUDE file so that it will be kept in one place and will thus be easily modifiable.

Other tables are a bit more difficult. The JIT (Job Information Table), for instance, contains not just word (integer) and bit field values, but also character arrays and double integers. For that, I equivalence (using the "BYTE/DOUBLE ARRAY xxx(*)=yyy") a byte array and a double array to JIT, the main array (which is an integer array). Thus, the file looks something like:

```
EQUATE SIZE'JIT=61;
INTEGER ARRAY JIT(0:SIZE'JIT-1);
BYTE   ARRAY JIT'B(*)=JIT;
DOUBLE ARRAY JIT'D(*)=JIT;
DEFINE JIT'JOB'ID      = JIT ( 9)      #,
      JIT'MAIN'PIN    = JIT (10). ( 8: 8) #,
      JIT'GROUP'SEC  = JIT'D( 7)      #, << Word 14 >>
      JIT'ACCT'NAME  = JIT'B(32)      #; << Word 16 >>
```

A very few tables contain double integers that are not aligned on an even word boundary (e.g. word 14), but are rather on an odd word boundary (e.g. word 21), and thus can't easily be accessed as elements of a double array equivalenced to the main integer array. For them, you have to equivalence another double array to the 1st (as opposed to 0th, the default) element of the main array, and then reference them as elements of this array. For example,

```
EQUATE SIZE'DIR'GROUP=41;
INTEGER ARRAY DIR'GROUP(0:SIZE'DIR'GROUP-1);
BYTE   ARRAY DIR'GROUP'B(*)=DIR'GROUP;
DOUBLE ARRAY DIR'GROUP'D(*)=DIR'GROUP;
DOUBLE ARRAY DIR'GROUP'D'1(*)=DIR'GROUP(1);
DEFINE DG'CPU'COUNT  = DIR'GROUP'D'1(6) #; << Word 13 >>
```

Another advantage of this scheme is that it makes it much easier to change your programs to work on a new release of MPE in which the

format of a table has been changed -- sometimes you only have to change the \$INCLUDE files and recompile. Also, it makes it surprisingly easy to have the same source code work with two different MPE versions (e.g. MPE IV and MPE V) -- just declare a compile-time switch, say, X5 that is ON when you're compiling the program to run on MPE V and OFF when you're compiling to run on MPE IV. Then, in all your \$INCLUDE files that describe tables that are different in MPE IV and MPE V, just check this switch and depending on its value, declare either the MPE IV or MPE V layouts. For more information on compile-time switches, see the SPL Reference Manual. An example is the PCB:

```

$IF X5=OFF << MPE IV >>
EQUATE SIZE'PCB=16;
INTEGER ARRAY PCB(0:SIZE'PCB-1);
...
DEFINE PCB'STACK'DST    = PCB( 3).( 1:10) #;
...
$IF X5=ON << MPE V >>
EQUATE SIZE'PCB=21;
INTEGER ARRAY PCB(0:SIZE'PCB-1);
...
DEFINE PCB'STACK'DST    = PCB( 3).( 2:14) #;
...
$IF

```

The major problem of this method is that it requires the entry to be stored in a certain fixed place (say, the array PCB). If you have two entries that you want to manipulate at the same time, you're out of luck; if you have an entire array of entries that you want to look through, you have to step through the array, moving each entry to the array PCB before processing the entry. Also, if you want to access system tables using SPL memory-resident system table support constructs (see below), you have to move each word of the table into the array.

One solution to this problem is to have the DEFINES contain only the word number of the element in the table. For instance, for the JIT we might have:

```

EQUATE SIZE'JIT=61;
DEFINE JIT'JOB'ID      = 9 #,
        JIT'ACCT'SEC   = 12 #,
        JIT'ACCT'NAME  = 32 #,    << Byte field! >>
        JIT'ALLOW'1   = 40 #;

```

Then, to get the job id, we'd read the JIT entry into anywhere we want to (say, the array MY'JIT'ENTRY), and access it as MY'JIT'ENTRY(JIT'JOB'ID). This will work not only for fields that are whole words, but also for bit fields:

```

DEFINE JIT'MAX'PRI     = 10).(0:8 #,
        JIT'MAIN'PIN   = 10).(8:8 #;

```

This way, MY'JIT'ENTRY(JIT'MAIN'PIN) would map to MY'JIT'ENTRY(10).(0:8) -- the right value.

Unfortunately, one of the drawbacks of this method is that if you want to retrieve a byte field or a double field, you can't just specify it by name -- you have to explicitly retrieve it as MY'JIT'ENTRY'B(JIT'ACCT'NAME). If you forget and enter MY'JIT'ENTRY(JIT'ACCT'NAME), you'll get a very wrong result.

Also, this method requires more typing -- you have to enter both the field designator (e.g. JIT'ACCT'NAME) and the array name.

Both methods are acceptable approaches, each with its own advantages and disadvantages. Use whichever you prefer, or even your own, but keep in mind the following:

- * Use DEFINES and EQUATES (for field names, entry size, data segment numbers, SIR numbers, possible field values, etc.). If you use "magic numbers" (e.g. bits 12:3 of word 35) in your code, it'll only make it harder to write, read, and maintain.
- * Put DEFINES and EQUATES into an \$INCLUDE file. That way, if you need to change something (because you made an error in putting it in in the first place or because it has changed in the new MPE release), you'll only have to change it in one place.

AND NOW, FOR SOMETHING COMPLETELY DIFFERENT...

Until now, I have been concentrating mostly on describing how system tables can be accessed. To do this, I've introduced some important tables like the PCB and JMAT, but have skimmed on description of what other system tables may exist and what they may contain.

Unfortunately, the System Tables Manual does not really explain anything except table formats. It'll tell you what the ODD looks like, but won't tell you thing one about how it fits into the overall system structure and what you need to know to access it. Once I've whetted your appetite by telling you how to access system tables, I feel obligated to present my concept of how all these tables fit together and where you should go to get a particular piece of data.

As I said before, system tables are just the places where the operating system "keeps its stuff" -- where it stores information on all the objects it must manage. A convenient way to look at system tables is in terms of what objects they describe.

Don't feel nervous if you don't understand the purpose or even the contents of some of the tables I mention below. Some of them are really arcane, and may not be worth much to you anyway. If you're just starting to learn the innards of the system, you might want to skip reading about everything except the simpler job-, process-, and file-oriented system tables.

JOB-ORIENTED SYSTEM TABLES

Almost all the work done on an HP 3000 is done on behalf of a job submitted by the user. Note that whenever I say "job", I mean either a batch job or on-line session, since they are rather similar to the system. From the system's point of view, a job is a collection of processes. The Command Interpreter (CI) of a job is one such process. Any processes you create, either using the :RUN command or using the CREATE or CREATEPROCESS intrinsics, also belong to that job.

Every job has an entry for it in the Job Master Table (JMAT). This is a very simple table; it has a header entry, which contains system global information (like the number of jobs/sessions, the job/session limit, etc.), and one entry for each job on the system. Each of these entries contains some (but not all) information about the job -- its job name, user, account, group, terminal number (for sessions), the Process Identification Number of its main process, and so on. A :SHOWJOB takes virtually all of the information it prints from the JMAT -- just think of the JMAT as a :SHOWJOB stored in a machine-readable form. Since there's only one JMAT, it occupies a data segment with a fixed data segment number. If you want to know the data segment number or the page of the System Tables Manual on which the JMAT is described, just look at Appendix 2 of this paper.

A job's JMAT entry, however, does not contain all the information about the job. Much of the remaining information -- CPU time, the capabilities of the user running the job, :ALLOW mask, etc. -- is kept in a table called the Job Information Table (JIT). There's one JIT for each session, and it's pointed to by the PXGLOB area (which we mentioned earlier) of the stack of each process that belongs to the job. To get to your JIT, you'd first find its data segment number from your PXGLOB, and then use an MFDS to read it. Just as the :SHOWJOB command reads the JMAT, the WHO intrinsic looks at your JIT; all the information it gives you -- capabilities, user, account, group, home group, local attributes, etc. -- all come from the JIT. The JIT is even easier to work with than the JMAT, being just a big record structure with a lot of fields (each of which is at least briefly mentioned in the System Tables Manual). Note that there's some duplication of data (user name, account name, group name, etc.) between the JMAT and the JIT. HP does some damndest things.

If you've been paying particularly close attention, you'll find that I lied. A job is more than just a collection of processes -- there are also some job-local entities, such as :FILE equations and temporary files, that have to be maintained for each job. These are kept in a Job Directory Table (JDT), of which there is one per session (also pointed to by the PXGLOBs of the processes belonging to that session). This table actually contains a header and five sub-tables: one for the job's temporary files, one for the :FILE equations, one for the JCWs, one for the job-local data segments, and one for the :CLINE (a datacomm command) equations. The header contains the pointer to each of the sub-tables, and each sub-table is in turn is a collection of variable-length entries. Thus, to find a particular :FILE equation, you'd get the JDT's data segment number from your PXGLOB, get the JDT-relative index of the :FILE equation table from the JDT header, and then go through the :FILE equation table entries until you find

one with the name you're looking for. Not the most entertaining job in the world, but not too difficult, either.

Finally, for completeness' sake I must mention two other, relatively unimportant, tables -- the Job Cutoff Table (JCUT) and the Job Process Count Table (JPCNT). The JCUT contains information used for aborting all jobs that have a CPU time limit. I haven't the foggiest notion of what the JPCNT is actually useful for, except for a little-used (if at all) device called Job SIRs. This is one table you can afford to ignore.

Thus, to summarize, the structure of those system tables that pertain to jobs is roughly like the following:

JMAT (a single data segment), which contains for each job
The job number, job name, user, account, group, terminal number, etc.
The Process Identification Number (PIN) of the job's main (CI) process

One PKGLOB per process belonging to the job, which points to
One JIT per job, which contains
The job number, job name, user, account, group, home group, CPU time, :ALLOW mask, etc.
One JDT per job, which contains information on the job's
Temporary files (JTFF)
:FILE equations (JFEQ)
Data segments (JDSD)
JCWs (JJCW)
:CLINE equations (JLEQ)
The JMAT entry referring to the job

JCUT (stand-alone table), which contains
Information on all jobs that have a CPU time limit

JPCNT (stand-alone table), which contains
A bunch of bits the reason for which I could never fathom

PROCESS-ORIENTED SYSTEM TABLES

Rather more important than the concept of a job is the concept of a process.

A process is a single instance of a program running. It has its own data space and code (the code may be shared among several processes that are running the same program). All work done on the system is done on behalf of a process, some of which -- the user processes -- belong to jobs, and others -- the system processes -- do not.

The master table that describes all processes is the Process Control Block table (PCB). It, like the JMAT, contains a header entry and one entry for each process. The process entry contains useful information like a process' priority, the data segment number of its stack, its family relationships (father, son, brother), wait flags (is it waiting

for a SIR, a RIN, etc.), and more. A process' PIN (Process Identification Number) is nothing more than the number of the process' entry in the table. If you have a PIN, you can just multiply it by the PCB entry size, and do an MFDS of the entry from that location in the PCB (which has a fixed data segment number).

Also like the JMAT, the PCB does not contain all the information worth knowing about a particular process. Much very useful information -- what files are open by the process, what its register values are, what traps it has enabled, where its JIT and JDT are, and so on -- is kept in the so-called PCB eXtension (PCBX), which is stored in the DL-negative area of the process' stack. The PCBX consists of

- * The PXGLOB, which is mostly the same for all processes in a job and contains global information like the job's JIT and JDT data segment numbers, the device on which the job is running, the index of the process' JMAT entry, etc.
- * The PXFIXED, which contains a variety of useful data, like the values of the process' DB and S registers (they have to be kept somewhere when the process gets swapped out), the addresses of whatever control-Y and other trap routines are set, the job number of the job the process is running, and so on. Like the PXGLOB, this is just one big record structure.
- * The PXFILE, which describes the files the process has open. Its structure is so complicated that it deserves a separate paper (which may or may not be forthcoming). Fortunately, the System Tables Manual is somewhat more lucid than usual with regard to the structure of the PXFILE (which isn't saying much), and you might be able to figure things out by reading it. But don't bet on it.

This is about all there is to the structure of the process tables. Note the recurring concept of a master table that contains some information on all processes or jobs, and a table for each process or job that contains more detailed information about the particular process or job. You'll see this again when we get to logical devices and the I/O tables.

Finally, two other, less useful, tables. The Process-Job cross REFERENCE table (PJKREF), whose format IS NOT GIVEN IN THE SYSTEM TABLES MANUAL, contains, for every process, its job number. The job number of process PIN is in the PINth word of this table. The Process-Process COMMunication table (PPCOM) contains information on any mail the process might have in its mailbox.

Thus, to summarize, the process table structure looks like:

PCB (a single data segment), which contains for each process
The process' priority, wait flags, etc.
The PINs of the process' father, son, and brother
The data segment number of the process' stack

PCBX (stored in the process' stack's DL-negative area), containing
PXGLOB, which contains

The data segment of the process' JIT and JDT

The index of the process' JMAT entry

Some other information (like the process' terminal)

PXFIXED, which contains

The job number, saved registers, trap addresses, etc.

PXFILE, which contains

Information on all the files used by the process

PPCOM (a single data segment), which contains for each process

Information on any mail message it might have in its mailbox

PJXREF (a single data segment), which contains for each process

Its job number; the job # of process PIN is stored in the

PINth word of the table

DEVICE-ORIENTED SYSTEM TABLES

Many system tables describe devices -- discs, terminals, etc. Of all the parts of MPE, the I/O system is probably the most intricate and most confusing portion.

Devices are referred to by their Logical DEvIce numbers (LDEVs). Although this means there's such a thing as a physical device, us software people don't care about it. In fact, I'll use the terms device, logical device, and LDEV interchangeably (since everybody else does, anyway).

Every device has an entry in the Logical Device Table (LDT) and the Logical-Physical Device Table (LPDT), both of which are single data segments with fixed data segment numbers. The LDT contains the device type, record with, main pin of owning process (for terminals and tapes), etc. Note that the second half of the LDT's data segment contains the so-called LDT eXTension (LDTX), which contains some more useful information. Like the LDT, the LDTX has one entry per LDEV. The start address of the LDTX can be deduced from finding out the number of configuring LDT entries from the LDT header entry and then calculating the index of the first word that isn't used by the LDT. The LPDT contains some other stuff, like the device subtype, state, some more flags, and the address (SYSGLOB-relative!) of the device's Device Information Table (DIT).

The DIT contains various information about the device that is highly specific to the particular device type. The System Tables Manual has 50-odd pages describing the various DIT formats. It is not very entertaining reading, and will never compete with a well-written, say, telephone book.

All device classes in the system are listed, along with the LDEVs they represent, in the Device Class Table (DCT), which contains one variable-length entry per device class. To find out, say, what devices the class TAPE contains, just traverse the table until you find an entry with a name of TAPE. Like the LDT, the DCT shares its data

segment with another (less useful) table called the Terminal Descriptor Table (TDT).

All the I/Os that are currently pending in the system have entries in the I/O Queue (IOQ, for non-disc devices) and Disc Request Queue (DRQ, for disc devices) tables. The last thing I want to do is expend more energy describing these truly complicated tables that most people will never use anyway.

Finally, some more useless tables: the Interrupt Linkage Table (ILT), Driver Linkage Table (DLT), System Buffers (SBUF), Terminal Buffers (TBUF), and the Interrupt Control Stack (ICS). Definitely not your bread-and-butter programming stuff. If you don't know about them, you're none the worse for it.

Finally, a recap of the I/O System tables:

LDT (a single data segment), which contains for each device Device type, record size, main owner pin (for non-discs), etc.

LPDT (a single data segment), which contains for each device Device sub-type, device status, various other flags, etc. The address of the device's DIT

DCT (a single data segment), which contains for each device class The LDEVs that are in the class The type of class (discs, serial I/O devices, etc.)

DIT (a chunk of memory for each LDEV, pointed to by a SYSGL0B-relative address), which contains a lot of device-dependent data

IOQ, DRQ, SBUF, TBUF, DLT, ILT, ICS, LDTX, and TDT, which you'll have to look up in the System Tables Manual if you really want to be a big hit at cocktail parties

FILE-ORIENTED SYSTEM TABLES

The entity that everybody works most with is the file.

Every file in the system has a File LABEL (FLAB) kept in a sector on disc. Although this is not a data segment, it can properly be called a system table, since it contains system-maintained data. The file label is a veritable treasure-trove of information, containing everything you'd ever want to know about a closed file. All modes of :LISTF get their information straight from the file label.

Getting to a file's file label, however, isn't very easy. The disc address of a file label may be stored:

- * for permanent files, in the system directory
- * for job temporary files, in the job's JDT

- * for spool files, in the Output Device Directory (ODD) or Input Device Directory (IDD)
- * for files that were FOPENed as new files and not yet FCLOSEd, in the File Control Block (FCB) belonging to that file

For open files, the matter is somewhat more complicated. In addition to the permanent, relatively unchanging information that is kept in the FLAB for a closed file, the system must also keep track of an open file's current record pointer, current block number, buffers, etc. As I said before, the topic of open file tables is such a complex one that it deserves a paper of its own. All I'll do here is give a brief structural description:

FLAB (stored on disc, one per file) contains The file name, code, size, extent map, etc. -- everything a :LISTF could give you

The system directory contains entries that point to FLABs of permanent files

Each job's JDT contains entries that point to FLABs of job temporary files

The spool file directories IDD and ODD contain entries that point to FLABs of spool files

Each process' Active File Table (AFT), stored in the process' PXFILE area, briefly describes all the files opened by the process (a file number is an index into this table) and points to each file's LACB and PACB

For each open of a file, a Physical Access Control Block (PACB) describes the current state of the file -- the current record number, the buffers used for buffered file access, the number of physical and logical I/Os that have occurred, and so on. For files opened MULTI, or GMULTI, there's only one such PACB for all openers; for all other open files, there's one PACB per opener.

For each open of a file that is being accessed using MULTI or GMULTI access, there's one LACB per opener; for files not opened using MULTI or GMULTI there are no LACBs

For each opened file, there's an FCB, which contains information common to all readers of the file (mostly a rehash of the file label, kept in memory for faster access). The FCB is pointed to both by all the file's PACBs and by the file's file label.

FMAVT (File Multi-Access Vector Table) contains information on all files opened with MULTI or GMULTI access

SEGMENT-ORIENTED SYSTEM TABLES

Segments are broken up into two very distinct categories: data segments and code segments.

Data segments are described in the Data Segment Table (DST); there's one entry for each data segment, and it contains information like the data segment length, memory address, flags, etc. That's all there's to it, a welcome change from the complicated structures in place for files, devices, and even jobs and processes.

No such luck with code segments, though. The Code Segment Table (CST) contains only the entries -- also listing the segments' lengths, addresses, and other flags -- referring to all currently loaded SL segments and the program segments of the process that is currently running.

Say that you want to find out about the code segments of a process given its PIN. What you want to do is to get to the entries of the CST eXtension (CSTX) table -- which is stored in a single data segment -- that correspond to the process' code segments.

Your first step would be to get the "CSTX block map index", known as CSTXEIX, of the process from the process' PCB. Note that this value is listed in the PCB table layout as BLKINX and PBX.

This is an index into a table called the CSTBLK, also known as the CSTXMAP. The CSTXEIXth entry in the CSTBLK contains one thing and one thing only: the address of the first CSTX entry describing the segments of the process.

However, this is NOT a CSTX-relative address. This is not even a SYSGLOB-relative address. An absolute address, you say? No.

This address is relative to the base of the DST, an entirely different data segment altogether! Since you can't very well MFDS CSTX entries from the DST, you have to subtract the contents of SYSGLOB location %33 from the value, and then you have the CSTX-relative index. SYSGLOB %33 (absolute %1033) is the DST-relative address of the first entry of the CSTX.

Once you're done with all this -- you got the CSTXEIX from the PCB, you got the DST-relative address from the CSTBLK, and you converted it to a CSTX-relative-address -- you can now get the CSTX entries. The first entry, the one pointed to by the address you worked so hard to get, tells you how many code segments the process' has and how many users are sharing it. Subsequent entries describe each of the program's segments.

In summary, the segment-oriented tables look like this:

DST, one entry per data segment containing
The length of the data segment
The memory address of the start of the data segment
Miscellaneous flags

CST, one entry per SL code segment and each code segment of the currently-executing program containing

The length of the code segment
The memory address of the start of the code segment
Miscellaneous flags

CSTX, one entry per loaded program (not process! all processes running the same program share code segments) containing
The number of segments in the program
The number of processes running the program

Also in the CSTX, one entry per segment of each loaded program, formatted like a CST entry

CSTBLK (also known as CSTXMAP), one entry per program containing
A DST(!)-relative pointer to the first CSTX entry belonging to the program

MISCELLANEOUS SYSTEM TABLES -- SPOOLING

The information on all spool files in the system is kept in the Input Device Directory (IDD) for input spool files and the Output Device Directory (ODD) for output spool files. Both of these are independent data segments, and the formats of all their entries are identical. XDD is a term used to generically talk about either the IDD or the ODD; an XDD entry is an entry that could be from the IDD or the ODD.

An XDD has three kinds of entries:

- * One header entry, describing the table -- it contains things like the size of the table and the next input/output spool file number to be allocated.
- * One device header entry for each device in the system. Each entry contains the device number, device outfence (if any), and the head and tail addresses of the linked list of spool file entries belonging to that device. The 0th device header entry contains the head and tail addresses of the linked list of all spool file entries belonging not to a device, but to a device class.
- * One spool file entry for each existing spool file. Each entry contains the name of the spool file, the user, account, and job name that it belongs to, the disc address of its file label, etc. SPOOK's >SHOW command (especially >SHOW;@) essentially print the contents of these entries.

This is a comparatively easy table to navigate. MPEX's spool-file manipulation routines -- %PURGE:SPOOL, %LISTF:SPOOL, etc. -- essentially traverse the ODD, picking up spool file file labels as they go along. Note that the addresses in the linked list are table-relative addresses of the start of the next spool file entry; 0 indicates no next entry.

MISCELLANEOUS SYSTEM TABLES -- OTHERS

There are several other relatively useful system tables.

The LST (Loader Segment Table) contains information on all loaded program files and SL files. MPEX's %LISTF,ACCESS goes through this table to find out who may have a particular file loaded.

The SIR (System Internal Resource) table contains information on all SIRs that are currently locked. If you see the system hanging up and you're afraid that it's because someone isn't releasing a SIR or you're getting a SIR deadlock, you can go into privileged mode :DEBUG and have a look at this table.

The RIN (Resource Identification Number) table contains information on all RINs that are currently locked. MPEX's %LISTF,ACCESS looks at this to find out who is locking a file or waiting to lock the file.

The RIT (Reply Information Table) contains information on all outstanding replies.

There are other tables, having to do with logging, private volumes, message files, :WELCOME messages, you name it. I won't talk further about them simply because of space limitations. Also, most of the stand-alone tables are relatively simply structured, and you can find out how to get to them by reading the System Tables Manual and conducting some experiments with privileged mode :DEBUG.

Remember that there is a brief description of every (well, almost every) table in the system and a "pointer" to its entry in the System Tables Manual in Appendix 2 of this paper.

CONCLUSION

System tables are not the incomprehensible monsters that they may seem to be at first glance. Once you master the various techniques described in this paper and get a reasonable familiarity with the actual contents of the tables, you should, with relatively little effort and worry, be able to access system tables just as easily as you would, say, a file or a database.

Good luck and HAPPY HACKING.

ACKNOWLEDGEMENTS

I would like to thank the following people for reviewing and making many useful comments on this paper:

Steve Cooper, Allegro Consultants
Robert Green, Robelle Consulting Ltd.
David Greer, Robelle Consulting Ltd.
Jack Howard, Consultant, Los Angeles
Stan Sieler, Allegro Consultants

Jim Squires, HP Fullerton
Vladimir Volokh, VESOFT, Inc.

and of course everybody -- too numerous to mention -- who taught me all this stuff. May your programs never fail and your systems never crash.

APPENDIX 1 -- PROGRAM 1

```
<< Don't forget to :PREP me with ;CAP=PM! >>
<< Works on MPE IV and MPE V. >>
$CONTROL MAIN=PROGRAM'1, NOSOURCE, USLIMIT
<< This program:
    Prints out word 0 of the user's capability matrix;
    Prints out the job or session's job/session number;
    Gives the user SM capability and does a LISTUSER
    before (which should fail) and after.
>>
BEGIN
INTRINSIC
    ASCII,
    COMMAND,
    GETPRIVMODE,
    GETUSERMODE,
    PRINT;
INTEGER POINTER
    DL;
INTEGER
    CAPS,
    DUMMY,
    ERROR,
    JS'NUMBER,
    LENGTH;
ARRAY
    BUFFER'L(0:127);
BYTE ARRAY
    BUFFER'(*)=BUFFER'L;
DEFINE
    INDEX'PXGLOB = DL(-1) #,
    INDEX'PXFIXED = DL(-2) #;

<< Get the DL pointer >>
PUSH (DL);
@DL:=TOS;

<< Get and print word 0 of the capability matrix >>
GETPRIVMODE;
CAPS:=DL(-INDEX'PXGLOB+2);
GETUSERMODE;
MOVE BUFFER':="Capability word 0 = %";
PRINT (BUFFER'L, -21, %320);
```

```
ASCII (CAPS, 8, BUFFER');
PRINT (BUFFER'L, -6, 0);

<< Get and print the session number >>
GETPRIVMODE;
JS'NUMBER:=DL(-INDEX'PXFIXED+19);
GETUSERMODE;
IF JS'NUMBER.(0:2)=1 THEN
  << JS'NUMBER.(0:2) indicates session (1) or job (2) >>
  MOVE BUFFER:="#S"
ELSE
  MOVE BUFFER:="#J";
PRINT (BUFFER'L, -2, %320);
LENGTH:=ASCII (JS'NUMBER.(2:14), 10, BUFFER');
PRINT (BUFFER'L, -LENGTH, 0);

<< First, show that LISTUSER MANAGER.SYS fails without SM >>
MOVE BUFFER:="Trying LISTUSER MANAGER.SYS before getting SM";
PRINT (BUFFER'L, -45, 0);
MOVE BUFFER:=("LISTUSER MANAGER.SYS", %15);
COMMAND (BUFFER', ERROR, DUMMY);
IF ERROR<>0 THEN
  BEGIN
    MOVE BUFFER:="CI Error # ";
    PRINT (BUFFER'L, -11, %320);
    LENGTH:=ASCII (ERROR, 10, BUFFER');
    PRINT (BUFFER'L, -LENGTH, 0);
  END;

<< Now, get SM capability >>
GETPRIVMODE;
DL(-INDEX'PXGLOB+2).(0:1) << SM bit >> := 1;
GETUSERMODE;

<< Now, show that LISTUSER MANAGER.SYS succeeds >>
MOVE BUFFER:="Trying LISTUSER MANAGER.SYS after getting SM";
PRINT (BUFFER'L, -44, 0);
MOVE BUFFER:=("LISTUSER MANAGER.SYS", %15);
COMMAND (BUFFER', ERROR, DUMMY);
IF ERROR<>0 THEN
  BEGIN
    MOVE BUFFER:="CI Error # ";
    PRINT (BUFFER'L, -11, %320);
    LENGTH:=ASCII (ERROR, 10, BUFFER');
    PRINT (BUFFER'L, -LENGTH, 0);
  END;
END.
```

APPENDIX 1 -- PROGRAM 2

```
<< Don't forget to :PREP me with ;CAP=PM! >>
<< Works on MPE IV; changes needed for MPE V are indicated. >>
<< This program assumes the existence of the DSEGREAD and MYSTACK
  procedures described in the text. >>
$CONTROL MAIN=PROGRAM'2, NOSOURCE, USLIMIT
BEGIN
<< This program:
  Prints out the job/session name of the current job/session
  (or blanks if no job/session name.)
>>
INTRINSIC
  PRINT;

EQUATE
  SIZE'PXGLOB = 8; << 12 for MPE V >>
DEFINE
  PXGLOB'JIT'DSEG = PXGLOB(6).(6:10) #; << JIT data segment # >>
  << PXGLOB(11) for MPE V >>
ARRAY
  PXGLOB(0:SIZE'PXGLOB-1); << Array for holding the PXGLOB data >>
EQUATE
  OFFSET'PXGLOB = 0; << Offset of the PXGLOB in the stack dseg >>

EQUATE
  SIZE'JIT = 61; << 67 for MPE V >>
DEFINE
  JIT'JS'NAME = JIT(44) #; << MPE IV or MPE V >>
ARRAY
  JIT(0:SIZE'JIT-1); << Array for holding the JIT data >>

DSEGREAD (MYSTACK, OFFSET'PXGLOB, PXGLOB, SIZE'PXGLOB);
DSEGREAD (PXGLOB'JIT'DSEG, 0, JIT, SIZE'JIT);
PRINT (JIT'JS'NAME, -8, 0);
END.
```

APPENDIX 2 -- SUMMARY OF SYSTEM TABLES

System tables are listed alphabetically by their initials. The chapter numbers of the tables in the System Tables Manual are given. Page numbers are not given because they may vary from release to release of each manual -- they shouldn't be too hard to find in any case.

All tables are stored in memory (not on disc) unless otherwise specified.

Table	Chapter	Table describes or contains
AFT	6 [3]	One/process; part of PXFILE
ASSOC	15	All :ASSOCIATE commands
BKPNT	17	:DEBUG breakpoints
CBT	6 [3]	Contains file system control blocks
CILOG	None	:(CMD) USER.ACCT logons
CST	2 *	Loaded SL and current program's code segments
CSTAB	23 [2]	Contains communications system info
CSTBLK	2 *	Contains pointers into the CSTX
CSTX	2	All code segments in the system
DCT	13	Device classes
DIT	13 [3]	One/device; contains device-specific info
DLT	13	Device drivers
DRQ	13 *	Queued disc I/O request
DST	2 *	Data segments
FCB	6 [3]	One/file opener; file system info
FLAB	6 [3]	One/file, stored on disc; file parameters
FMAVT	6	Files opened with MULTI/GMULTI access
ICS	13	Contains stack of interrupt-handling process
IDD	14	Input spool files
ILT	13	Contains interrupt info
IOQ	13 *	Queued I/O requests
JCUT	8 *	Jobs that have a CPU limit
JDS	8 [3]	One/job; job data segment info, part of JDT
JDT	8 [3]	One/job; job's :FILE eqns, temp files, etc.
JFEQ	8 [3]	One/job; job :FILE equation info, part of JDT
JIT	8 [3]	One/job; detailed job info
JJCW	8 [3]	One/job; job JCW info, part of JDT
JLEQ	8 [3]	One/job; job :CLINE equation info, part of JDT
JMAT	8	Jobs
JPCNT	8 *	Contains arcane info about jobs
JJFD	8 [3]	One/job; job temp file info, part of JDT
LACB	6 [3]	One/MULTI or GMULTI file opener; file system info
LDT	13	LDEVs
LDTX	13	LDEVs
LIDTAB	17	Logging identifiers
LOGBUFF	17	Contains logging buffers
LOGTAB	17	Logging
LPDT	13 *	LDEVs; points to DITs
LST	11	All loaded program and SL files
MEASINFO	17 *	Measurement info

MVTAB	12	Mounted private volumes
ODD	14	Output spool files
PACB	6 [3]	One/file opener; file system info
PCB	7 *	Processes
PCBX	7 [3]	One/process; contains PXGLOB, PXFIXED, PXFILE
PJXREF	None	Maps PINs into job numbers
PPCOM	7	MAIL messages that have been sent but not gotten
PVUSER	12	Users of private volumes
PXFILE	6 [3]	One/process; contains info on process' files
PXFIXED	7 [3]	One/process; contains miscellaneous process info
PXGLOB	7 [3]	One/process; points to JIT and JDT
RIN	5	RINs (including file locks)
RIT	15	Outstanding :REPLYs
SBUF	13 *	Contains buffers used for some system I/O's
SIR	5 *	Locked SIRs
SRT	2 *	Special memory management requests
SWAPTAB	2 *	Contains segment swapping info
SYSGLOB	1	Contains miscellaneous system info
SYSJDT	8	Contains the JDT used by system processes
SYSJIT	8	Contains the JIT used by system processes
TBUF	13 *	Contains buffers used for terminal I/O's
TDT	13 [1]	Special configurations for terminals
TRL	17 *	Requests for system timer (PAUSEs, timeouts, etc.)
UCRQ	8	All requests to the process controller process
VDD	17	Mounted labelled tapes
VDSMTAB	3 *	Virtual memory on discs
VTAB	3	Discs
XDD	14	Generic name for either the IDD or ODD

* - Indicates a system table that is accessible using the SPL "INTEGER POINTER table = number;" construct. For the system table number (not necessarily equal to the data segment number), see the layout of SYSGLOB in chapter 1. If SYSGLOB cell N is indicated as containing the base of a table, then N is the table's number.

[1] - Indicates a table that exists in MPE V but not in MPE IV.

[2] - Indicates a table documented in the MPE V manual but not in the MPE IV manual.

[3] - Indicates that there is more than one of these tables, one for each one of a number of objects. For instance, a JIT, of which there's one per job, and an FLAB, of which there's one per file. All tables not so marked are unique.

3016. The poor man's DS, fact or fiction.

Rene van Geesbergen,
Holland House, AALST, Holland.

Abstract.

Through the years, the only way of connecting two HP3000 systems in a reasonable feasible way, was the well-known DSN/DS Software in combination with additional hardware (INP, SSLC). With the introduction of the cheaper systems (Mighty Mouse) this proves to be a rather expensive method.

For those, who do not require/demand high transmission speeds, a cheaper method could be envisaged. By connecting two HP3000 terminal ports, two systems can communicate with each other at a speed of 9600 or even 19200 bps.

This paper will elaborate on the software and hardware requirements to support this type of communication.

Items to be discussed will be:

- * Opening a Line;
- * Communication Internals;
- * Remote Logon and Remote Commands;
- * Network File Transfer;
- * Program-to-program Communication;
- * Remote Database Access;
- * Peripheral Sharing;
- * Remote File Access;

Contents.

1. Introduction.
2. Design considerations.
3. The easy part.
4. The not-so-easy part.
5. The difficult part.
6. Conclusions.

1. Introduction.

One of the big advantages of Hewlett Packard computers has always been that they offer various communication methods like RJE, MRJE, IMF, DS and more recently NRJE. The most widely used method is DSN/DS that enables the user to communicate from an HP3000 with another HP3000 or an HP1000 in a very sophisticated way. The features supported by DS are :

- Remote command execution
- Network File transfer
- Program-to-program communication
- Remote database access
- Peripheral sharing
- Remote file access

The combination of these features makes DSN/DS to a very powerful product. However, there is one big disadvantage : it is rather expensive. To make this communication possible two pieces of hardware (f.i. INP's) and two copies of the DSN/DS software are needed. Total costs approx. \$ 15.000,-. This will not be a big problem when connecting two series 68 computers, it gets out of proportion when connecting two or more series 37 computers.

This paper will elaborate on the possibility to access a remote HP3000 on a DS feature level, using an asynchronous link, i.c. using two ATC, ADCC or ATP ports connected with a \$ 50,- cable, or connected over a public or leased telephone line using datacomm equipment.

In this paper, it is assumed that the reader is somewhat familiar with datacomm and the DSN/DS software. Only the principle of a poor man's DS is discussed here. A detailed study reveals a lot of additional problems not mentioned in this paper. Also the programming techniques needed to make a software package like PMDS are not discussed here. This would make the problem less understandable by non-technical readers.

The following chapter will discuss a number of design considerations with regard to a Poor Man's Distributed Systems connection or PMDS as it will be called in this paper.

Chapter 3, 4 and 5 will discuss the various DS features to be incorporated in PMDS with an increasing level of difficulty.

2. Design considerations.

Principle of operation

The first thing to do is look at DS, and see what can be learned from that. Figure 1 shows a schematic representation of the DS principle.

There is one DS-subsystem process (DS monitor) for each communication link. A DS monitor (DSMON) communicates with an colleague DSMON on the other system. A user process that wants to use that link communicates

with the local DSMON, and the DSMON takes care of the actual transport. On the remote machine communication is established between the remote DSMON and a remote user process. The DS processes are transparent to the user, but they are there. In order to give a user remote interactive capabilities, a virtual terminal is assigned to the local user.

Figure 2 shows the ideal situation for PMDS, because now the expensive INP connection has been changed into a cheap asynchronous port connection. However, there is one big problem : How can virtual terminals be assigned to the local users? Virtual terminals are internal to MPE, they must be configured in the I/O configuration, they must have a special driver etc. MPE looks at the asynchronous port as a physical terminal, and it will be very difficult to change that. It requires changes in the MPE internals, Privileged Mode, with the risk of brand new system failures and other kinds of nasty things. This might be the ideal operation, but it is not an ideal situation.

Figure 3 shows the simplified version of the previous one. The line is now exclusively in use by one local user. Now one 'virtual' terminal exists on the remote system, but that remote system thinks it is a physical terminal. Obviously the disadvantage of this method is that the line cannot be used concurrently by more than one user. However, for the time being this method of operation will be used by PMDS.

Protocol

The PMDS software has to take care that all data is transmitted and received correctly, and that no data is lost. Examples of what can go wrong are :

- Local system is writing but remote system is not reading
- Local system does not read while remote system is writing
- Both systems are writing
- Both systems are reading
- Etc.

The local PMDS software is almost unaware of the activities performed on the remote system, it does not know if it is talking to MPE or TDP, whether it must expect little or much data, etc. Since the output generation of the remote system must be controlled, a handshake between the two systems must be used. The major two handshake methods used on the HP3000 are xon/xoff and enq/ack.

In the terminal-to-computer communication, the xon/xoff protocol is controlled by the terminal. As the terminal transmits an xoff (dc3) to the computer, the computer suspends sending data, as the terminal sends an xon (dc1) the computer resumes sending data. The enq/ack protocol is controlled by the computer. After a block of text, the computer sends an enq and stops transmitting. When the terminal is ready to receive the next block, it sends an ack to the computer.

In the PMDS situation, the enq/ack protocol can be very useful when the remote system must be controlled when generating output. The xon/xoff

protocol has no meaning in the PMDS situation, since the local PMDS software cannot send an xoff while reading. It effectively means no handshake protocol at all. This can only be very useful when the local PMDS software knows in advance how many data to expect during a read.

A problem will arise when the local PMDS software has to communicate interactively with the remote system. When a command is sent to the remote system, that remote system will not necessarily wait until the local software is ready to receive data. It may well be possible that it starts sending data immediately. In this case data may get lost, especially when using high transmission speeds and when the workload on the local system is big.

A possible method to prevent this is by using NOWAIT I/O. The only drawback of using NOWAIT I/O is that the program file suddenly needs Privileged Mode capability. The use of NOWAIT I/O makes it possible for the local PMDS software to issue a read before sending the command to the remote system. (Using Privileged Mode for NOWAIT I/O is totally harmless, Nobody ever understood why HP requires that the user must have PM capability for using NOWAIT I/O instead of some other (NW?) capability). Experimentation will probably be necessary to solve the protocol problem completely.

Data integrity

Especially when two systems are connected via modems, line distortions may occur, corrupting the data being transmitted. It is important that PMDS checks for data validity as much as possible. This will be a problem when the process that sends the data is a non-PMDS process, because that process just sends the data, and that's all. Even if the PMDS software on the remote side detects a corruption, there is no way to request a retransmission.

The situation changes when two PMDS processes communicate with each other, because then full data integrity can be ensured. A checksum can be calculated for each record transmitted, and checked again on the other system. If necessary a retransmission can be requested, the number of retries can be counted, etc.

It is therefore mandatory, that all communication, except for remote command execution, is handled by two PMDS processes, so that a high degree of data integrity can be achieved.

User interface/MPE compatibility

It would be nice if PMDS would look like DS. This would minimize the changes in UDC's, job streams etc. and it will require hardly any special training for the users and application programmers.

In order to achieve this, PMDS must initially look like MPE, it must display a semicolon as a prompt, it must accept and execute local MPE

commands, it must intercept DSLINE and REMOTE commands, file equations, etc. A very nice feature would be if PMDS could handle UDC's.

It would also be nice if PMDS would include a configuration table that links a remote system name to one or more logical device numbers of the ports connected to that remote system. In that case it will be possible for the user to access a remote system using the remote system name, and PMDS will find out which of the configured ports is available, if any.

PMDS intrinsics, used for program-to-program communication must have the same names, parameters and functions as their DS-colleagues, f.i. POPEN, PWRITE, GET, ACCEPT etc. They must reside in a segmented library that can be linked to a program using the LIB= option in the :RUN command. This means that existing programs using PTOP will run under PMDS without modification or recompilation. The same goes for IMAGE intrinsics, used for remote database access.

3. The easy part.

This chapter will discuss a number of features, to be incorporated in PMDS. A number of ideas, thoughts and solutions will be given on the subjects Remote Logon, Remote Command Execution and Network File Transfer.

Remark : the title of this chapter does not imply that it will be easy to build a program as described below, it merely means that it can be done without changes in MPE software and/or HP3000 hardware.

Remote Logon

The first part of PMDS is very simple. A user just should enter the MPE command :RUN PMDS. Of course this can be done using a (logon) UDC. Nothing spectacular happens. PMDS opens the file COMMAND.PUB.SYS and checks if UDC's exist for this user. If so it opens the UDC files and displays a semicolon, thus pretending it is the MPE command interpreter.

The user can enter commands like SHOWJOB, RUN, or L (a UDC command meaning LISTF). PMDS must pass the commands to the COMMAND intrinsic, or start programs, i.c. it must do everything as if it were the MPE command interpreter.

Now the user wants to start a remote session, so he enters the DSLINE command. PMDS recognizes the DSLINE command and gets the dsdevice name of the remote system that should be part of the command entered. Using the PMDS configuration file it checks the dsdevice and gets the logical device number (or numbers) of the ports connected with the remote system. Now it can check if that port (or one of these ports) is available, and open it. Note that this is the moment for PMDS to enter Privileged Mode, because the port has to be opened with the NOWAIT I/O option. After a successful open of the port, User Mode is restored. To

make everything look real, PMDS can now display a message like 'PMDS LINE NUMBER = #Ln.', where n is a number identifying the remote system.

The next step for the user is to enter the 'REMOTE HELLO user.acct' command. PMDS recognizes the REMOTE HELLO command, issues a (nowait) read, sends a carriage return (cr) to the remote system, and waits for the read to complete. If everything is OK, the remote system will respond with something like cr/lf/:/dcl. If the response is garbage or nothing at all, a second, third, etc. attempt can be made to get this first and important response from the remote system. After a sufficient number of attempts, PMDS should stop trying and report to the user that the remote logon failed due to a non-responding remote system. Suppose that the proper sequence of characters was received, PMDS strips the word 'REMOTE' from, and adds a 'TERM=10' parameter to the original command string, issues a second (nowait) read, transmits the command string, and waits for the read to complete. It is not evident what will be received now, it could be f.i. 'CAN'T INITIATE NEW SESSION NOW', 'ENTER USER PASSWORD' or even 'HP3000 / MPE V etc.' But this sequence of actions, i.c. issuing a read, transmit data and wait for the read to complete can be repeated until we have a remote session established. Until now PMDS has done nothing more than emulating a terminal.

A few rules have to be observed when emulating a terminal.

- Always issue a nowait read before transmitting data to the remote system. Then complete the read.
- Configure both ports initially as terminal type 13.
- Transmit an ack to the port when receiving an enq.
- Start reading from the terminal keyboard when receiving a dcl.

Remote Command Execution

Now the user is successfully logged on to the remote system. The last thing received from that remote system was the string :/dcl, so PMDS knows that it must read input from the terminal, it will display a semicolon and wait for user input. It keeps on behaving like MPE, so the SHOWJOB command generates showjob output and EDITOR still runs the program EDITOR.PUB.SYS. But as soon as PMDS detects that a command starts with the word REMOTE, it will take special action. It will strip off the word REMOTE, and send the command to the remote system. It then will keep on receiving and displaying data until a new :/dcl is received, and that is the sign for a new user input.

Network File Transfer (NFT)

It is well known, that users will never be satisfied with the features they already have, and the PMDS user is no exception to that rule. Now he wants to copy a file from the local to the remote system. In order to remain compatible with MPE, network file transfer is (of course) initiated by the DSCOPY command. As soon as PMDS detects this command, it will start a remote PMDS process by sending a 'RUN PMDS,SLAVE' command to the remote machine. Note that the remote version of PMDS uses

a different entrypoint, so the remote process knows that it is the slave of a master process on the local machine. As soon as the two processes have shaken hands, and have identified themselves to each other, the master PMDS parses the DSCOPY command, opens the local file, and retrieves all information on that file. The master then sends a request to the slave to build that file on the remote system.

Two problems arise here, the first problem is that the master has to ensure that the slave has received the data correctly. A line distortion during the transmission could cause the file information to be garbled, or worse: slightly changed. The usual method to prevent this is by adding block check characters to the data. The master has to perform a smart calculation on the data, and add the result to the data. When the data is received by the slave, it performs the same calculation, and checks if the result is the same as the result sent by the master. If not, the slave must request a retransmission, and the master must send the information again (and again, and again...).

The second problem is that the file information from the file label is binary data, although it will have an ASCII representation. The 'recordlength' field is one 16 bit word, but is also 2 ASCII bytes, and those two bytes can have any value, f.i. enq, dcl, xoff, etc. It may well be that the PMDS software or the terminal driver takes undesired actions depending on the 'random' characters in the data transmitted to the remote system. The same goes for 'intelligent' datacomm equipment used for the physical connection. The master PMDS software has to make sure that only 'harmless' ASCII information is sent to the remote system. It does so by checking each character of the data. If that character is a 'dangerous' character, PMDS will change it to an 'innocent' character, and mark it for the slave by placing a 'special' character in front of it. Note that the 'special' character is also a 'dangerous' character, so if that character is part of the original data, it has to be replaced as well. When the master has finished the 'editing' of the data, it can be transmitted to the slave. Then the slave must scan the data, looking for 'special' characters, and reverse the modifications made by the master.

Now that the slave has received, checked and transformed the file label information, it can build the file on the remote system, and report success or failure to the master. If the build was successful, the master can start transmitting the actual file data to the slave. The same procedures for data checking as described above must be executed for the actual file data as well, to ensure integrity of the file contents.

While playing around with the data to be transmitted, it can be considered to try our hand at some data compression. This can give tremendous speed increase or tremendous overhead, and everything in between. It could prove wise to add some code to the compression module that tries to determine if it is worthwhile to spend valuable CPU time on compression. This is dependent on all sorts of factors like CPU load on the system, transmission speed, contents of the data being transmitted, etc. The compression module could f.i. try to compress the first 50 records of the file, and estimate if there was any benefit.

If so it could decide to transmit the entire file compressed, and if not to transmit the entire file uncompressed. Of course there is a risk that the wrong decision is made, but experiments have shown that the conclusion is usually correct.

When the master encounters the end-of-file, it must send a special control sequence to the slave, so the slave knows that the entire file has been transmitted. The slave then can close the file and report success or failure to the master. It would be nice if the master would have the possibility to request another file to be copied, just like DSCOPY, and repeat the actions described above, but eventually there will be no more files to copy. The master will then kill the slave on the remote system, display a semicolon and wait for user input.

The method described above is the same when the user wants to transfer a file from the remote system to the local system. The local PMDS then behaves like the slave, and the remote PMDS behaves like the master. The local PMDS must send a special control sequence to the remote, meaning 'until further notice you will be the master'

One additional remark on protocol: During the file transfer, both PMDS processes have a rough idea on the amount of data that will arrive during a read. It will make the NFT code easier if the protocol is changed to xon/xoff during file transfer. This will avoid a record transmitted being spliced in two or more blocks with enq's, and it will speed up transmission, when intelligent datacomm equipment is being used.

4. The not-so-easy part.

The subjects discussed in this chapter fall in the category 'not-so-easy' for the following reasons :

- A relatively new feature in the MPE operating system, i.e. IPC (message files), must be used to accomplish the tasks discussed in this chapter.
- These tasks are normally performed by system intrinsics, that reside in the system SL. In the PMDS situation they are performed by PMDS intrinsics, linked to the program that uses them. This means that part of the MPE operating system is temporarily replaced by other software.
- Using the features described below can involve a maximum of four processes communicating with each other.

Still it looks feasible to include the features Program-to-Program Communication and Remote Database Access in a basic PMDS system.

Program-to-Program communication (PTOP)

PTOP is initiated by a user program on the local system. That user program calls special (PM)DS intrinsics, and these intrinsics take care

of the actual communication. The most important intrinsics are summarized below :

Master intrinsics :

- POPEN, creates a slave user process on the remote system.
- PREAD, requests information from the remote system.
- PWRITE, sends information to the remote system.
- PCONTROL, exchanges control information with the remote system.
- PCLOSE, terminates the slave user process on the remote system.

Slave intrinsics :

- GET, receives PTOPI request from the local system.
- ACCEPT, accepts and completes the local systems PTOPI request.
- REJECT, rejects the local systems PTOPI request.

As mentioned earlier, PMDS should use the same names for its PTOPI intrinsics as DS does. In that case any PTOPI program can be used both with DS and PMDS. When the program wishes to use the PMDS intrinsics, it should be run with the PMDS library linked to it.

A schematic diagram of the PMDS program-to-program communication operation is illustrated below.

In order to prevent confusion, resulting from the use of the terms master and slave, the following notations will be used to distinguish the different processes involved.

- PMDS-master is the master PMDS program, running on the local system.
- PMDS-slave is the slave PMDS program, running on the remote system.
- USER-master is the master user program, running on the local system.
- USER-slave is the slave user program, running on the remote system.

The USER-master process is started as a son of the PMDS-master process with the :RUN command. The first interesting action, performed by USER-master is that it will start USER-slave on the remote system to communicate with. It does so by calling the POPEN intrinsic. The POPEN intrinsic must get in touch with PMDS-master, and send it information such as the name of USER-slave, entrypoint, stack size, etc. This information must be sent to PMDS-slave, and then PMDS-slave can create USER-slave, with the PMDS library linked to it. In the meantime PMDS-master builds and opens two message files in order to facilitate the exchange of information between PMDS-master and USER-master via the PMDS intrinsics. Two message files are needed to support both send and receive operations. The same is done by PMDS-slave on the remote system.

Now USER-slave is running, and executes the GET and ACCEPT (or REJECT) intrinsics to confirm its existence to USER-master. These intrinsics pass the confirmation to PMDS-slave, PMDS-slave transmits the information to PMDS-master, and PMDS-master sends it to USER-master. When USER-master (i.c. the POPEN intrinsic) receives the information, it will continue running and executing other PMDS intrinsics.

The same principle goes for the other PTOP intrinsics. F.i. USER-master executes the PWRITE intrinsic. The intrinsic sends information (via IPC) to PMDS-master. PMDS-master sends the information (via ATC/ADCC/ATP) to PMDS-slave, and PMDS slave sends it (via IPC) to USER-slave, who receives the information via the GET intrinsic. Then the ACCEPT intrinsic is executed by PMDS-slave, it sends information to PMDS-slave, to PMDS-master and finally to USER-master, who receives the information via the PWRITE intrinsic that was called previously.

To summarize the PTOP principle :

- A master intrinsic sends information to PMDS-master, and waits for response from PMDS-master.
- Information received by PMDS-master is sent to PMDS-slave, then PMDS-master waits for response from PMDS-slave.
- PMDS-slave sends the information to USER-slave (GET intrinsic) and starts waiting for response from USER-slave.
- The GET intrinsic starts waiting for response from PMDS-slave. When it has received information USER-slave continues running.
- The ACCEPT and REJECT intrinsics send information to PMDS-slave.
- PMDS-slave receives the information, and sends it to PMDS-master, it then starts waiting for response from PMDS-master.
- PMDS-master sends the information to USER-master, it then starts waiting for response from USER-master. Which closes the circle.

Remote Database Access

Surprisingly the principle of remote database access is similar to PTOP. IMAGE databases are accessed using intrinsics, f.i. DBOPEN, DBGET, DBPUT and DBCLOSE. As with PTOP it is possible to define PMDS 'database' intrinsics that can be linked to the user process that wishes to use them.

Figure 5 illustrates the principle of remote database access. The user process wishes to access a database on the remote system, and calls the DBOPEN intrinsic. The DBOPEN intrinsic sends the request to open the database to PMDS-master (via IPC), PMDS-master sends the request to PMDS-slave and PMDS-slave executes the actual DBOPEN intrinsic. It must return the condition code to PMDS-master, who returns it to the user process. This is the generalized method of operation used for all database intrinsics.

There is one pitfall however. Suppose that the user process wishes to use two databases simultaneously, one on the local, and one on the remote system. This means that all PMDS database intrinsics should be

capable of recognizing whether a database is located on the local or on the remote system.

There are two methods to access a remote database. The first method is to set a file equation for the database, pointing to the remote system, and the second method is using a database access file. In both cases, a true DBOPEN call will fail, since IMAGE will report that the remote system was not found (IMAGE does not know of PMDS remote systems). This will be the indication that the database resides on the remote system. If the true DBOPEN succeeds, or results in a failure, other than a communication failure, then PMDS may assume that the database resides on the local system. The PMDS DBOPEN should mark the databases as local or remote, so other database intrinsics know if they must pass the information to PMDS-master, or to the actual IMAGE intrinsic.

A very nice feature of IMAGE is the database access file. This file makes it possible for the user to access a remote database as if it were on the local system. At the time the database is opened, there is no remote session running. The database access file contains information about the remote system, the remote logon id and the name of the remote database. The IMAGE DBOPEN is called using the name of the database access file, IMAGE discovers that the database resides on a remote system, and issues a remote logon automatically. It would be very nice if PMDS supported the same way of accessing a remote database. This will not be a big problem, except for the fact that the database access file is a Privileged file. This means that the database access file must be opened in Privileged Mode (sorry about that), the information retrieved, a remote session started and the remote database opened in one PMDS DBOPEN call. Technically the problem can be solved, however, Privileged Mode is needed once more. But, like in the case of NOWAIT I/O, this is safe usage of PM, since the database access file will be opened with read access only.

5. The difficult part.

This chapter discusses the two most difficult subjects in the PMDS situation. The biggest problem is that both subjects have to do with the MPE file system, and the fact that there are many ways for a program to access the file system. Just look at how f.i. SPL, FORTRAN and COBOL access a file. Almost every language has another interface to the file system. So it will not be simple to write PMDS substitution routines that replace that part of MPE, just like PTOP and Remote Database Access did.

Peripheral sharing

The next feature the PMDS user wants to have is the possibility to access a peripheral device on the remote system.

Figure 6 shows the principle of peripheral sharing. The user again has a semicolon and a blinking cursor on his screen. Now he enters a file equation like :FILE OUT;DEV=dsdevice#device;CCTL. The local PMDS

software must detect that a file equation was entered for a remote device. It must NOT send this command to the COMMAND intrinsic, but it must strip off the 'dsdevice#' part of the command and send it to a slave PMDS process on the remote system. Then the slave must execute the command, so the file equation for the actual device now exists on the remote system. Now the master will create a message file, change the original file equation to :FILE OUT=msgfile;CCTL, and execute it. That completes the preparations for accessing the remote device.

The next step is that the user wants to write information to the remote device, so he starts his own program by issuing a command like :RUN MYPROG. This is a local command, so PMDS will start MYPROG as a son process. However, there is one difference compared with earlier situations : PMDS will not suspend itself, it will start looking if something happens with the message file. It is not important what actions are performed by MYPROG, but one thing will surely happen : MYPROG opens the file OUT, and writes data to it. The master discovers that data is being written in the message file, and sends a request to the slave that the remote file OUT has to be opened. Once the slave has done that, and has reported success to the master, the master can start reading data from the message file, send it to the slave, and the slave can write the data to the device. This procedure goes on until the master encounters an end-of-file condition on the message file, indicating that MYPROG has closed the file OUT. The master then can tell the slave that he can close the file OUT also.

That is the principle of peripheral sharing. However it is not as simple as it looks. The master has to know that the user wants to access a remote peripheral device. This will not be a problem when the file equation is entered directly in the master PMDS program, but suppose that the file equation is executed by MYPROG using the COMMAND intrinsic. Then the master process knows nothing about the plans of MYPROG to write to a remote peripheral. An advanced version of PMDS could support a COMMAND intrinsic, linked to MYPROG, that looks out for remote file equations, and passes all the other commands to the actual COMMAND intrinsic. That would be an improvement. But now suppose that MYPROG uses the FOPEN intrinsic to access the peripheral directly through the DEV parameter. In this case we have the same problem again, unless we also substitute the FOPEN intrinsic. But now suppose, etc. Things tend to get more complicated now, but the general conclusion is that peripheral sharing is only possible when the master knows in advance that a user process wishes to access a remote peripheral, and that the master knows the name of that peripheral.

The second problem to be encountered is that the communication between the master process and the user process is established through a message file. This has great advantages, as described in other Conference Proceedings, but it also introduces an additional problem. A standard disc file can be opened by a process with read/write access. A message file, however, must be always opened with read OR write access, not both. This implies that we can access the remote peripheral either with read or with write access. So full access to all remote printers can be achieved, but only limited access to f.i. remote tape units. Read or write access will work fine, but more complicated actions that read and

write to the tape will not work. In that case the user process will fail due to an invalid access to the message file.

The method described above will also work when a remote user process wants to access a local peripheral. To make things more complicated, imagine the following situations :

- A program wants access to a remote peripheral more than once.
- A program wants access to more than one remote peripheral simultaneously.
- A program accesses a remote tape unit, the reply request will be issued by the remote slave on the remote system, while the local user process already writes to the message file, thinking that it is the tape unit.
- A process, accessing a remote tape unit, checks the device characteristics of the file opened, and fails when it discovers that it has actually opened a message file.
- etc.

Surely more of these potential pitfalls will come to mind while reading the above, however this paper will not present solutions for all these problems. In some cases there will be no solution at all, so this is the first time that PMDS will lose full compatibility with DS. However, the most frequently used form of peripheral sharing, i.e. remote printing, can be covered by PMDS.

Remote File Access (RFA)

The discussion on Peripheral Sharing is also valid for Remote File Access.

The principle of RFA, shown in figure 7 is similar to the principle of peripheral sharing, shown in figure 6. The problem of read only/write only access is the biggest problem here, and it will be very hard to solve. An additional problem that arises using RFA is that not all operations are permitted or possible on IPC files. Things like direct read/write are not allowed, certain control operations are not allowed, and things like file locking will work fine... but, the IPC file will be locked, not the remote file. Part of this problem can be solved by writing substitution modules for the standard file system intrinsics FOPEN, FREAD, FWRITE, etc. However, there still remain a lot of other 'intrinsics' that are part of the file system. Some examples :

- IO'OPEN'C, IO'READSEQ'C, IO'WRITEKYD'C, IO'CLOSE'C (COBOL).
- FMTINIT', IIO',TFORM' (FORTRAN).
- CKOPEN, CKREAD, BKWRITE, BKCLOSE (KSAM).

All these modules reside in the system SL. They are part of the run-time library belonging to a specific language.

This is truly the difficult part of PMDS. It will be almost impossible to write (and maintain!) substitution modules for all file system

modules that are part of the system library. Most of these modules are not documented, and not used directly by programmers. (How many COBOL programmers know that they call IO'OPEN'C each time they open a file).

If a user desperately needs to read from and write to a remote file, perform direct reads and file locks, there seems to be only one solution. An advanced version of PMDS could support substitutions for the 'normal' file system intrinsics FOPEN, FCLOSE etc. The functions, performed by these PMDS intrinsics would be similar to the functions of the remote database access modules. If the user is willing (and able) to use these intrinsics in his program, his problem is solved.

However, if only the read/write problem needs to be solved, there is a second, much simpler solution. Suppose the user were able to access two remote files simultaneously. The only thing PMDS should do to support that feature is passing the file number to the remote slave. That filenumber uniquely identifies each file opened by a process. Further it should maintain some table to keep track of the relationship between local and remote file numbers. If PMDS would support that feature, two way file access is possible. Simply open the remote file twice, first with read access, and then with write access. The obvious disadvantage of this method is that special programming changes have to be made, and that it will give more overhead on both systems.

Of course these limitations are partly caused by the wish that Privileged Mode must be avoided as much as possible, and should be very safe if really needed. (How about writing a replacement module for ATTACHIO ??).

6. Conclusions.

The previous chapters have shown that it is possible to build a PMDS-like system, that supports most of the features of DS, without mandatory changes in application software, operating software and/or hardware. The feature of special operations on remote files/peripherals is the bottleneck encountered so far.

Still, we believe that there will be a lot of situations where PMDS can be used very effectively, despite the lack of advanced RFA possibilities.

A quick comparison of DS and an advanced version of PMDS :

	DS	PMDS
Max. speed (cps)	7000	1920
Rem. commands	yes	yes
NFT	yes	yes
PTOP	yes	yes
Rem, DB access	yes	yes
Per. sharing	yes	limited
RFA	yes	limited
X.25/X.21	yes	yes
Concurrent access	yes	no
Hardware costs (\$)	10.000	1.000
Software costs (\$)	4.750	?..???

The prices mentioned above are based on the connection of a series 68 to a series 37, with full software costs.

Building a system like PMDS will not be simple, a lot of special programming techniques are required. The software must run very efficiently to prevent unnecessary overhead, and there still remain a lot of problems to be solved, f.i. break and control-y handling, detection of operator messages that disturb a critical message, detection of system failures and power failures of one of the systems, etc. However, we hope that this paper has shown that it is possible to perform system-to-system communication on a 'DS feature level', by connecting two asynchronous ports with a simple cable.

Biography

Rene van Geesbergen met his first HP3000 system in 1979, when he joined the Dredging Division of the Royal Boskalis Westminster Group as a technical programmer. In 1981 he moved to the Information Department of the same company. His new function was a combination of system manager of two HP3000's and system programmer. In April 1984 Rene and his partner Jelle Grim founded Holland House, a small company specializing in HP3000 system management consultancy, and supporting software. Within Holland House he specializes in communication products and HP3000 internals.

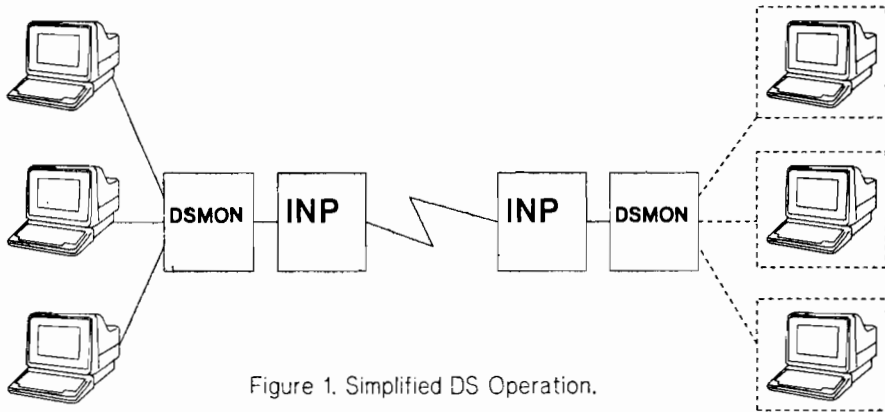


Figure 1. Simplified DS Operation.

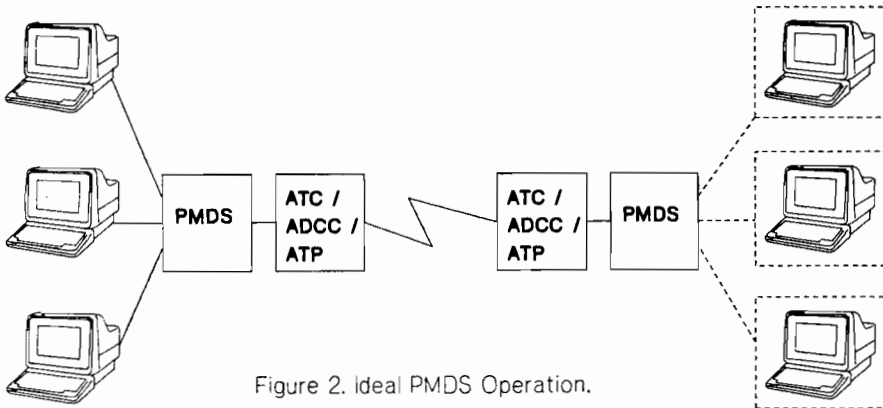


Figure 2. Ideal PMDS Operation.

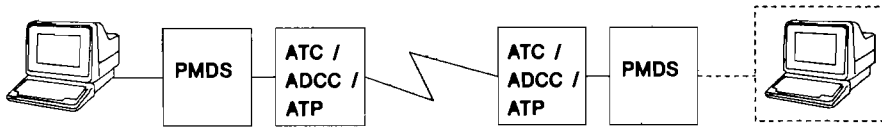


Figure 3. Actual PMDS Operation.

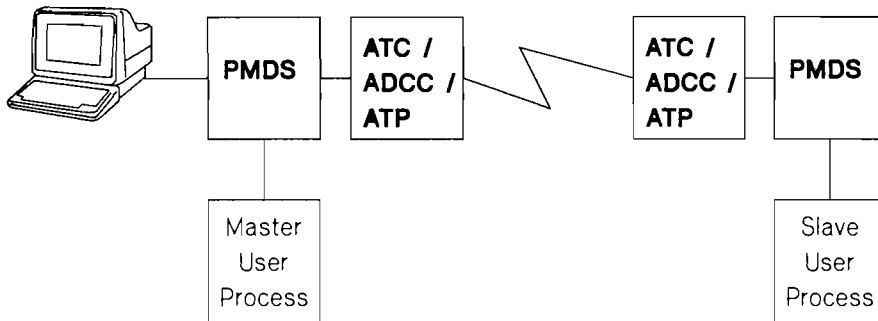


Figure 4. PTOp Principle.

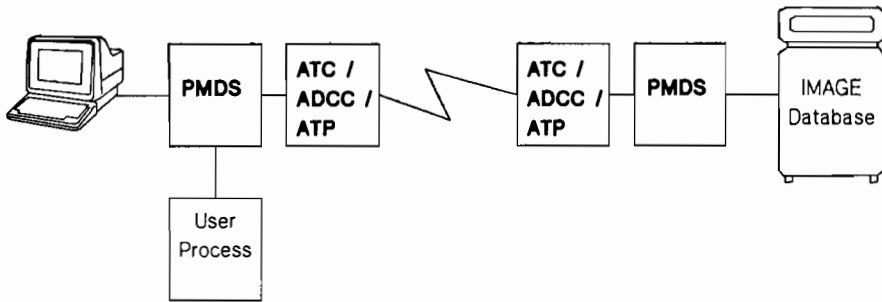


Figure 5. Remote Database Access Principle.

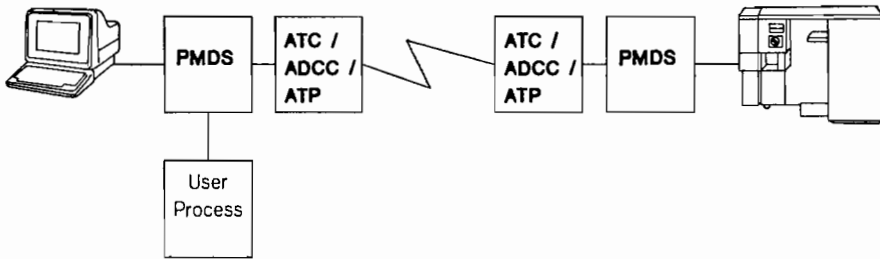


Figure 6. Peripheral Sharing Principle.

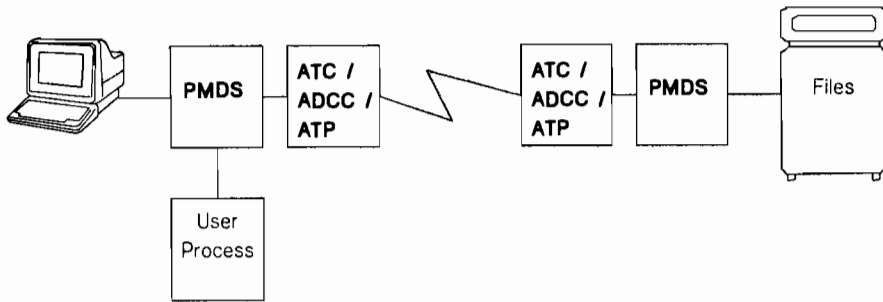


Figure 7. Remote File Access Principle.

3017. A GUIDE TO SOFTWARE EVALUATION AND SELECTION

Roger J. Olsen
Productive Software Systems, Inc.
5617 Countryside Road
Edina, Minnesota 55436
United States of America

SUMMARY

There are many challenges facing the data processing community today. Not the least of which is, the selection of application software. Once a decision is made to purchase a particular package, there are many long term ramifications. This paper will identify several issues that should be explored before you commit to purchasing your application packages. Your other alternatives are to write your own or possibly convert an old package. Each has its place depending upon the circumstances at your site.

The check list of issues includes:

- 1). Your Requirements.
- 2). The Make or Buy Decision.
- 3). The Software Itself.
 - Adaptability
 - Interfaces
 - Reliability
 - Documentation
 - Maintainability
 - Expandability
 - Security
 - Overhead
 - Checks and Balances
 - Recovery
 - Limits
 - Standards
- 4). The Vendor.
 - Training
 - Support
 - Enhancements
- 5). The Software Contract
- 6). Cost vs. Benefit Analysis

The Life of a Typical System

Requirements
Conceptual Design
Detail Design
Development
Implementation
Review
New Requirements

The Requirements

A definition of the objective - the problem to be solved. Who will use the system? What are the major inputs, outputs? What are the expected volumes of inputs, outputs? What are the interface requirements? What hardware, communications, and software are required to support this application? What are the performance objectives such as on-line response times for file maintenance and data entry. What are the expected costs for development, operation, and maintenance? What are the systems benefits and can they be quantified?

The Conceptual Design

A functional description from the users point of view. This includes: the operational and management functions of the system, the systems inputs, including types of transactions, and the types of edits to be performed, the processing methods and techniques to be used, the data that will be maintained within the data base or file systems, the systems outputs, reports, screens, & purpose of each. A functional description from the D.P. point of view which includes: system flow charts, data flow diagrams, an estimate of input, output, and storage requirements, and other system interfaces.

The Detail Design

A detail description of each transaction including the data elements and the source of each. A description of the processing logic. A description of the file structures to be used (MPE, KSAM, IMAGE), and the access methods along with a volume estimate. A description of the outputs, screens, reports, files, security of access to screens & distribution of reports. A description of the backup, recovery, and audit trails. A description of additional programming required to alter a package or provide custom interfaces to other systems.

The Development Phase

Program specifications. Coding and unit testing. System testing. Documentation - Program, system, operation, user. Training - management, users, programmers, analysts, operators. Acceptance by all involved.

The Implementation Phase

Conversion of old files, or loading new data. On-the-job training and problem solving. Interpretation of data on screens and reports. The first test of your documentation.

The Review Phase

How well is it working? How much did it really cost, in dollars and time? What is the level of satisfaction? What needs to be changed and by when?

The New Requirements Phase

Now with a better understanding of the entire system including a long list of change requests, start the process all over again.

The First Real Issue -- Your Requirements

As described above it is assumed you clearly understand and have listed the minimum requirements.

You probably also have some desired, but not necessary requirements. They should also be listed.

In the process of reviewing software packages, some additional features offered by these packages may be added to your list of required or desired features. Several times a package supplier has encountered problems and developed solutions for them that you had not considered in your initial study.

The Second Real Issue -- Make or Buy

We will assume that after a thorough evaluation of what software is available on the market, you and the potential users have selected a few packages to investigate further that seem to meet the majority of your requirements. If a package meets 80% of your requirements, it is generally considered a good fit. If the package is not a 100% fit, which it rarely is, you must then decide to modify your requirements, have the vendor modify the software, or plan on modifying the software yourself. If you plan to modify the package yourself, you will want system documentation before you begin. Generally the vendor can provide modification quicker, although the charges per hour are higher than most outside contractors, and much higher than your own staff. If your staff will be doing the long term enhancements and maintenance, this is a great opportunity for them to learn the new system. Have the vendors quote to your requirements. A firm fixed price to have all the modifications made. And if they make them, how does that affect your support costs?

Before You Buy -- What about the Software Itself

Adaptability

Will it be adaptable to your changing requirements over its expected life? Is it written in a language that you consider appropriate? How machine dependent is it? Generally performance vs machine independence is a trade off. Can the files and data bases be easily changed (i.e. Cobol copylibs or \$include files for all file and dataset layouts? Can your people understand how the code and system is designed? Is the code itself structured to allow easy maintenance? Does the vendor provide you with adequate documentation and or training on such things as the locking strategy employed and the naming conventions used.

Interfaces

Will it have the interfaces you need, and if not, how difficult is it to develop them. Interfaces can be online and/or batch but they should be as independent as possible between modules. This is important so that as your system grows and hardware prices continue to fall, you may want to distribute various applications across several processors. Always keep online interfaces to a minimum.

Reliability

Is the software reliable? How do you find out? Just because it has 200 or more installations, does not mean it is. Don't assume reliability! The biggest flaw of most application software today is the potential of two users trying to update the same record, and while doing so, each believe the software works, but one user overlays the other users transaction. There are several packages on the market that currently do not lock records properly. Check out the locking strategy employed in the code itself before you buy.

Documentation

What does the documentation consist of? Is it enough for your needs? How is it going to be maintained as changes occur? How much you need will vary with each type and your requirements. But don't underestimate your requirements for all the following types of documentation. This is one area where there never seems to be enough or the right type.

Types of Documentation:

User manual, operation manual, system flow charts, data flow charts, program and system manuals including control tables and files, security, cross-references (file to program, file to job stream, element to file, element to program, program to job stream, program to file, calls to program, copy or include files to program), internal audit trails, interfaces to other systems. Documentation should be online and maintained on your computer system to accommodate changes. All of the above cross-references can be generated and maintained automatically through the use of your computer system and the ROBOT/3000 Automatic Documenter software or manually with HP's Dictionary/3000.

Is the Software Maintainable? And by whom?

The useful life of the software can be increased by several years if it is maintainable at a reasonable cost. The design of the application, the use of control tables, the language used, the documentation available, the amount of structure within the language, and the standards used to develop the entire application will determine the maintainability.

Is the Software Expandable?

Expandable or shrinkable as far as field sizes, file sizes, data sets, data bases, program segments. Can an additional field or element be added to a file or data base without recompiling the entire application, or retesting the entire system? Can an unused field, file, or data set be eliminated, or reused for another purpose?

How does the Level of Security fit your needs?

Is it compatible to your existing systems? Will it make your systems more complex than need be?

Are there any Check and Balance Programs Provided?

What is a check and balance program? These are programs that provide an audit of logical data structures, i.e. in a sales order system, each item on order will have a quantity and price. A good system would also carry a total amount on the order header and a check and balance program that would verify the order detail amounts always add up to the total. Inventory systems may allow items to be in several locations, and also carry a total on the master file. Either an online checking of the correctness of these or a batch program that would either sample or process all records to assure logical integrity.

System overhead should be a major concern!

For online response time, establish an acceptable level before you install any applications. Response time cost money, a slow machine lowers the productivity of all users. It also degrades as you put more applications on any one machine.

How do you Estimate overhead?

First locate, talk to, and preferably visit another site that is using the package you are considering. Make sure your intended use and volumes of transactions are similar. Too many sites that I've visited have real performance problems after using a package for a few months. It is very difficult to guess the actual load any application will take except by using another site as a comparison. Many times to reduce system overhead, or increase the transaction throughput, data bases must be redesigned, programs rewritten, additional programs written to convert data to the new data base. And if you do your development on the same machine that is running your production, it becomes even more difficult to arrive at a valid estimate.

An example of unacceptable overhead!

Installing a new general ledger system went great for the first two months. The updating was done by a batch program that was run nightly and was taking about one hour. Then the previous six months of data was loaded and the nightly one hour update now took twelve(12) hours even though the daily transaction rate had not increased. Obviously this was not acceptable. Since the new system was now fully operational, and the old non HP computer had been sold, the need for immediate action was required. The entire process keep getting longer each day. To make matters worse development and testing had to be done on the now overloaded machine. After using various tools to monitor the program and the data base, it was determined the access path had to be changed and the summary amount field moved to a new data set. This required a few conversion programs, and all other programs that accessed that data set and data item to be changed. The actual analysis of which programs needed to be altered was accomplished within minutes using ROBOT/3000's online cross-reference capabilities, and a staff of two programmers were given the specification. A few long nights and a very long week end resulted in the newly structured ledger data base being in full production, with the update process time reduced to less than one hour. You definitely want to avoid this kind of situation, if possible, by ether another users experience, or by carefully analyzing the pur chased software first, and if that is not possible, make sure your have the proper tools for changing the software quickly.

Limits of the software

All systems have limits. Find out what they are. How will they affect your intended use? Several limits are imposed with the use of any file system including DBMS systems and only your use and your volumes of data will determine if they will become a problem. Your vendor should know the limits of their package and review them with you.

Recovery

What happens when the machine fails due to hardware or software? Does the application software support automatic recovery such as transaction logging? Maybe you will not want to use it now, but you may in the future. Does it support logical recovery? There are some that have been developed to support rollback recovery for MPE, KSAM, and IMAGE files. If the software does not have a logical recovery scheme, are the users willing to re-enter their lost data?

Standards

Find out and get a copy of the standards used in the design of the software. If you are to maintain it, this could save you many days, weeks, and perhaps months of learning time. Standards are subjective and written to achieve various goals. Find out what the goals were.

The Vendor -- Training, Support, Enhancements

Training

First Determine your training needs. How much, if any is available. Where is the training conducted? What type of areas does the training include (users, operators, analysts, programmers, management)? How much do the various training options cost? Is there any online training available?

Support

Determine what level of support and for what functions do you need support for. Don't buy more than you need. How much support and at what times will support be available? What type of areas does the support cover? Support for users, operators, analysts, programmers, management, or only software bugs.

Enhancements

What is the vendors track record for enhancements to the package? What are the planned enhancements? Do you have to install the enhancements to receive support? How do you install updates, if you have modified the package? A source code comparison tool is a must if that is the case. One strong feature of buying packages is the future enhancements. How much input do you have, as to the direction of these enhancements? How should you communicate these requests to the vendor?

The Software Contract -- Know What you are Signing!

Generally most software contracts have several general items that will protect you from law suits, etc. In addition to those general items, make sure the terms of support are covered. And if possible, get the option to purchase additional copies covered as well. Who knows how many machines will be using it in the years to come. The largest and several times overlooked aspect of a software contract is the liability the purchaser assumes. Although this generally is required to protect your vendor of unauthorized copies being distributed, it is a liability. Your company should have a method of safeguarding the software you buy and clearly explaining to all individuals that have access to it, what your policy is.

COST vs. BENEFIT ANALYSIS

(A MATRIX APPROACH)

Put your spread sheet program to work for you here. List your requirements down the left side of the spread sheet. Start first with your absolute needs. Next add your desired options. Next add the features that make a package more attractive. Include all needs, and don't forget about support. Now add the negative items that you have discovered. Cover all aspects including the items in this paper, and your own specific needs. Now assign a relative number to each item in the list. This could be +100 to -100. This number should represent how important each of these items are to the decision. Now add each solution across the top of the spread sheet, vendor #1, vendor # 2, vendor #2 with modifications, in house customizing, total in house developed solution. Now fill in the blanks by rating each solution from 1 to 10 on how well you believe the solution meets the requirements list. Obviously after multiplying each of these and summing them, you have as nonbiased a rating for each solution as is normally possible. We have found this method a useful tool in making software selections. The main problem is still the items you overlook. Remember the more items generally yields a more accurate picture of reality.

The MATRIX Approach Solutions for an Order Entry System

	Rel Value +100 to -100	vendor #1	vendor # 2	vendor # 2 Mods	Total in house
Absolute needs:					
allow 200 items per order	100	10	5	10	10
maximum of 25,000 customers	100	10	5	10	10
up to 99 ship to addresses	100	10	10	10	10
back orders per item	100	10	10	10	10
automatic credit approval	100	3	9	10	10
on line maintenance	100	8	10	10	10
inquiry by item	100	10	10	10	10
2500 orders per day	100	10	10	10	10
Desired features:					
ship from multiple warehouse	80	5	9	10	10
requires new HP 3000 48 or >	-50	6	1	1	10
software cost high	-100	5	2	6	10
inquiry response < 1 sec	35	7	5	5	10
online interface to AR	35	3	10	10	10
written in structured Cobol	55	1	10	10	10
proven reliability	90	8	5	1	1
Documentation Online	25	8	9	9	1
Other features:					
Easily Maintained	60	1	8	3	1
Easily Expanded	50	1	8	3	5
Compatible security	20	3	1	10	10
good checks and balances	35	1	6	7	1
design standards available	25	1	3	1	10
local training available	10	6	2	1	10
good enhancements history	35	9	3	1	1
TOTAL RATING PER SOLUTION:		8,630	10,430	10,385	9,595
** Vendor # 2 is the choice with highest rating **					

Biography

Roger Olsen is the C.E.O. of Productive Software Systems, Inc (PSS) which develops and markets a family of management and productivity tools for the HP 3000 under the trade name ROBOT/3000. Roger has worked with the HP for over 6 years, and installed several application systems while serving as the director of information systems for a large manufacturing firm.

3018. Things That Go Bump in The 3000
and
"Non-Traditional" uses for OPT.
Denys P. Beauchemin
Keith Bowers
Nothern Telecom Inc.,
4001 East Chapel hill-Nelson Highway,
P.O. Box 13010
Research Triangle Park, NC 27709

Introduction

The reader should be aware that this is a technical presentation. It is aimed for the system manager that has a certain degree of expertise on the HP3000. We make no representation of any kind on any of the techniques that will be shown herein. While we have successfully used these techniques, we are fully aware that there may be other ways to attain the same results. The reader MUST be totally cognizant of the fact that HP does NOT support these techniques and that the reader would be using them at his or her own risk. The things that will be layed out here should only be used as a LAST RESORT. You may be successfull, and then again you may magnify the problem. SO BEWARE.

Now with this out of the way, let us begin.

In this paper, we will be adressng many aspects of the 3000 that are not covered by any HP User-oriented course. The following is a list of the subjects that will be covered in the first half of the talk.

- o A short course on DISKED5.
- o How to deal with account/group disk space confusion.
- o How to deal with accounts/groups that can't be purged.
- o How to deal with disappearing files and file label corruption.
- o Some IMAGE tricks (totally verboten as per HP).

The second half of the talk will deal with:

OPT is a tool sold by HP for global resource utilization monitoring. It is usually considered a means of short-term, global monitoring. We would like to discuss the use of OPT to extract rather detailed information about a job/process and then as a means of monitoring long-term system activity.

- o A short course on DISKED5

There is a utility on the system called DISKED5.PUB.SYS. All systems have it. It is supplied and supported by HP. This utility permits the SM user to read and write anywhere on disk. Yes, it's a dangerous tool in the hands of the uninitiated. But we will put these fears aside, and start to play with DISKED5.

The first thing you should do is to make sure that you have SM capability. I always have SM and PM and all the other goodies, so I sometimes forget that not everyone is like me. But get SM first, then run DISKED5.PUB.SYS.

DISKED5 G.01.00 (C) HEWLETT-PACKARD CO., 1983

TYPE 'HELP' FOR INFO

>help

DISKED5 allows to dump and/or modify : file contents or any disc sector (sys. mgr capability is required).

```

B[ASE] [<ABS SEC #>]
DEBUG
DISC <LOG DEV #>
D[UMP] [ [<REL SEC #>] [, <# OF SEC>] ] OR [<'ALL'>] [, A=ASCII ]
      (AT LEAST ONE PARAMETER MUST BE PRESENT.)
F[ILE] <FILENAME>
L[IST] [<DEVICE CLASS>] OR [<LOG DEV #>]
M[ODIFY] <SEC NUM, REL WORD ADDR [,NUM OF WORDS]>
      (NEW VALUE STARTS WITH : # - DECIMAL, ' - ASCII)
W[IDTH]
E[XIT]

```

This is the list of commands acceptable to DISKED5.

BASE will specify the base from where you will be working.
If undefined, you will be relative to sector %0.
More on this with examples later.

DEBUG calls MPE DEBUG within the program.

DISC specifies the logical device for the disc you wish to paly with.

DUMP will dump the contents of the disc starting at the sector specified relative to BASE and for as many sectors as you wish. You may specify ASCII-only.

FILE specifies a file on the system. This is used for quick access to the file label & contents.

LIST is used to redirect the listing of DISKED5.

MODIFY will change the contents on disc at the location and for the length that you specify. You may start your replacement values with # to specify decimal values, or ' to specify ascii values.

WIDTH is used to specify a narrow format. I always use it.
Answer YES in uppercase to the question.

o How to deal with account/group disk space confusion.

There may come a time when after you issue the command :REPORT @.@, you get up to hit someone over the head because they are using a huge amount of disk space. But before you sneak up on the target in question, you decide to arm yourself with some hard-hitting facts. So you naturally do a listf,2 of the account to your nearest printer. You rip out the report and as you approach the object of your wrath, you glance at the listf. Lo and behold, there is no way that the files and their sizes can even approach the total disk space figure shown with report. My God, if you can't trust REPORT @.@ who can you trust. I have seen this situation only once, but the REPORT said that an account was using 2.9 million sectors, when in actual fact the usage was closer to 70,000. What went wrong? Well, I guess that somewhere along, the directory was not made aware that someone had purged a database which did consume the remaining 2.8 million sectors. The file system got confused ?, maybe I don't know. But I can fix this glitch, and I can do so without a reload. Here is how I fixed that one.

1- Let's see what is out there now:

```
:REPORT @.DENYS
```

ACCOUNT	FILESPACE-SECTORS	CPU-SECONDS	CONNECT-MINUTES
/GROUP	COUNT LIMIT	COUNT LIMIT	COUNT LIMIT
DENYS	2000030 **	0 **	0 **
/PUB	2000030 **	0 **	0 **

Well, that's a fair size account, let us see what the files look like.

```
:LISTF @.@.DENYS,2
```

```
ACCOUNT= DENYS          GROUP= PUB
FILENAME CODE  -----LOGICAL RECORD-----  ---SPACE---
```

FILEA	128W	FB	0	1000	1	1001	16	16
FILEB	128W	FB	0	1000	1	1001	16	16
FILEC	128W	FB	0	1000	1	1001	16	16

Here is the problem, the total sectors should be 3003, not 2000030.

```
:R DISKED5    <<HP's trusty utility>>
```

```
DISKED5 G.01.00 (C) HEWLETT-PACKARD CO., 1983
```

```
>debug
```

```
*DEBUG* PRIV.0.1000
```

```
?da 1130,2    <<find dirbase from Absolute 1130>>
```

```
A1130 000000 002215
```

```
?e
```

```
>BASE %2215
```

```
NARROW FORMAT?
```

```
Y
```

```
>DUMP 3      <<Always third sector for dirbase>>
```

```
SECTOR %00000002220    LDEV = %000001
```

```
000: 110143 000004 000000 000021 010743 000000 020040 020040
```

```
010: 020040 020040 041511 050120 051440 020040 000010 000006
```



```

020: 045117 044116 020040 020040 001137 000004 050525 044532
030: 020040 020040 005076 000003 051531 051440 020040 020040
040: 001420 000004 002004 002004 002004 002004 002004 002004
050: 002004 002004 002004 002004 002004 002004 002004 002004
060: 002004 002004 002004 002004 002004 002004 002004 002004
070: 002004 002004 002004 002004 002004 002004 002004 002004
100: 002004 002004 002004 002004 002004 002004 002004 002004
110: 002004 002004 002004 002004 002004 002004 002004 002004
120: 002004 002004 002004 002004 002004 002004 002004 002004
130: 002004 002004 002004 002004 002004 002004 002004 002004
140: 002004 002004 002004 002004 002004 002004 002004 002004
150: 002004 002004 002004 002004 002004 002004 002004 002004
160: 002004 002004 002004 002004 002004 002004 002004 002004
170: 002004 002004 002004 002004 002004 002004 002004 002004

```

```

      .1234567.1234567.1234567.1234567.1234567.1234567.1234567.1234567
000: .c.....          CIPPS    ....JOHN    .._QUIZ    >..SYS
100: .....
200: .....
300: .....

```

We are looking for account DENYS, it comes after CIPPS but before JOHN so we use the pointer of account CIPPS, it's at word %16; value : %10;

```

>DUMP %10    <<%10 taken from above, everything is relative to>>
              <<dirbase>>

```

```

SECTOR %00000002225          LDEV = %000001
000: 041511 050120 051440 020040 000215 000662 070003 000601
010: 000000 000000 020040 020040 020040 020040 000000 000000
020: 077777 177777 000000 000002 077777 177777 000000 000003
030: 077777 177777 002525 000226 000000 000000 041517 052040
040: 020040 020040 000055 000066 072607 000613 000000 000000
050: 020040 020040 020040 020040 000023 001320 077777 177777
060: 000000 000032 077777 177777 000000 000470 077777 177777
070: 002525 040226 000303 000000 042105 047131 051440 020040
100: 000173 000661 177677 000713 000000 000000 053505 044522
110: 042040 020040 000036 102236 077777 177777 000000 000000
120: 077777 177777 000000 000000 077777 177777 002525 000226
130: 000000 000000 042123 031065 033440 020040 000307 000310
140: 070003 000600 000000 000000 020040 020040 020040 020040
150: 000000 006311 077777 177777 000000 000010 077777 177777
160: 000000 000007 077777 177777 002525 000226 000000 000000
170: 043111 051503 044105 051040 000465 000466 177637 000713

```

```

      .1234567.1234567.1234567.1234567.1234567.1234567.1234567.1234567
000: CIPPS    ....p.....          .....U.....COT
100:    .-.6u.....          .....8.....U@.....DENYS
200:    .{.....WEIRD          .....U.....DS257    ....
300:    p.....          .....U.....FISCHER .5.6....

```

We now have the account entry for DENYS, and if you look closely you can see the account password: WEIRD. Now the account entry comprises quite a bit of information, but at this time the two pieces that we want are the account disk space and the group pointer. The disk space information is at word %16 of the account entry. Note that this is a double word value. The group pointer is located at %4 of the entry, it's a single word long. In our example, the disk space value is found at %112 and the value is %000036, %102236. The group index is at location %100 and the value %173 can be extracted from that location.

>DUMP %173

```

SECTOR %00000002410          LDEV = %000001
000: 104141 000001 000001 000001 005222 000003 042105 047131
010: 051440 020040 050125 041040 020040 020040 001052 000001
020: 000401 000401 000401 000401 000401 000401 000401 000401
030: 000401 000401 000401 000401 000401 000401 000401 000401
040: 000401 000401 000401 000401 000401 000401 000401 000401
050: 000401 000401 000401 000401 000401 000401 000401 000401
060: 000401 000401 000401 000401 000401 000401 000401 000401
070: 000401 000401 000401 000401 000401 000401 000401 000401
100: 000401 000401 000401 000401 000401 000401 000401 000401
110: 000401 000401 000401 000401 000401 000401 000401 000401
120: 000401 000401 000401 000401 000401 000401 000401 000401
130: 000401 000401 000401 000401 000401 000401 000401 000401
140: 000401 000401 000401 000401 000401 000401 000401 000401
150: 000401 000401 000401 000401 000401 000401 000401 000401
160: 000401 000401 000401 000401 000401 000401 000401 000401
170: 000401 000401 000401 000401 000401 000401 000401 000401

.1234567.1234567.1234567.1234567.1234567.1234567.1234567.1234567
000: .a.....DENYS  PUB  .*.....
100: .....
200: .....
300: .....
```

We now have the pointer for the entry to the group PUB, this block will tell us where to find it. The group pointer is at location %4 and in this example, it is at %16 and the value is %1052

>D %1052

```

SECTOR %00000003267          LDEV = %000001
000: 050125 041040 020040 020040 001047 020040 020040 020040
010: 020040 000036 102236 077777 177777 000000 000000 077777
020: 177777 000000 000000 077777 177777 020143 015006 000713
030: 000000 001051 020040 020040 020040 020040 020040 020040
040: 020040 020040 020040 020040 020040 020040 000000 000000
050: 000000 000000 000000 000000 000000 000000 000000 000000
060: 000000 000000 000000 000000 000000 000000 000000 000000
070: 000000 000000 000000 000000 000000 000000 000000 000000
100: 000000 000000 000000 000000 000000 000000 000000 000000
110: 000000 000000 000000 000000 000000 000000 000000 000000
```

```

120: 000000 000000 000000 000000 000000 000000 000000 000000
130: 000000 000000 000000 000000 000000 000000 000000 000000
140: 000000 000000 000000 000000 000000 000000 000000 000000
150: 000000 000000 000000 000000 000000 000000 000000 000000
160: 000000 000000 000000 000000 000000 000000 000000 000000
170: 000000 000000 000000 000000 000000 000000 000000 000000

```

```

      .1234567.1234567.1234567.1234567.1234567.1234567.1234567.1234567
000: PUB      .      ..... c.....)
100:      .....
200:      .....
300:      .....

```

The group entry also has a lot of information, but at this time, what we are after is simply the disk space information. It is found at location %11 and again, it's a double word value. In this example it is indeed at %11,2 and the value is the same as the account entry: %000036,%102236.

Now comes the time to make our changes. With our trusty HP16c calculator, we see that the value for 3003 decimal is in reality %5673. We will place this value at the proper location. We must also not forget that we are placing this value in a double word; so in actual fact 3003 decimal is %000000, %005673 in double word octal. Let's try it and see.

```

>MODIFY %1052,%11,2  <<First we modify the group entry>>
SECTOR %00000003267  LDEV = %000001
011: %000036,0      <<Remember, it's a double word value>>
012: %102236,%5673
WRITTEN
>MODIFY %10,%112,2  <<And then we modify the account entry>>
SECTOR %00000002225  LDEV = %000001
112: %000036,0      <<Again, double word value>>
113: %102236,%5673
WRITTEN
>E

```

END OF PROGRAM

:REPORT @.DENYS

ACCOUNT	FILESPEC-SECTORS		CPU-SECONDS		CONNECT-MINUTES	
	COUNT	LIMIT	COUNT	LIMIT	COUNT	LIMIT
/GROUP	3003	**	0	**	0	**
DENYS	3003	**	0	**	0	**
/PUB	3003	**	0	**	0	**

:

And that's all there is to that problem.

o How to deal with accounts/groups that can't be purged.

I have only seen this type of problem, in real life, twice. One tries to purge an account, and MPE comes back with the message that IN USE; CAN'T PURGE, and there is no one logged onto that account, and no one is accessing any files within the account or the group.

Let's see how MPE purges an account; first MPE will purge all users providing that the user is not in use at that point. There is a flag in the directory that keeps track of the number of active session/jobs for each user. (In fact MANAGER.SYS is hardcoded with a starting active count of 1 instead of 0, therefore you should never be able to purgeuser MANAGER.SYS.) MPE will then purge all the files in every group. As the groups are emptied of files, the groups themselves are purged. When all groups are purged, the account is then purged. You can see that if a file is busy, its group will not be purged, and if a group is not purged, the account will not be purged either.

Therefore, if a file is busy, the account is not purged. If a user is logged on, the account is not purged. So, when you issue the command PURGEACCT xyz, and MPE comes back after a long time and says: IN USE, CAN'T PURGE, you just do a listf of the files in the account and you get a listing of the files that are in use. Lo and behold, there are no files in the account. Well then, do a LISTUSER @.xyz and you will get a list of the users. At this point, you get one of two things; one, you see one or more users; they could be logged on, or the counters in the users are not equal to zero. If the last is true, then pull out your tables manual, follow the directory to the user entry and reset the counter.

If there are no users, you are facing three possibilities. 1- The user count in the account entry is non-zero, use DISKED5 and the tables manual to reset the counter. 2- There are still some groups left unpurged, do a LISTGROUP @.xyz and see if in fact there are any groups. If there are groups, you may have to set their active bit to zero. 3- There are no groups, but the group count within the account entry in the directory is non-zero. Again, use DISKED5 and your tables manual to reset the counter.

o How to deal with disappearing files and file label corruption.

When you use a file, the file system does I/O work on the file label on your behalf. This procedure uses the intrinsic FLABIO. If FLABIO encounters an I/O error it assumes that the file is bad. It will therefore effectively remove it from the system by making it inaccessible. To accomplish this, FLABIO will "flip" the high-order bit of the 3rd word of the file name in the directory entry. If you do a LISTF to the specific file name, you will get a NON-EXISTANT FILE. Yet, if you do a LISTF with a wild card after the first four characters, you will see the file. If you have extended ASCII on your terminal, you will see a strange character in lieu of the regular fifth character. This means that FLABIO has encountered a problem with this file label and that it presumes the file to be bad. This is not always the case. In order to be able to access the file, you must go into the directory and fix this entry.

Please read the previous example in order to understand the directory layout. I will only outline new material or differences in this example.

Here is how you proceed.

```
listf deny@,2
ACCOUNT= DENYS      GROUP= PUB
```

FILENAME	CODE	-----LOGICAL RECORD-----				----SPACE----			
		SIZE	TYP	EOF	LIMIT	R/B	SECTORS	#X	MX
DENYSFIL		128W	FB	0	1023	1	128	1	8
DENYSUDC*		80B	FA	18	18	16	15	1	1

```
/run disked5
DISKED5 G.01.00 (C) HEWLETT-PACKARD CO., 1983
TYPE 'HELP' FOR INFO
>debug
```

```
*DEBUG* PRIV.0.1000
?da 1130,2
A1130 000000 002321
?e
>base %2321
>w
NARROW FORMAT?
Y
>d 3
```

```
SECTOR %00000002324      LDEV = %000001
000: 110143 000014 000003 000125 010743 000000 020040 020040
010: 020040 020040 040503 052040 020040 020040 000010 000003
020: 040522 042126 040522 041440 011347 000006 041511 052040
030: 020040 020040 000263 000013 042105 050124 030461 031060
040: 007143 000010 043117 054040 020040 020040 000407 000010
050: 045117 047105 051440 020040 000701 000006 046515 046515
060: 020040 020040 001025 000006 046517 047124 051105 040514
070: 006717 000006 050107 046440 020040 020040 001151 000012
100: 051117 051461 030060 041440 001275 000010 051503 051125
110: 043440 020040 007733 000003 051531 051440 020040 020040
120: 000544 000012 005012 005012 005012 005012 005012 005012
130: 005012 005012 005012 005012 005012 005012 005012 005012
140: 005012 005012 005012 005012 005012 005012 005012 005012
150: 005012 005012 005012 005012 005012 005012 005012 005012
160: 005012 005012 005012 005012 005012 005012 005012 005012
170: 005012 005012 005012 005012 005012 005012 005012 005012
```

```
.1234567.1234567.1234567.1234567.1234567.1234567.1234567.1234567
000: .c.....U....      ACT      ....ARDVARC ....CIT      ....DEPT1120
100: .c..FOX      ....JONES      ....MMM      ....MONTREAL....PGM      .i..
200: ROS100C ....SCRUG      ....SYS      .d.....
300: .....
```

We will look at the 3 sectors starting at location %263, because I know that there are lots of accounts. You can just keep stepping by one if you do not find your account on the first page. So here goes.

>d %263,3

```

SECTOR %00000002604      LDEV = %000001
000: 041511 052040 020040 020040 000275 000276 070003 000600
010: 000000 000000 041517 051524 051501 053105 000000 163452
020: 077777 177777 000000 155202 077777 177777 000000 147066
030: 077777 177777 004551 000226 000000 000000 041514 052105
040: 047040 020040 007161 007162 070203 000600 000000 000000
050: 050105 047107 020040 020040 000000 123523 077777 177777
060: 000002 171661 077777 177777 000000 175033 077777 177777
070: 004545 040226 000532 000000 041517 043516 047523 020040
100: 002551 002552 070003 000613 000000 000000 052110 044522
110: 042040 020040 000000 011127 077777 177777 000000 000003
120: 077777 177777 000000 000002 077777 177777 004551 000226
130: 000000 000000 041517 046126 044516 020040 003552 003625
140: 160607 000713 000000 000000 041501 051105 043125 046114
150: 000000 000014 077777 177777 000000 000037 077777 177777
160: 000000 000057 077777 177777 002525 000226 000000 000000
170: 041517 047114 042504 020040 000317 000320 072203 000600

```

```

.1234567.1234567.1234567.1234567.1234567.1234567.1234567.1234567
000: CIT      ....p.....COSTSAVE...*.....6.....i.....CLTE
100: N      .q.rp.....PENG      ...S.....e@.Z..COGNOS
200: .i.jp.....THIRD      ...W.....i.....COLVIN .j..
300: .....CAREFULL...../.....U.....CONLED      ....t...

```

```

SECTOR %00000002605      LDEV = %000001
000: 000000 000000 042117 047522 020040 020040 000021 115617
010: 077777 177777 000021 144306 077777 177777 000004 074767
020: 077777 177777 004545 040226 000242 000000 041517 051524
030: 020040 020040 001520 001521 072203 000600 000000 000000
040: 046517 047105 054440 020040 000000 005724 077777 177777
050: 000003 016757 077777 177777 000000 002100 077777 177777
060: 002525 040226 000474 000000 041517 052040 020040 020040
070: 000047 000050 072207 000611 000000 000000 053511 046114
100: 053517 051113 000000 000000 077777 177777 000010 125763
110: 077777 177777 000001 006223 077777 177777 004545 000226
120: 000000 000000 041524 046040 020040 020040 000432 000433
130: 070203 000600 000000 000000 046505 042111 040514 020040
140: 000000 075757 077777 177777 000000 010563 077777 177777
150: 000000 004711 077777 177777 004551 040226 001304 000000
160: 042101 052101 041517 046515 000715 000716 072003 000600
170: 000000 000000 052105 046105 020040 020040 000000 000142

```

```

.1234567.1234567.1234567.1234567.1234567.1234567.1234567.1234567
000: ....DOOR      .....y.....e@.....COST      .P.Qt.....
100: MONEY      .....@.....U@.<..COT      .'.(t.....WILL
200: WORK.....e.....CTL      ....p.....MEDIAL
300: ..{.....s.....i@.....DATACOMM.....t.....TELE      ...b

```

```

SECTOR %00000002606      LDEV = %000001
000: 077777 177777 000000 000730 077777 177777 000000 002750
010: 077777 177777 002525 000226 000000 000000 042103 046440
020: 020040 020040 003633 003634 070203 000613 000000 000000

```

030:	020040	020040	020040	020040	000006	062663	077777	177777
040:	000006	110226	077777	177777	000001	107420	077777	177777
050:	004545	040226	000262	000000	042105	047131	051440	020040
060:	006043	006044	167607	000713	000000	000000	020040	020040
070:	020040	020040	000000	056056	077777	177777	000000	013744
100:	077777	177777	000000	047006	077777	177777	002525	000226
110:	000000	000000	000000	000000	000000	000000	000000	000000
120:	000000	000000	000000	000000	000000	000000	000000	000000
130:	000000	000000	000000	000000	000000	000000	000000	000000
140:	000000	000000	000000	000000	000000	000000	000000	000000
150:	000000	000000	000000	000000	000000	000000	000000	000000
160:	000000	000000	000000	000000	000000	000000	000000	000000
170:	000000	000000	000000	000000	000000	000000	000000	000000

.1234567.1234567.1234567.1234567.1234567.1234567.1234567.1234567

```

000: .....U.....DCM .....p..... ..e.....
100: .....e@.....DENYS .#.$..... ..\.....
200: .....N.....U.....
300: .....

```

>d %6043

SECTOR	%00000010364	LDEV = %000001						
000:	104141	000001	000001	000002	005222	000003	042105	047131
010:	051440	020040	050105	051106	046111	041040	006050	000002
020:	001002	001002	001002	001002	001002	001002	001002	001002
030:	001002	001002	001002	001002	001002	001002	001002	001002
040:	001002	001002	001002	001002	001002	001002	001002	001002
050:	001002	001002	001002	001002	001002	001002	001002	001002
060:	001002	001002	001002	001002	001002	001002	001002	001002
070:	001002	001002	001002	001002	001002	001002	001002	001002
100:	001002	001002	001002	001002	001002	001002	001002	001002
110:	001002	001002	001002	001002	001002	001002	001002	001002
120:	001002	001002	001002	001002	001002	001002	001002	001002
130:	001002	001002	001002	001002	001002	001002	001002	001002
140:	001002	001002	001002	001002	001002	001002	001002	001002
150:	001002	001002	001002	001002	001002	001002	001002	001002
160:	001002	001002	001002	001002	001002	001002	001002	001002
170:	001002	001002	001002	001002	001002	001002	001002	001002

.1234567.1234567.1234567.1234567.1234567.1234567.1234567.1234567

```

000: .a.....DENYS PERFLIB .(.....
100: .....
200: .....
300: .....

```

>d %6050

SECTOR	%00000010371	LDEV = %000001						
000:	050105	051106	046111	041040	011630	020040	020040	020040
010:	020040	000000	034143	077777	177777	000000	006021	077777
020:	177777	000000	001515	077777	177777	002041	004102	000713
030:	000000	011632	020040	020040	020040	020040	020040	020040
040:	020040	020040	020040	020040	020040	020040	000000	000000

050:	000000	050125	041040	020040	020040	006045	020040	020040
060:	020040	020040	000000	021713	077777	177777	000000	005723
070:	077777	177777	000000	045271	077777	177777	020143	015006
100:	000713	000000	006047	020040	020040	020040	020040	020040
110:	020040	020040	020040	020040	020040	020040	020040	000000
120:	000000	000000	000000	000000	000000	000000	000000	000000
130:	000000	000000	000000	000000	000000	000000	000000	000000
140:	000000	000000	000000	000000	000000	000000	000000	000000
150:	000000	000000	000000	000000	000000	000000	000000	000000
160:	000000	000000	000000	000000	000000	000000	000000	000000
170:	000000	000000	000000	000000	000000	000000	000000	000000

```

.1234567.1234567.1234567.1234567.1234567.1234567.1234567.1234567
000: PERFLIB ..      ..8c.....M.....!B.....
100:          .....PUB      .%      ..#.....J..... c..
200: .....
300: .....

```

>d %6045

SECTOR	%00000010366	LDEV = %000001						
000:	100142	000002	000001	000110	000142	006043	050125	041040
010:	020040	020040	040503	051503	030063	034065	003623	000044
020:	046522	050105	047126	020040	006441	000044	022044	022044
030:	022044	022044	022044	022044	022044	022044	022044	022044
040:	022044	022044	022044	022044	022044	022044	022044	022044
050:	022044	022044	022044	022044	022044	022044	022044	022044
060:	022044	022044	022044	022044	022044	022044	022044	022044
070:	022044	022044	022044	022044	022044	022044	022044	022044
100:	022044	022044	022044	022044	022044	022044	022044	022044
110:	022044	022044	022044	022044	022044	022044	022044	022044
120:	022044	022044	022044	022044	022044	022044	022044	022044
130:	022044	022044	022044	022044	022044	022044	022044	022044
140:	022044	022044	022044	022044	022044	022044	022044	022044
150:	022044	022044	022044	022044	022044	022044	022044	022044
160:	022044	022044	022044	022044	022044	022044	022044	022044
170:	022044	022044	022044	022044	022044	022044	022044	022044

```

.1234567.1234567.1234567.1234567.1234567.1234567.1234567.1234567
000: .b.....H.b.#PUB      ACSC0385...$MRPENV      !.$$$$$$$$$$$$$$$$$$$$
100: $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
200: $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
300: $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

```

Just like the entries for account, groups and users, files are kept the same way. Thus DENYSFIL comes after ACSC0385 and before MRPENV, so we look at location starting at %3623.

>d %3623

SECTOR	%00000006144	LDEV = %000001						
000:	040503	051503	030063	034065	005012	123607	040503	051503
010:	030066	034065	003400	046241	040517	020040	020040	020040
020:	001415	136625	040520	051115	052110	020040	003400	074537
030:	041117	051523	020040	020040	001400	071664	041517	052111

```

040: 046122 020040 002011 033365 041522 044524 044503 040514
050: 002410 052220 042102 040516 040514 054523 002001 005603
060: 042105 047131 151506 044514 002002 012076 042105 047131
070: 051525 042103 004000 064277 042111 051105 041524 020040
100: 003012 103230 042111 051513 030440 020040 004001 030642
110: 042522 051117 051114 047507 004401 001532 044516 051524
120: 040514 046040 004413 031200 044516 052105 051123 044523
130: 004007 177761 044516 053101 042105 051040 005012 123741
140: 044516 054040 020040 020040 001415 136637 045101 053060
150: 030460 051503 002011 033411 045101 053064 030060 051503
160: 002410 052256 045101 053067 030460 051503 003012 103261
170: 045101 053067 031060 051440 003407 047710 045101 053067

```

```

.1234567.1234567.1234567.1234567.1234567.1234567.1234567.1234567
000: ACSC0385....ACSC0685..L.AO      ....APRMTH ..y BOSS    ..s.COTI
100: LR  ..6.CRITICAL..T.DBANALYS....DENY.FIL...>DENYSUDC..h.DIRECT
      | this is the problem.

```

```

200: ...DISK1  ..1.ERRORLOG...ZINSTALL ..2.INTERISIS....INVADER ....
300: INX      ....JAV01OSC..7.JAV400SC..T.JAV710SC....JAV720S ..O.JAV7

```

```

>m %3623,%62,1
SECTOR %00000006144      LDEV = %000001
062: %151506,%051506
WRITTEN
>e
END OF PROGRAM

```

```

TDP/3000
/listf deny@,2
ACCOUNT= DENYS      GROUP= PUB

```

FILENAME	CODE	-----LOGICAL RECORD-----	----	SPACE----	
	SIZE	TYP	EOF	LIMIT R/B	SECTORS #X MX
DENYSFIL	128W	FB	0	1023 1	128 1 8
DENYSUDC*	80B	FA	18	18 16	15 1 1

And the file is ready. I strongly suggest that you verify right away that the data is in fact correct. Use FCOPY or some other utility.

o Some IMAGE tricks

The following tricks will permit you to do things to an IMAGE database. You must remember two things about database. One, they are ordinary, albeit privileged, files. Two, IMAGE checks its own integrity whilst in use.

Therefore, you can access an IMAGE dataset or root file as though it was a flat file. But make sure that what you will do will be acceptable by IMAGE. Always DBSTORE your databases before you attempt to play with them.

- 1-Change the name of the database.
- 2-Reset the DBCONTROL flag after a system failure.
- 3-Move databases around discs or around accounts.

All the above can be accomplished in a few minutes for any database. And none of these procedures required an unload or a reload. Granted, there are utilities that exist that do the same things, but not everyone has these on their system.

In order to twiddle a database, it is highly recommended that you acquire the IMAGE/3000 HANDBOOK.

What makes a database privileged? Well the answer is very simple; the file code is negative. So, in order to play with a database with tools like FCOPY or such like utilities, one must make them non-privileged. The easiest and cheapest way is to use DISKED5. Here is the procedure to make a dataset or root file non-privileged:

- 1-First run LISTDIR5.PUB.SYS
 - listf the file name with ;PASS
 - get the following information:
 - i - File code (-400:root, -401:dataset)
 - ii - Get device number.
 - iii - Get adress of label: eg %0004356234.
- 2-Run DISKED5.PUB.SYS
 - DISC devicenumber
 - BASE %labeladress (don't forget the %)
 - WIDTH narrow format.
 - D 0
 - You will obtain a listing of the file label.
 - M 0,%32,1 <<set file code to zero>>
 - M 0,%42,1 <<checksum to zero>>
 - answer with 0 for both prompts.

That's it!, exit from DISKED5 and enjoy yourself. Note: the reason that you must modify two locations is that you MUST set the checksum to %0 before you FOPEN the file.

-To change the name of the database, you can now issue a rename on each and every dataset. Don't forget the root file. After you have renamed all the files, you must change the database name contained in the first record of the root file. You may either write a program, or you can use DISKED5.

-To move databases around accounts or discs, you can issue FCOPYs to all the datasets.

After you have finished with the database, you must go back in DISKED5 and put the file codes to their original values. Don't forget that if

you have FCOPIed the files, their location will be different, use LISTDIR5 first. Don't forget to reset the checksums.

In order to reset the DBCONTROL flag after a system crash, you do not have to change filecodes. You simply use DISKED5 to go to the first label of the root file. The first word of the first label will contain an 'RM', you simply change that to 'FW'. You can now use your database. WARNING please be aware that if the DBCONTROL flag was set in the first place, the database may now be totally inconsistent. In fact, I have only seen one or two occasions where this technique was usefull, but it did save 15 hours of reload time.

"Non-Traditional" uses for OPT -

An example will be used to illustrate the use of OPT to obtain detailed job/process information. The program is a dummy to keep things simple. We will assume that this program is part of a stream that normally runs from 2000 thru 2200 every week night and is still running at 0700 the next morning. Is something hung? in a loop? or just trying to run forever? Let's go through the steps for using OPT to find out what the job is doing.

Issue a "SHOWJOB" command and verify the job name and number. In this case job #72 named INALOOP run by user KEITH in account BOWERS. A portion of the typical output is shown in Figure 1. RUN OPT.PUB.SYS. The first screen will be global CPU and MEMORY data. See Figure 2. Observe the cumulative CPU states display at the center of the screen. Note the percentage BUSY (B), overhead (O), pause for disk (P), and idle (I); this information will be needed later. Now enter the command "#P", this will put you into program context. The screen will be similar to Figure 3. Locate the job name and determine its PIN (process identification number - the number by which MPE keeps track of each process). In our example a program named LOOPERP. DEVELOP.BOWERS is running under KEITH.BOWERS as PIN 43. If the job name does not appear in the present display, don't give up yet. The job may be in a system process or CI (command intrepter). Enter the command "A". This will list all active processes in the computer (see Figure 4). A by-product of this listing will be the display of the last command processed by the CI. One case in which we found a job in the CI was a DSLINE command to a remote machine which was in the process of a LEVEL 0. The job hung waiting for the remote machine to respond and OPT pointed to the problem. If the job still can't be found, just give up and live with it until you can cool start the system...we have yet to see MPE get this totally confused.

Now that the PIN is known, the next step is to enter the command "P"; in response to the request, enter the PIN found above(43). This will put you into user context shown in Figure 5.

At this point we can find out if the program is "hung", get some idea about where, and maybe even the reason.

Start with the upper right corner of the display which contains the Status Flags. If one of the flags is displayed in inverse video it indicates that the job is waiting for that reason. The most often seen reasons are: IMP - impeded waiting for some system resource, usually related to waiting for locks on an IMAGE database; SIR - waiting for a system internal resource, possibly LOADER or DIRECTORY (type "S"...any locked SIRs will be displayed. The SIR name can be identified with the aid of an MPE System Tables manual); MEM - the process has been swapped out to disc and is waiting for memory space to get back; BIO or I/O - check for a pending REPLY. Don't forget...you can enter a colon ":" and issue SHOWJOB, RECALL, etc. from within OPT. If you find a System Process in MSG (message) wait and the related subsystem hung the only reasonable way to free the process is almost always a COOLSTART. This kind of hang seems to show up most often in the datacom products.

The next place to look is CPU TIME at the top center of the display. Watch this field for a few minutes. If it doesn't change, the job may be "hung", or higher priority processes may be using up the entire CPU and leaving no time slices for this job. Remember that a job eventually settles to the bottom of its queue since it usually doesn't post a terminal read. Just to be on the safe side don't forget to verify that the job status from SHOWJOB was "EXEC" (some helpful individual may have suspended the job without telling you).

To determine if the CPU is too busy requires a bit of judgment and experience. We will discuss a means of collecting historical data later in this paper. Recall how busy the CPU was in the when you entered OPT. If you saw any idle time, especially in the cumulative area, the job is "hung" not just left out. Refer to the present display and note if any jobs with lower priority are getting time, if they are not, then your job has worked down to a priority too low for current system load. At this point there are two choices: (1) reduce system load by suspending higher priority users of system resources i.e. BREAKJOB #J46, reducing the number of sessions, or (2) use the TUNE command to alter the priority of the queue in which your job is executing. You can TUNE the "D" queue so that it heavily overlaps the "C" queue, i.e. TUNE 1000;DQ=152,160. Don't forget to re-TUNE the "D" queue back to normal after the job finishes (TUNE 1000;DQ=190,238... we keep the "C" and "D" queues slightly overlapped so that jobs can be introduced, suspended, or aborted even if "C" queue processes are consuming the entire CPU). Be careful...you can have a VERY bad effect on system response time, but if sessions are consuming the entire CPU you may not have any other choice beyond waiting until everyone goes home.

If the job is running (CPU time increasing) we must dig deeper. Note that the files the program has open are displayed below the status displays. LISTF <filename>,2 or DBUTIL or QUERY can be used to determine if file sizes are changing. This may help to determine if

the program is still doing useful work. At this point a knowledge of the function of the executing program is necessary.

If you aren't familiar with Stack Markers and how programs execute under MPE it might be best to skip this paragraph; also, take it with a grain of salt as MPE V/E seems to occasionally change stack markers on the fly (remember GETINFO?). For an idea of where the program is executing enter "M" to activate Stack Marker display. See Figure 6. Several successive cycles of display should be examined to determine if the program is in a loop (the same addresses showing up in the top few lines of the Stack Marker). If system calls are involved the name(s) of the modules are listed, otherwise the user segment must be identified by the program code addresses. Compile and PREP/PMAP listings of the program are necessary at this point to determine what the program is doing.

Compile and PREP listings for our dummy program are shown in Figure 7. The mainline has a couple of assignment statements to generate code, followed by a call to procedure FIND'THE'LOOP which has a couple more assignment statements to generate some more code, and then a call to procedure ITS'A'LOOP. The loop procedure has another assignment statement followed by a do while loop whose exit condition can never be satisfied, thus an infinite loop. Stack markers are built from the bottom up. The first stack marker above MORGUE (where all processes end up) has an address of %14 which is the procedure FIND'THE'LOOP in source line 18. The next address on the stack is %20. Refer back to the PMAP where the next lower code address is %14 (procedure ITS'A'LOOP). Subtracting %14 from the address at the top of the stack markers gives a difference of %4. The five-digit numbers in the second column of the compiler listing represent the number of words of code that have been generated by the compiler for each instruction in the current procedure(delta-P). If we look at the code count for ITS'A'LOOP we find that %4 falls between lines 14 and 15 indicating that the program was executing in the while loop when this sample was taken.

At this point you should have enough information to decide what to do with your long-running job.

The type of information presented above can be of great value in finding the solution to many day-to-day problems. We have found that as we learn more about MPE and the HP3000 in general the detailed capabilities of OPT become more and more useful.

This concludes our discussion of detailed data analysis with OPT; now we move to the other extreme of collecting and using data collected over days and weeks.

LONG TERM DATA COLLECTION:

A serious problem that arises when using OPT to analyze system behavior is the lack of a baseline. Without knowing what is normal,

it is rather difficult to identify the abnormal. If you are a real packrat OPT can be run continuously with regular printouts archived for future reference.

OPT logfiles can be collected over a long period of time using a stream that runs continuously. We have a job (appropriately called "IGNORME") that does the following: (1) appends the previous day's logfile to a cumulative logfile, (2) runs OPT for 24 hours with a sample interval of 1 hour, (3) re-streams itself, (4) !SET STDLIST=DELETE. This job runs on all systems at all times (even during a Level-0 unless a RELOAD is planned to immediately follow). The IGNORME job from one of our systems is shown in Figure 8. A typical run of 23.5 hours consumes about 90 seconds of CPU time so system loading for data collection is insignificant.

The cumulative logfile should be built wide enough to accommodate some system growth since adding peripherals extends the record width. Make the number of records large enough to hold several months of data if you have adequate disk space. For example a n 8 Meg., 3-bay Series 68 configured with 3-7914s, 7-7933s, 2-7976s, and 2-2680s produces a logfile record length of 442 words. The cumulative logfile for this system is built 512 words wide and can contain 10000 records. The file presently contains about six months of data (5004 records) and occupies 20,480 sectors.

A logfile record consists of a header record, a record for each data collection interval, and a summary record. This file doesn't grow very fast at twenty-six records per day.

We must be sure to remember that OPT interprets CPU BUSY as user activity only. Total CPU busy is equal to CPU BUSY + GARBAGE COLLECTION + ICS OVERHEAD.

A program to format some of the data from this cumulative logfile will be submitted to the swap tape. As of this writing the program allows selecting date and time windows, as well as excluding weekends. Data is output to two files: CPUDATA which contains all selected samples of fixed global data and CPUDAY with this same data accumulated on a daily basis.

Plotted data from OPT logfiles can be used to track system resource utilization. A few examples of this data are shown in Figure 9 through Figure 14. Data of the type shown in Figure 10 is useful for determining when a system is reaching CPU capacity while data of the type shown in Figure 11 is helpful when planning how to even out system loading. The information in Figure 12 gives a detailed breakdown of the data in Figure 11. Figures 13 and 14 show global disc utilization of long and short term timebases. The logfiles also contain data that allows monitoring of disc, tape, and printer utilization to the level of each individual logical device.

We hope that this paper will encourage others to develop additional tools to make use of OPT logfiles. The data that can be extracted from these logfiles can help us to make better use of our

computers as well as providing an invaluable tool when justifying the acquisition of new equipment.

Figure 1

```
SHOWJOB JOB=@J
```

JOBNUM	STATE	IPRI	JIN	JLIST	INTRODUCED	JOB NAME
#J2	EXEC		10S	LASER	THU 1:32P	IGNORME,KEITH.BOWERS
#J72	EXEC		10S	LP	THU 7:23P	INALOOP,KEITH.BOWERS
#J38	EXEC		10S	LP	THU 5:30P	JPMS2050,BATCHPMS.PMS
#J8	EXEC		10S	LP	THU 1:40P	MAILROOM,MAILROOM.HPOFFICE

```
4 JOBS (DISPLAYED):
```

```
0 INTRO
```

```
0 WAIT; INCL 0 DEFERRED
```

```
4 EXEC; INCL 0 SESSIONS
```

```
0 SUSP
```

```
JOBFENCE= 0; JLIMIT= 7; SLIMIT= 150
```

Figure 1.

Results of a SHOWJOB command. Note that our example job is #J49

Figure 2

HP32238A.00.19 OPT/3000 GLOBAL RESOURCE USAGE REPORT

ACTIVITY IN CURRENT INTERVAL

29.5 SECONDS

MEMORY USAGE	M	10	20	30	40	50	60	70	80	90	100%
		C	S	D			X				
CPU STATE						B					0
DISC I/O ACTIVITY		10	20	30	40	50	60	70	80	90	100 per second (0/0/0)

ACTIVITY OVER ALL INTERVALS

5.3 MINUTES

CPU STATE		10	20	30	40	50	60	70	80	90	100%
						B					0
DISC I/O ACTIVITY		10	20	30	40	50	60	70	80	90	100 per second (0/0/0)

MEMORY USAGE LEGEND:	CPU STATE LEGEND:	DISC ACTIVITY LEGEND
M Resident MPE	B CPU Busy	X Cached I/O
C Code segments	P Pause for I/O	S Segment Manager
S Stacks	I Idle	U User I/O
D Data segments	O Overhead (ICS & MM alloc)	
C Cached Disc Domains		

ENTER CHARACTER FOR CONTROL OPERATION COMMAND P

Figure 2.

First screen of OPT run. The main item of interest on this screen is CPU STATE in ACTIVITY OVER ALL INTERVALS. Note that the CPU is almost totally busy with only a few percent overhead.

Figure 5

```

PIN: 43    LOOPER.DEVELOP.BOWERS    USER:    KEITH.BOWERS    (J 72) 7:27 PM
-----
      STACK INFORMATION
DST: 210*  SYSOV:  430 14.2%
SIZE: 3020  DL-DB:   88  2.9%
MAXDATA: 3782  DB-Q1:   4  .1%
MAX Z-DL: 2590  Q1-Q:   12  .4%
              Q-S:    0  .0%
              S-Z: 2486 82.3%
CPU TIME: 83279MS
PRIORITY: 222
CAP:SM AM OP AL GL
DI CV UV ND SF PM
PH DS MR IA BA
STATUS FLAGS:
MRNG GRIN LRIN MAIL
BIO  I/O UCOP JUNK
TIME MSG  SON  FA
IMP  SIR TOUT  MEM
-----
XDS USAGE    SON
DST#  SIZE    PROCESS  OPEN FILES
-----
                $STDIN  $STDLIST

```

Figure 5.

USERS screen showing detailed information about our demo program. Note that this is the only screen in OPT that combines user, program name, job number, and PIN.

Figure 6

PIN: 43 LOOPER.DEVELOP.BOWERS USER: KEITH.BOWERS (J 72) 7:27 PM

STACK INFORMATION				CPU TIME: 83279MS	STATUS FLAGS:			
DST: 210*	SYSOV: 430	14.2%		PRIORITY: 222	MRNG	GRIN	LRIM	MAIL
SIZE: 3020	DL-DB: 88	2.9%			BIO	1/O	UCOP	JUNK
MAXDATA: 3782	DB-Q1: 4	.1%		CAP:SM AM OP AL GL	TIME	MSG	SON	FA
MAX Z-DL: 2590	Q1-Q: 12	.4%		DI CV UV ND SF PM	IMP	SIR	TOUT	MEM
	Q-S: 0	.0%		PH DS MR IA BA				
	S-Z: 2486	82.3%						

XDS USAGE	SON	OPEN FILES	
DST#	SIZE	PROCESS	
			\$STDIN \$STDLIST

STACK MARKER INFORMATION:

ADDRESS	Q-7	Q-6	Q-5	Q-4	X	DELTAP	STATUS	DELTAQ	SEGMENT-NAME
---------	-----	-----	-----	-----	---	--------	--------	--------	--------------

000020						000000	000013	060001	000004 (user segment)
000014						000000	000005	060001	000004 (user segment)
000010	000103	000000	000000	000000		000000	040000	140030	000004 MORGUE

SIR INFORMATION:

SIR NAME	HOLDER	IMPEDED
(no locked sirs)	PIN	PIN(S)

Figure 6.

USERS display with Stack Marker and SIR displays activated. This is one of the more detailed displays of which OPT is capable. Our demo program has called two procedures. There are no locked SIRs.

Figure 7

:SPL LOOPER

PAGE 0001 HEWLETT-PACKARD 32100A.08.04 SPL[4W] MON, JUN 10, 1985, 8:55 PM

```

1      00000 0      $control uslimit
2      00000 0      BEGIN
3      00000 1
4      00000 1      << This program LOOPS FOREVER >>
5      00000 1
6      00000 1      INTEGER LOOP'COUNTER := 0; << SET UP A COUPLE OF COUNTERS >>
7      00000 1
8      00000 1      INTEGER LOOP'COUNTER'1 := 0;
9      00000 1
10     00000 1      PROCEDURE ITS'A'LOOP;
11     00000 1          BEGIN
12     00000 2              LOOP'COUNTER'1 := 67;
13     00002 2
14     00002 2              WHILE LOOP'COUNTER'1 <> 99 DO
15     00005 2                  LOOP'COUNTER := 990;          << IT NEVER STOPS >>
16     00011 2                  END;
17     00000 1
18     00000 1      PROCEDURE FIND'THE'LOOP;          << PROCEDURE IN THE MIDDLE >>
19     00000 1          BEGIN
20     00000 2              LOOP'COUNTER'1 := 36;          << JUST GENERATE SOME CODE >>
21     00002 2              LOOP'COUNTER := 45;          << AND SOME MORE CODE >>
22     00004 2              ITS'A'LOOP;          << EXECUTE THE LOOP >>
23     00005 2          END;
24     00000 1
25     00000 1      << MAIN LINE OF CODE >>
26     00000 1
27     00000 1      LOOP'COUNTER := 54;          << SOME MORE CODE >>
28     00002 1      LOOP'COUNTER := 55;
29     00004 1
30     00004 1      FIND'THE'LOOP;          << START INTO THE LOOP >>
31     00005 1
32     00005 1      END.

```

```

PRIMARY DB STORAGE=%002;      SECONDARY DB STORAGE=%00000
NO. ERRORS=0000;              NO. WARNINGS=0000
PROCESSOR TIME=0:00:00;      ELAPSED TIME=0:00:02

```

:PREP \$OLDPASS,LOOPERP;PMAP

PROGRAM FILE LOOPERP.DEVELOP.BOWERS

```

SEG'          0
NAME          STT  CODE ENTRY SEG
OB'           1    0    0
TERMINATE'    4
FIND'THE'LOOP 2    6    6
ITS'A'LOOP    3   14   14
SEGMENT LENGTH          34
PRIMARY DB    2    INITIAL STACK 4704  CAPABILITY 600
SECONDARY DB  0    INITIAL DL    0    TOTAL CODE 34
TOTAL DB      2    MAXIMUM DATA  ?    TOTAL RECORDS 6
ELAPSED TIME  00:00:01.127
PROCESSOR TIME 00:00.460
END OF PREPARE

```

Figure 7.

Compiler and PREP output for "LOOPER".

Figure 8

```
!JOB IGNORME,KEITH.BOWERS,PERFMON;HIPRI
!CONTINUE
!RUN SUPRTOOL.PUB.ROBELLE
INPUT OPTLOGA
OUTPUT OPTLOGSA,APPEND
EXIT
!CONTINUE
!PURGE OPTLOGA
!FILE OPTREPT=$NULL
!RUN OPT.PUB.SYS
60
8
3
OPTLOGA
SYSTEM A-68
!CONTINUE
!SET STDLIST=DELETE
!CONTINUE
!STREAM IGNORME
!EOJ
```

Figure 8.

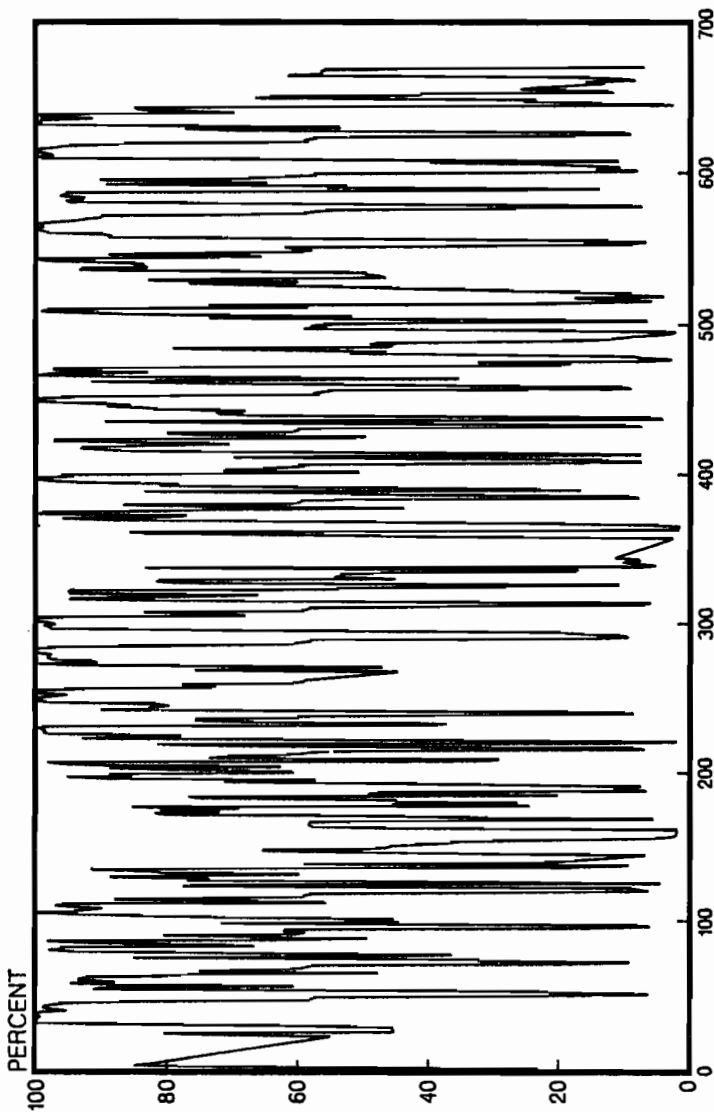
Stream job to collect OPT data into an accumulated logfile.

Figure 9

CPU UTILIZATION

HP SYSTEM - A FISCAL MAY, 1985

AVERAGE_CPU_BUSY



HOURS SINCE START OF MONTH
DATA COLLECTED BY OPT IN ONE HOUR SAMPLES

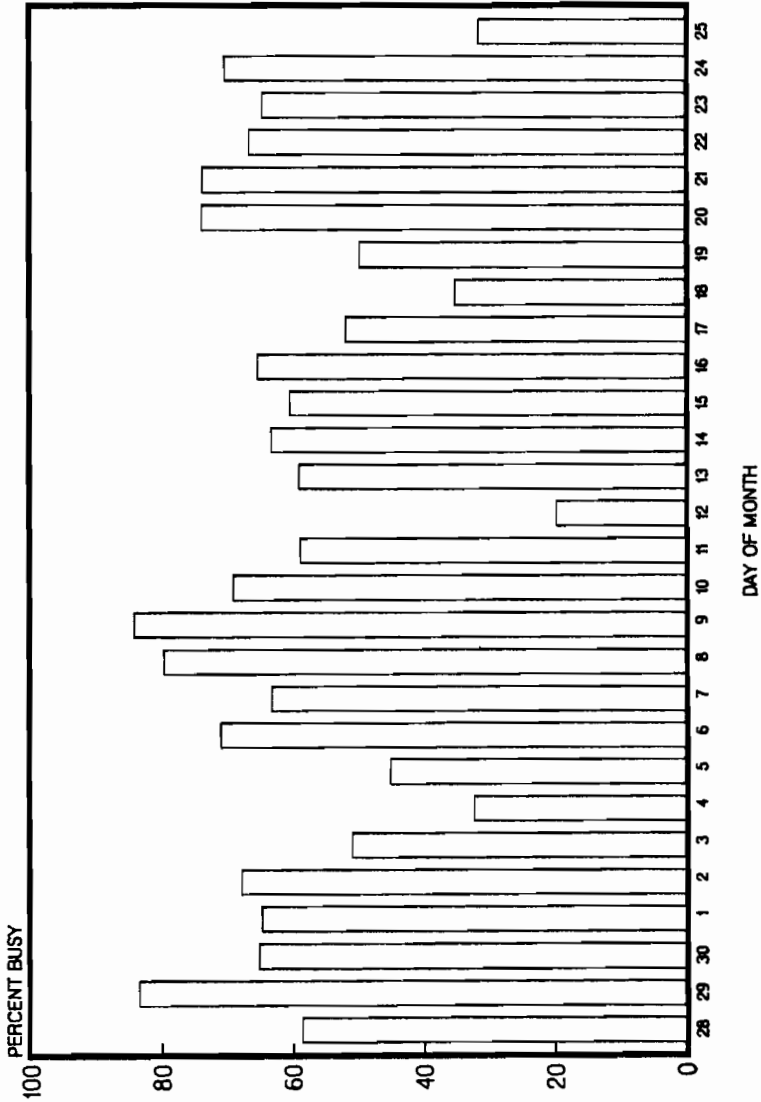
FIGURE 9.

Figure 10

CPU BUSY - 24 HOUR DAY

SYSTEM - A FISCAL MAY 1985

AVERAGE CPU BUSY



DATA COLLECTED BY OPT IN ONE HOUR SAMPLES

FIGURE 10.

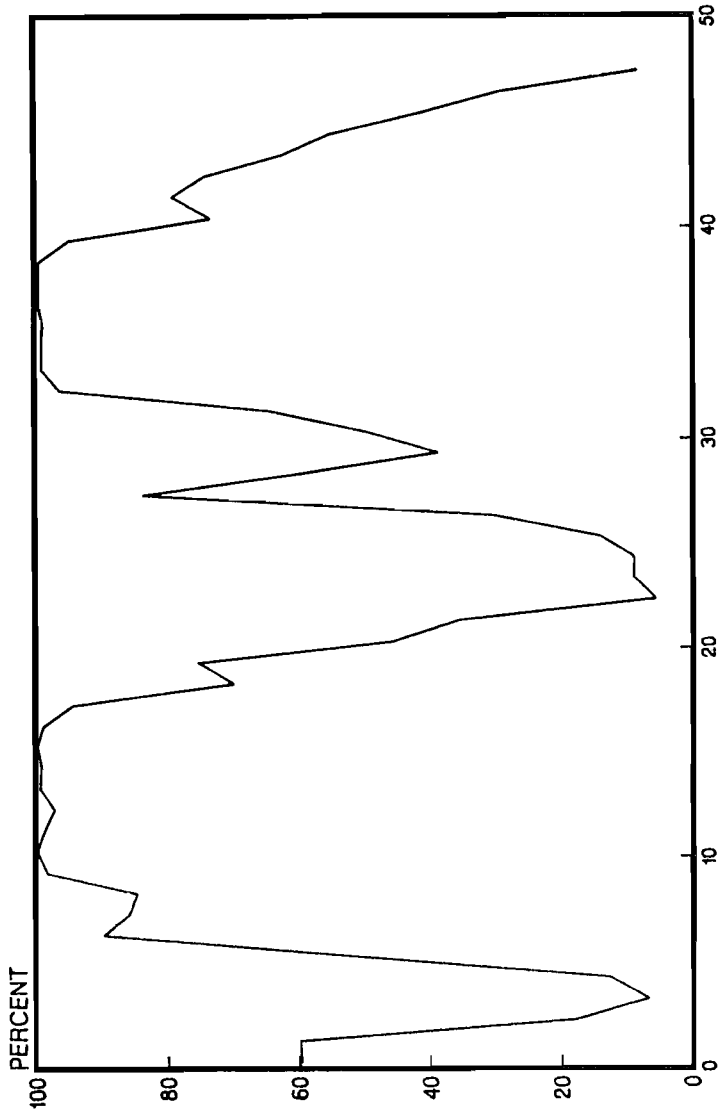
5/29/85 KAB

Figure 11

CPU UTILIZATION

HP SYSTEM - A JUNE 5/6, 1985

AVERAGE_CPU_BUSY



HOURS SINCE MIDNIGHT 6/5
DATA COLLECTED BY OPT IN ONE HOUR SAMPLES

FIGURE 11.

Figure 12

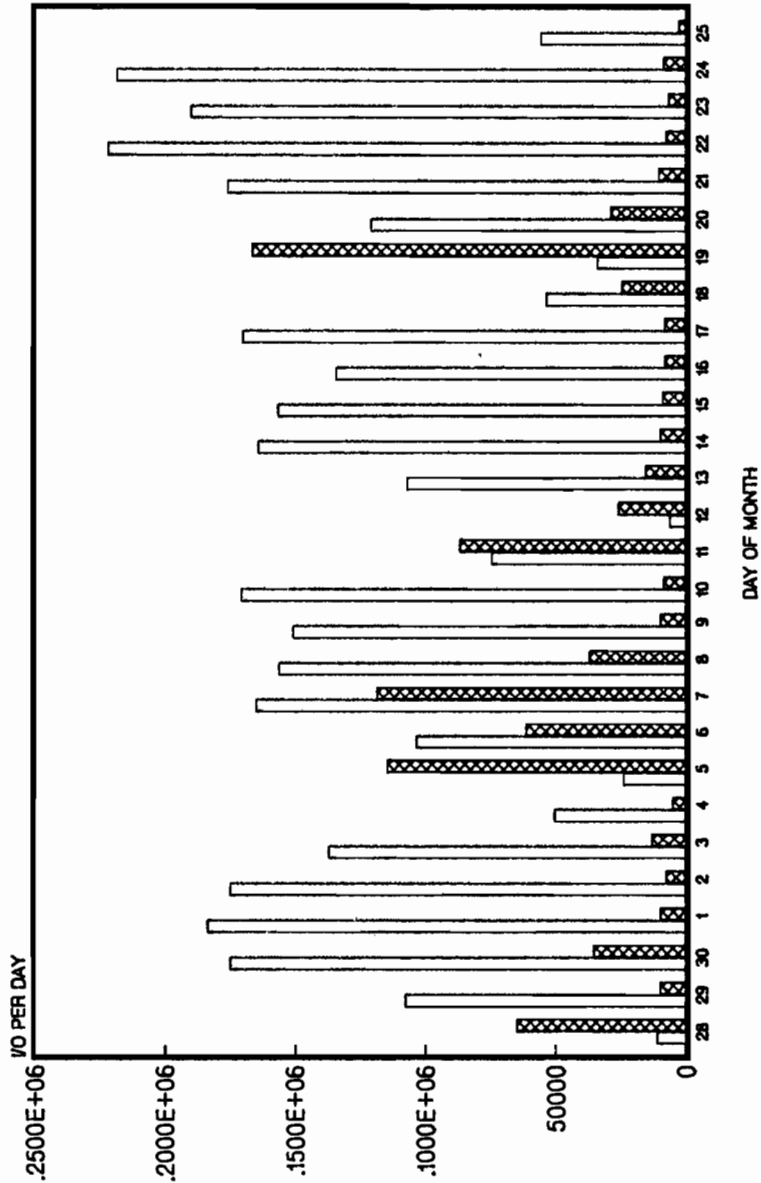
DISC IOs - 24 HOUR DAY

HP SYSTEM - A FISCAL MAY 1985

LOGICAL_IO



PHYSICAL_IO



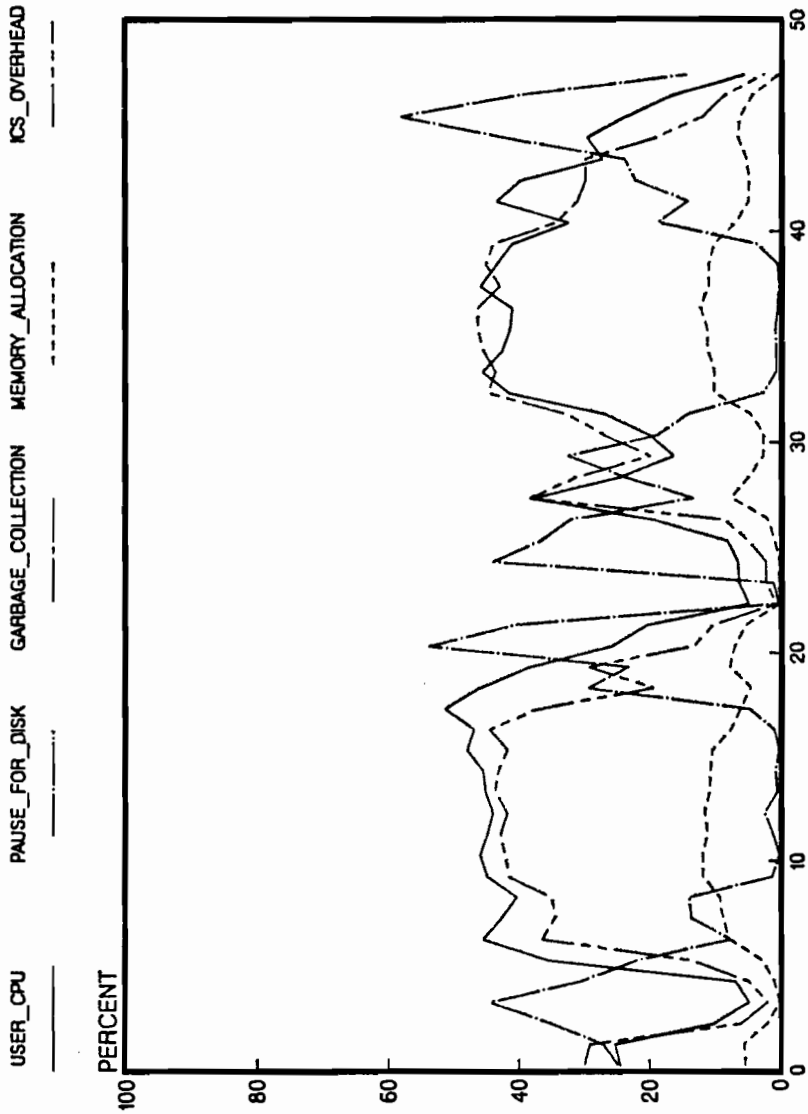
DATA COLLECTED IN ONE HOUR SAMPLES BY OPT

FIGURE 12

Figure 13

DETAILED CPU UTILIZATION

HP SYSTEM - A JUNE 5/6, 1985



HOURS SINCE MIDNIGHT 6/5
DATA COLLECTED BY OPT IN ONE HOUR SAMPLES

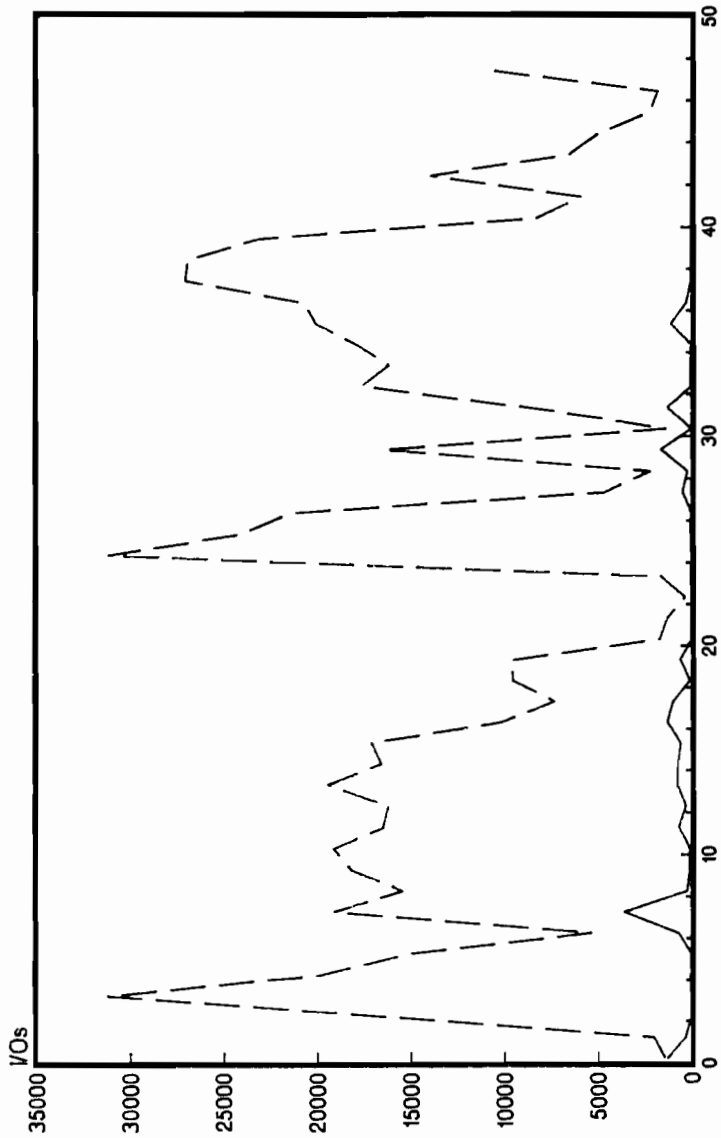
FIGURE 13.

Figure 14

DISC UTILIZATION

HP SYSTEM - A JUNE 5/6, 1985

LOGICAL_IO _____
PHYSICAL_IO - - - - -



HOURS SINCE START OF DATA
DATA COLLECTED BY OPT IN ONE HOUR SAMPLES

FIGURE 14.

3019. TRAINING AND SUPPORTING OFFICE SYSTEMS USERS

Gail Gross
Hewlett-Packard Company
19055 Pruneridge Ave.
Cupertino, California 95014

My charter is to promote the use within HP of released office products, and my department's activity is related to training and supporting users of HP3000, HP Touchscreen and The Portable software/peripherals.

There are five trainers in my department and one administrative support person. Our customer base is the approximately 2,500 people on Cupertino, CA, site. We offer 20+ individual product-specific courses covering Series 37 operation/management, word processing, presentation and business graphics, spreadsheets, list creation and maintenance, and basic PC/HP3000 file transfer.

What is an Office System...

This can get a little sticky...after all, programmers use COBOL and BASIC and designers use CAD/CAM/CAE equipment in offices, so do COBOL, BASIC and CAD/CAM/CAE qualify as office systems? I can assure you that you will have your hands too full training and supporting word processing, graphics and spreadsheet users to even think about trying to support other products.

...and Who Uses It?

An MIS department supplies computerized tools for specific purposes for use by limited numbers of people. For instance, programming languages for programmers, financial and payroll programs for accounting, inventory control for manufacturing, etc. Office automation products, however, have the potential to reach EVERY PERSON in your organization regardless of their job or department...including those who have no previous computer experience. Your company has and will continue to invest a lot of money in hardware, software and support services for OA; too late you realize the number of people who need to be trained NOW on a wide variety of products in order to effectively utilize this technology and maximize your company's investment, not to mention the ongoing training that will be required as new software/hardware products and staff are brought into the organization. Clearly, training and support are critical success factors in most OA installations.

Why Can't They Train Themselves?

Less than ten per cent the people in any organization will take the time to overcome frustration and adversity to teach themselves, with or without self-paced, computer aided or videotaped training vendors provide for office products. The

other 90% of your OA users, with their new terminals or PCs on their desks, will look up at you with shining faces and eager expressions. "Tell me how to start," they plead. If you don't help them, they will push the equipment aside and do their jobs as they have always done them...without the high-tech tools; and unused equipment is much more costly to your company than equipment that's in use.

When I first started HPWORD training, every divisional secretary had a 2626W terminal, a 2601A printer and a box of self-paced HPWORD training on which the cellophane wrapping remained intact. Their justification was that they're interrupted too often at their desks to concentrate on using self-paced guides. Other divisions at HP set up Learning Centers in quiet areas with all required software, hardware, self-paced and videotaped material and reference guides. These are fairly well attended for technical training of professionals but not for office product training. The justification given for this lack of attendance is that their jobs require them to be at their desks so they can't go to a Learning Center. Those are the same people who will be first in line to leave their desks to attend your class. These are the same people who won't find it ironic that you as the instructor are using the same self-paced instruction material in class they wouldn't use on their own. The simple fact is, people want their hands held while they're learning something new. But if you attempt to train them one-on-one, one at a time, your time will not be effectively utilized.

HOW TO DETERMINE THE NEED FOR OA TRAINING

The Informal Survey

Expensively unused equipment...PCs, terminals, printers and plotters pushed into corners or on the floor with the power cables rolled up on top; typewriters still the obvious preferred tool on secretaries' desks; manuals and self-paced kits still wrapped in cellophane; documents with obvious erasures or "white out" on them; overhead transparencies with typewriter-produced text instead of plotter-produced overheads in full color.

The Formal Survey

To find out exactly how many people require what kinds of training, distribute a modest list of classes available with descriptions of the software products and a sign-up matrix. You can ask them to prioritize their requests based on job requirements. You'll find you need to provide very basic level introduction classes to the HP3000 and PCs for at least 50% of your organization. For example, our Introduction to the HP Touchscreen is a pre-requisite for all HP Touchscreen product-specific classes; WordStar class is not the place to learn how to boot the system and format diskettes. Your courses don't have to be completely finished before you advertise; you can clearly state on the cover memo that you are using the matrix

to assess the need over, say, the next six months; and you can state a given course will be available next month or the month after...they'll still sign up. By the way, even if your end users have taught themselves and been using the product, they will still sign up for class because they realize there's much more they can learn.

Be sure to specify a due date for the return of the matrices, otherwise they'll trickle in forever and won't be helpful at all in the planning process. Expect bootlegged copies of your matrix to be returned by people other than the ones to whom you sent it to begin with...something as exciting as product training will be passed around.

Even with the matrices, no one I know at HP has ever overestimated the OA training requirements of their customers. EVER. The volume of the response will astonish you.

HOW TO DEVELOP COURSES TAILORED TO YOUR CUSTOMERS' NEEDS

Trainer, Know Thy Customers!

Because OA tools reach virtually every person in an organization, your training will have to meet the demands of technical professionals, clerical and administrative staff, production workers as well as managers and executives.

You need to understand what your users do, how they've been doing it, how the new tools can help them do it faster and easier and how the tools can potentially change the way they perform their work. With word processing, for instance, secretaries are not going to be creating more new documents; they will spend most of their time editing existing documents. A manager who would hesitate asking a secretary to completely retype a document on a typewriter to make just a few changes will have no qualms about asking him/her to re-edit a word processed document. And with the proliferation of PCs in the office environment, more and more professionals and managers will start generating their own documents, thus creating a fundamental change in the way your customers perform their work based on the introduction, training and support you provide on office products.

Course Development Tips

First things first...think up a catchy name for your service such as MyCompanyName Computer College or High Tech High, something they'll remember and that sounds appealing, energetic and exciting.

HANDS-ON TRAINING IS ABSOLUTELY CRITICAL. You must have space and equipment even if it's borrowed for the duration of the class. You can't lecture to your students about a word processor or graphics package and expect them to be able to use it when

they return to their desks. THEY MUST PRACTICE ON THE TERMINAL OR PC, PRINTER OR PLOTTER.

Go through any available training material yourself and jot down questions you have as they occur to you. Use the product a lot to familiarize yourself with how it works, what happens when you press the right key at the wrong time, ideas on how to use it to support your company's applications. Use it, use it, use it. If you have other people testing or actively using the product, meet with them to get their ideas on better ways to apply it. Just remember that no matter how well you know a product or how often or long you've used it, users will still ask questions you never dreamed of, can find ways to use it you never thought of and will trick it into supporting applications you wouldn't have thought it was appropriate for. The better prepared you are, the less often you'll have to say, "I don't know." (Of course, "Let's try it and find out" is a great escape mechanism!)

Integrate your own experiences with the product...both good and bad...into your material. "Here's how I use this feature" and holding up and passing around an example is a really good teaching tool. If you made foolish mistakes, tell them about it so they don't feel so foolish when they make them. Tell them, or better yet, develop lab exercises to show them, how to recover when mistakes are made. This will give them confidence that they can recover from a mistake when they're on their own.

Once you know the product, deliberately push the wrong keys to see what happens and how to recover from all potential disaster situations...students will do the most amazing things in class. Don't assume they know more than to push the ESC key or the BREAK key. Figure out how to recover from these kinds of situations prior to the actual class so you don't waste valuable class time and lose their attention.

Develop lab exercises so they can practice, practice, practice. They can't practice too much...after all, don't Larry Bird and Joe Montana keep practicing?!

Develop several quizzes - either verbal or written - to reinforce the key points made throughout the class (vs. one big final exam at the end of class). Don't give them trick questions or set them up to fail...the idea is not to demonstrate how much more you know than they do but to prove to them how much they've learned and how much product competency they've acquired in such a short time.

If you're breaking up the course into two parts separated by a period of time, give them homework after Part I and start Part II with a quiz to refresh their memory. They won't remember much more than 50% from Part I, even if it was the previous day.

Set up alpha classes to test and time your training materials. Students will take much longer to perform a given task than you

do. Make their attendance in the alpha class dependent upon their filling out a detailed course evaluation. Use the feedback to improve the lab exercises, presentation or workbook. Jot down questions they have as the class progresses...do you need to rearrange the order of presentation and exercises, do you need to improve the exercises and quizzes or add more, etc. If you know your users personally, invite some speedy, some average and some slower learners to be your students. You'll then have a much better idea of how the wide range of your users' sophistication levels will respond to your training. You'll rarely be declined when you personally invite someone to participate in an alpha test of a training class; they're really flattered to be chosen.

Believe it or not, students' SELF-CONFIDENCE in their own ability to successfully use and apply the product in their daily work environments is more important than basic push-this-key competency. Design your class so that they successfully print a document or plot a drawing or chart within the first half-hour of class. It will be a very basic image or memo; the idea is to show them how easily and quickly they can successfully obtain finished output from the product. Approach each successive level of complexity in the same fashion to prevent their being overwhelmed and intimidated.

Try to make the course fun. If their body language is open, they are relaxed and smiling, they're more receptive to your material. Have funny anecdotes or trivia questions available for them to use in HPDeskManager class as messages; let them create a birthday card for someone special or a Mother's Day card in HPDRAW, DIAGRAPH or Drawing Gallery class. We have a special HPDeskManager training node with movie star names---for half a day they get to fantasize they're Humphrey Bogart or Marilyn Monroe. I'm not advocating turning the class into a party, but if they're having fun they're paying attention and are receptive to the information you're presenting...they learn. And they'll come back to another of your classes because it's worth their time investment.

Lab exercises don't have to be complex to be effective...be inventive and create fun lab exercises. For example, have them create a traffic signal in HPDRAW class...they learn object placement, pen color selection, text entry, editing and placement, font selection and sizing for titles in one exercise. Try to create fail-proof labs. The resulting memo, list, drawing or chart may be funny looking, but they didn't fail to create it.

And be sure the lab teaches something; I know that sounds incredibly obvious, but just because they're pushing keys doesn't mean they're acquiring any knowledge they can transfer to their jobs. Ask yourself when you're testing the exercise, "What skills or product knowledge are they going to acquire from this?"

If you are teaching them chart and graph creation on PICTURE PERFECT, Charting Gallery, HPEasyChart or DSG/3000, it isn't

enough to teach them which keys to push to create pie charts, line and bar graphs. Most people (regardless of job description or title) know very little about presenting data in a graphic form, or even when to use what chart type. Show them how their data forces the chart type; give them a verbal quiz to be sure they understand. Use the excellent graphs in TIME magazine or USA TODAY newspaper as examples. Show them how to use the colors on the plotters effectively; textures, too. Have examples so they can compare poorly planned graphs to good graphs. A Bit of Wisdom Learned From Experience: For lab exercises, give them the information they way they'll get it on their jobs...handwritten (scribbled) titles, hand-drawn "ideas" of the graph their manager wants plus his/her notes, circles and arrows. Because it's a real-world exercise, it gives them a better opportunity to be successful using the product in their daily work.

If manuals are not available, create student workbooks they can use as reference guides after class. Many people like to write notes as you lecture, and a workbook is an excellent place for this. Make it an example of "OA in Action"--combine your text and graphics on the laser printer and tell them how you did it. Our student workbooks for the majority of our classes are produced with TDP/3000 or HPWORD and HPDRAW figure files and printed individually on the 2680A for each student. Each plotted overhead transparency shown on the screen is also presented on the top of the corresponding workbook page with room for notes underneath. The facing printed page has important tips about the feature illustrated. For some reason, students hate to look things up in the "real" reference guides; they will, however, look things up in their class workbook, so keep in mind the fact that this may become the only reference guide they'll use when you're creating it. Pretend you're a slow learner and read through your workbook carefully. Deliberately try to misunderstand the material...you'll probably find you've based some of your statements on assumptions or that the students might easily misunderstand them.

Teach them to work smarter, not harder. A common mistake people make when developing internal training courses is to concentrate on which keys to push and demonstrating the features. Most users regardless of job title aren't able to make the connection between product feature and how to use the feature to make their work easier and faster. In word processing, for example, showing students how search and replace works doesn't provide them with much. Instead, tell them why and when to use the search and replace feature to save time. If they have to type "ABC CORPORATION" even two or three times in a document, search and replace can save them 28 or more keystrokes if they type an "&", then search for and replace the "&" with "ABC CORPORATION" when they've finished entering text. Now you're providing them with information they can take advantage of to effectively utilize the technology. Another good technique is to ask for ideas of other ways people could use this feature to save time. You'll be surprised at how creative your students can be if the learning

environment is supportive and open. The added benefit of this technique is that because they thought of it themselves, they'll remember it.

HOW TO REGISTER AND SCHEDULE STUDENTS

Registration/Training Record Keeping

You have your first course(s) nearly ready, and you've advertised with the course description list and sign-up matrix. You can't imagine the time you'll waste if you restrict your record keeping to the completed matrices. Soon they will be covered with yellow "Post It" notes and scribbled notations in the margins. Plus, you need to keep accurate records of your completed training activity as well as the wait list. That's too much paper to work with, and I can tell you from experience it will drive you crazy trying to keep track of everything on paper. We were saved when HPListKeeper was released; now we use HPListKeeper to track student's name, manager's name, location code, building number, phone extension, division, course requested, date requested, charge for the class, and date taken. We currently have more than 4,000 entries in our list and provide divisional activity reports to our customers, reports to accounting to bill out our charges, and our monthly list for calling students for each course. If your activity is considerably less, Personal Card File/150 will provide similar benefits. Because HPListKeeper and Personal Card File/150 are easy to learn and use, you can hire temporary help to perform periodic maintenance of your data if you don't have dedicated administrative help.

Time Involvement

Once you've determined the volume and types of training needed, you will discover most of your time is spent not in course development, training or support but in trying to schedule students. You'll call twenty people, play telephone tag with ten just to get four scheduled for class, and then two of those will need to reschedule in the week preceding class. You will need to get either full time or volunteer administrative help; it's even more important to acquire administrative assistance first than it is to add technical staff because your time and any additional trainer's time will be spent on clerical activities instead of training and supporting end users. You need to be free of the time involvement that scheduling students requires in order to accurately assess your need for additional technical staff.

No-Shows

The biggest problem you'll face in training--other than a large training backlog--is no-shows. They feel their reason is valid for not appearing as scheduled; the effect on your backlog will be devastating. If you have only four chairs and two people are no shows, your training effectiveness is halved and bringing down your backlog of waiting students is hindered. And the numbers of

actual students trained may make it appear to your management that your efforts are having limited success.

Our first effort to prevent no shows was to send a paper reminder to each student telling them the date, time and place of the class. That didn't help much because usually the problem of no shows was initiated by students' managers who had crises prior to class. So we sent a paper notice of the student's registration to each manager as well. That helped somewhat, but it wasn't until we instituted a \$250 no-show charge to the managers' location codes that the problem was almost totally eliminated. We rarely have a no-show now. However, you need to obtain the billing information when you first schedule them into class; it's nearly impossible to get it after the fact when they know you're going to charge them. And make it a large no-show charge. We state the "less than 24 hour notice" rule and the no-show charge amount clearly on both the students' and the managers' notice forms.

HOW TO BALANCE STUDENT EXPERTISE LEVELS IN CLASS

Honest! Many Don't Know How to Type

Many of your users will resist taking advantage of the new technology simply because they don't know how to type. Even students who understand the concepts of the software will dramatically slow your class searching for keys during lab exercises. You can try offering a PC-based typing tutor for those people who lack basic typing skills prior to their coming to class. This can also be helpful to those who can type on a typewriter but are intimidated by a terminal/PC keyboard. Or you can have an Intro to the PC or Intro to the HP3000 or Intro to HPDeskManager class specifically for those who can't type. You need to encourage them to learn to type either at a local college after work or on a PC-based typing tutor...have class names and phone numbers available for them to sign up or offer to loan them a PC-based typing tutor because at some point their continued lack of typing skills will present them with major hurdles in taking advantage of the various software programs available. You need to support their learning this basic skill; it's the foundation of the total skill set people need to be successful using OA products.

Basic Computer Literacy Training

If you are introducing OA products to computer-naive students for the first time in a class situation, you will probably have to spend considerable time overcoming resistance in some people, downright fear in others. Lack of basic computer literacy provides you with an opportunity to enhance your contribution to your users. As a matter of fact, we developed a "Surviving and Thriving in the Electronic Office Workshop" in order to provide some basic understanding of the HP3000 environment to our users...group and account structure, OA product recovery files,

FCOPY, and assorted MPE commands. It was very successful, and our users really appreciated the opportunity to learn more about the computer. Many of your OA users will ultimately spend 75-80% of their day at their terminals/PCs, and having a deeper understanding how the computer works makes them less frustrated and much less of a support burden for you. We also developed an Account Manager workshop to relieve the operations department of the burden of managing all the HP3000 accounts. Department secretaries were designated to be the appropriate people to become account managers, and they were very excited to have the opportunity to learn more technical skills on the HP3000; we had very few accounts that didn't have a volunteer in the workshop. If you can find the time to provide the training, you'll never lack for students, no matter what the class covers.

Group by Relative Computer Sophistication

There are those few in every company regardless of job description who can grasp software applications much more quickly than others, so you want to schedule fast, medium and slow students with their peers. Your class will proceed only as fast as the slowest student. If you know each of your students and their relative expertise personally but aren't able to separate them into classes based on their relative speed in learning, try using "place cards" and seating fast students next to slow ones; the buddy system that develops in class can be very valuable to you and help keep your class on schedule.

Group by Job

Because your customer base crosses all functional and job description boundaries, balancing student levels in class is very important. A manager doesn't want his professionals or clerical staff to see him/her grapple with the new technology, or worse yet, fumble with the keyboard. So group managers with other managers, professionals with others, and administrative personnel with each other whenever possible. Administrative personnel are more comfortable with repetitive exercises; technicals and professionals like having technical explanations; and managers/executives want the training FAST.

HOW TO JUSTIFY ADDITIONAL STAFF, SPACE AND EQUIPMENT DEDICATED TO TRAINING AND SUPPORT

Initially you will most likely have some degree of difficulty justifying staff, space and equipment dedicated to training. For my very first classes, I had each secretary wheel her 2626W terminal on her office chair to my cubicle to learn HPWORD...it provided terminals as well as chairs for the class. As the demand grew, the logistics of lugging terminals around became more complex not to mention a general pain in the back. In order to justify the purchase of dedicated terminals for training, I "advertised" on the Cupertino site. I put out a memo describing the product training available along with a thumbnail description

of each product. I attached a matrix for each department to fill in with names, location codes, phone extensions, etc., with spaces for class requests. The mailroom could barely keep up with the load of completed matrices mailed back to me. I had totally underestimated the demand and need. Based on the numbers the matrices reflected, I was able to justify dedicated training equipment and an additional person. Ultimately, the backlog of training requests (plus the fact that I'm a zero-cost center because my expenses are billed back to my customers) helped me justify two dedicated training rooms (five HP Touchscreen IIs in one plus a 2601A, a 2602A, a LaserJet, a ThinkJet; eight HP Touchscreen IIs and a plotter in the other in addition to use of a substantial portion of the resources of an HP3000/64), and an HP3000/Series 37 plus eight terminals in a borrowed training room.

Or it may not be a simple question of getting approval for allocation of funds to acquire adequate resources, but instead be a more complex question top management has about perceived value to the company, cost of employee time away from the job when attending classes and long-term need/benefits. You can conduct management awareness sessions to inform them of the demand, project training and support requirements into the future, present a training summary listing the classes available with the goals and objectives clearly stated. You can conduct a survey to determine the value users and their managers attribute to your training and support and provide top management with the results. You can have satisfied customers give testimonials...supervisors and managers alike will be able to attest to their employees' if not their own increased productivity and morale and decreased frustration. This is a complex issue because value is often perceived vs. being easily measurable.

Write down every request for information and assistance and use the information and total numbers of assistance requests to leverage acquiring more resources. It takes time to write down the time of the request, the name of the person and the nature of the request, but the results at the end of a single day will astonish you and your management. Many of your users are so naive they don't think to check before they call you to see if their terminal or printer is plugged in. They don't understand that ports stay with the cable not the terminal when they move it to another location. They want information about training classes. They want information about what products to use. They want information and assistance in deciding what equipment and configurations to order. If you're the only person providing OA user support in your organization, you'll be on a dead run all day long and/or have a cauliflower ear from being on the phone so much. Because you trained them, your friendly, smiling face is the one they'll look for when they have problems. One benefit of this is that you can use what you learn from user support to improve your training material so you don't have to continue providing the same level of support for everyone you train.

Hiring Additional Staff

Having extensive technical expertise is not enough...can the person talk to computer-naive end users effectively? Can the person understand how your customers do their jobs and then bridge the feature/benefit gap? Being wonderfully supportive and patient is not enough; can the person acquire technical knowledge? After you've been solving user problems for a while, you'll think you know as much as any HP SE, so any staff you hire has to have the ability to assimilate technical information and apply it to solve sometimes poorly defined user problems. Being an expert user of the product is not enough; can the person actually teach others how s/he uses the product? Can the person control a classroom? Can the person talk effectively in front of people in essentially a presentation environment?

Recruiting Volunteers Among Your Users

Many divisions at HP recruit talented end users in each department to be the trainers and first-line problem solvers. In order for this strategy to be successful, however, you must have the highest level of management supporting it. After all, helping you isn't in a secretary's or professional's job description so his/her manager is not likely to appreciate and support his/her taking time away from the job. You need to decide exactly what these volunteers are going to do and what they need to know. For instance are they going to need to spend their volunteer time only training, or do you need them to get approval to invest lots of time developing the course they'll teach? It's harder to get management support for course development time. Are they going to train and support several products or just one? How much training do they require in order to be effective trainers and first-line problem solvers and will their managers permit their time investment?

Think up an exciting name for their "part-time positions" such as "OA Mentors" or "PC Pioneers", etc. It will become a status symbol to be one, and people will exert extra pressure on their managers to support their time involvement.

You will need to invest your time in providing these volunteers with additional training. Just because they're wonderful end users doesn't mean they know how to train others; they'll need to be trained how to train others. They will also need to have some technical training in order to solve user problems. And remember, volunteers will train others only on those features they understand or worse yet, provide misinformation about product features, so be sure your volunteers really, really know the product.

The biggest problem those divisions using this strategy have encountered is the trainer being pulled at the last moment because of a department priority or crisis. Remember, it's not their job to help you, and department crises take priority. Some

divisional OA coordinators have given up in frustration and sent their students to other divisions for training; others have identified several alternate or substitute teachers for each class.

HOW TO "SELL" YOUR SERVICES

After you've been training awhile and have kept accurate records of your activity, you need to decide if your services are going to be "free" or not. You may feel that training should be free, and I agree with you up to a point. I've discovered that there's no end to the demand for training and support. Especially with the introduction of PCs into the workplace and the associated software explosion, one begins to realize that part of every employee's week will be spent being trained; you can never offer enough classes to satisfy the demand. Please don't fool yourself into thinking that you can provide your users with one or two classes and be done with it for the rest of your life. OA training is not a one-time solution; it's an ongoing, constantly changing, evolving and improving process. By the time you finally get all your current HPDeskManager users trained, new people are hired and new releases of the product require updating of user skills...and that's just one product! Someone has to pay for the ongoing time involvement in course development, training, scheduling and post-training support.

If you're one person in a small- to medium-sized company, that probably won't be a problem. If you're one of many in a large company or site, you may have to compete for the same customers. You might try offering different classes than your peers offer, or restrict your training to your own department or division. Or group together to become a site, building or divisional training department...leverage your combined customer demand to justify space, staff and equipment vs. duplicating efforts.

How to Determine Class Costs

We target expenses for the next fiscal year; then we take the number of classes we can conduct based on the past year's activity multiplied by the number of students per class and divide the total number of students into the total cost to estimate the cost per student. We publish a course description handout with the charge for each class on it (as well as the notation that those divisions which are receiving flat allocations per month pay no per class charges).

Monthly Allocations

Some of our customers are divisions on the Cupertino site. It is less expensive for them to receive monthly allocations from us than it is for them to attempt to replicate our training and support resources and services. They can send an unlimited amount of students to class at "no charge" because the division is billed a flat amount each month. Even though we don't bill

the student location codes directly per class, we still use the location codes for these divisions in our HPLISTKeeper listings to show monthly training activity. Billing these location codes directly for no-shows is the exception.

Charges per Class

Another method we use is to charge students on a per class basis. We obtain their location code when we initially place them on the wait list. Those divisions not being charged a flat amount per month are billed per student on an class-taken basis. The Finance Department really prefers the monthly allocation method, but not all of our customers are willing to be billed a flat fee.

TRAINING TIPS

OA TRAINING GOLDEN RULE: Be prepared, be flexible, be patient and assume nothing.

It's better to cover less material well than it is to cover all possible material briefly. Your students are not computers with perfect storage and recall ability; they're people who must grapple with new terminology and techniques, and attempt to process and store the information so they can successfully recall and implement on their jobs. They and you have a better chance for success if you limit the processing they must perform in class.

Remember that adults learning new material will forget between 25 and 50% overnight or over a weekend...make it a requirement that they must have their equipment installed and operational at their desks in order to attend class. Otherwise they will have forgotten everything by the time their equipment arrives and will require retraining.

Have all materials ready for class the night before--log-on instructional cards, handouts, pencils, paper, student manuals, certificates, instructor's materials, etc. Organization and preparation convey your image as a professional in control. (Certificates of completion are important in building the self-confidence of new OA users. You'll be surprised how many people display them prominently on their walls.)

We've set up bright plastic crates, something like milk crates, for each course holding all necessary handouts, lab exercises, overhead transparencies, and instructor's manuals; this is especially helpful if you don't have a dedicated training room and have to set up from scratch prior to every class. If you have a training room and conduct a variety of courses and/or have volunteer instructors, the crates are also helpful in keeping your material separated but still easily transportable and accessible. The added benefit of crates is that they stack vertically, so space doesn't become a problem.

We've set up special groups and accounts for our HP3000 OA classes and stream jobs after each class to purge student-created files and restore files required for lab exercises so that one job cleans up after one class and at the same time prepares you for the next...techniques like this will save you a lot of time. We do not use UDCs to run programs for HP3000 classes so they leave class with the ability to run any HP3000 OA program wherever they might be. TEST EVERYTHING AND ASSUME NOTHING.

After you introduce yourself (if your organization is so large that many of the students don't know you or each other), have the students introduce themselves and give a brief summary of their job responsibilities. This helps the class gel as a unit and will foster bonding between speeders and slow students; this buddy system will be invaluable to you in class.

Keep class sizes small, if possible. The more students you have, the slower class will go, the fewer questions they will ask for fear of appearing stupid, and the more trouble they can get into because you aren't able to keep tabs on all of them. If you must conduct large classes, enlist a secretary who knows the product being taught to be a lab assistant roaming around the room and helping people. Students can be deeply in trouble before alarm is reflected on their faces; having a lab assistant peeking over their shoulders can head off problems before they become catastrophes you have to invest instructor time in resolving.

Learn to read your students' faces. If you see a lot of blank looks during class, keep presenting the same material different ways until they have that wonderful "AHA!" look. Glazed eyes in the afternoon mean (1) they need a break to move around or (2) they've had it. There's only so much new information people can absorb; if the course seems too ambitious, break it into two parts and let them practice what they've learned after the first part.

Don't speak "RS232" to them, and don't assume they already know how to turn the equipment on and off. Begin by teaching them where the on/off switches are, showing them where power cords and cables are and where they connect. Don't assume they know. If it's an HP3000 OA product, show them the REMOTE MODE star. Remember that terms such as cursor, colon prompt, function keys, and boot the system are familiar to you but not to them. Take them on a tour of the computer center; show them the laser and line printers, the computer, the disc drives and tape drives, printout bins, etc. Introduce them to the operations staff.

Use objects physically as teaching tools vs. an overhead drawing whenever possible; pass the objects around the room. For instance, open a 3-1/2" diskette and pass it around for them to look at in an HP Touchscreen or The Portable class. Amazingly, even technical professionals will examine it carefully...and adults remember better if they touch the object. Use real file

folders and in/out trays for HPDeskManager class to illustrate the IN TRAY, OUT TRAY and FILE CABINET environments.

Try to prevent long lunches; the last thing you need as an instructor is a student who had one too many Long Island iced teas at lunch.

Repeat, repeat, repeat, repeat, repeat, repeat.

Be prepared to do lots of handholding and rescue work.

Give sincere "attaboys" and "attagirls" when they finally master a concept that's been eluding them.

Students have a very low frustration threshold.

Students--especially managers and executives--have a very short attention span.

Define everything--do not assume they know how to correlate the box at the bottom of the screen with the associated function key.

They need to feel confident about your technical ability, knowledge and experience with the product; they need to feel comfortable enough with you as a person to ask questions and expose their ignorance.

They can't practice the tools enough. (Remember Larry Bird and Joe Montana!)

Use course evaluation forms to improve your class and learn to take negative feedback with the good...you can't please all the people all the time. But neither should you be devastated at bad evaluations...take them professionally, not personally. And I can assure you that you will receive unfair negative evaluations as well as ego-inflating smile sheets.

Be sincerely supportive of their efforts to master this new technology.

Be flexible.

Be nice; sometimes it's really hard to do.

Don't assume that when they tell you they've taught themselves how to use a product that they know what they're doing...users skip the sections in the self-paced material they don't understand or they ask their peers and proceed on a misconception of how the feature works.

If you have the time to do it, develop "advanced" level classes to provide users with the next level of expertise. For instance, after they've been using HPWORD for a while, offer an Advanced HPWORD class. After the initial class they will settle down to a

very narrow feature set...they don't even see the screen labels of the keys they don't use. Even though you present the same material in the second class as you did in the first, they're now ready to learn and use the next level of expertise; they'll think you're godlike because you've provided them with it. Develop an overall TURBO OA class that touches on many products for the advanced users. Use your marketing skills here; non-technical people love taking classes with "Advanced" in the title. Be sure to give them an "Advanced" certificate they can put on their walls!

Be prepared to perform career counseling--everyone wants to do what you do.

Do not base any action or statement on an assumption; you'll get caught in it every time and only then will you remember you weren't supposed to do that.

I guarantee you that you'll learn as much or more as your students in your first several classes. It's truly amazing the keystrokes new users can combine, the assumptions they've made, the questions they can ask, the misconceptions they have.

USER SUPPORT TIPS

Realize you will be providing a significant level of ongoing user support for most of your organization and learn how to manage it lest it manage you.

Institute a HOTLINE phone number for user problem calls. Be sure to have some form of call log to keep track of who calls at what time with what questions. Announce the HOTLINE on brightly colored paper with the phone number contained in a box they can clip out and tape to their terminal/PCs. If you use HPDesk, send a message to your customers announcing your HOTLINE service; set up an HPDeskManager "name" such as "HELP OFFICE" for non-critical questions. Purchase an answering machine for those times when you must be away from your desk.

Whenever possible, start by looking up the answer in their reference guide or student workbook if you go to their desks, or have them do it with you over the phone to teach them to look up their questions first instead of calling you. Users never want to look in the reference guide.

When you go to their desks to solve a problem, you tend to disregard what they say and instead concentrate on the screen. However, when you are answering a problem call over the phone, you must define the terminology they use as well as deal with their frustration. For instance, their entire problem description, stated in a near-hysterical voice, is, "It doesn't work!" or "I broke it!". They think "the system is down" because HPDeskManager is temporarily unavailable. To many users, all software is described as "the systems". Some people call

function keys "pushbuttons". Since you can't see the screen, you need to have them define and describe clearly what the problem is so you can diagnose it and prescribe the solution. This can be difficult since they're already frustrated because they have a problem they don't know how to solve, they're under intense time pressure and your response is to ask question after question when they think they've already defined it clearly to you. BE PATIENT. Very often you'll find the problem is a simple case of user misunderstanding. Now you've added embarrassment to their frustration. BE NICE.

Don't assume they know enough before they call you to check to be sure (1) it's turned on, (2) it's plugged in or (3) it has a port if it's an HP3000-related product and (4) the port cables are securely connected. It's amazing how many people will call you before they check to see if the terminal/PC is plugged in.

Some people relate better when you explain the WHY and HOW of a problem solution so they can handle it themselves the next time it occurs; others don't want an explanation at all, they just want the problem fixed FAST...these people are often your ongoing support problems.

After you've been solving problems for a while, analyze your trouble log sheets to see which departments have the most logged calls and think about conducting special workshops at their department meetings to lessen your support burden. Very often entire departments have formulated a unanimous assumption about your role; for instance, you exist to respond to their calls about every little thing. Very often, also, user misinformation spreads throughout a department so a department-wide solution is more important than resolving the misunderstanding on an individual basis.

When you get stumped with a problem, you can consult with your system manager, an available "wizard" (this can be just about anybody in an organization, and every organization has one), or HP's Response Center. Don't forget to look it up first in the reference guide! Sometimes, although I suppose it's unprofessional to admit it, if the problem can't be replicated the only answer is sunspot activity or a full moon. On rare occasions, there just isn't a reason why. It's hard for a computer professional to admit that and not invest time in finding a reason somehow, but you already have enough to do, right?!

Be prepared for lots and lots of calls ranging from the extremely complex to the ridiculously simple (remember to BE NICE).

If you're the one and only OA support person, all those users will very soon become the major consumer of your time and efforts. If this is your situation, try to set up department support people to be the first line problem solvers. Use those "OA Mentors" and "PC Pioneers"! Another good use of your trouble

log is in training your PC Pioneers and OA Mentor volunteers, or new staff as you acquire them. There's no point in their learning how to solve all those problems on their own; use your proven experience to help them be more successful and lessen your involvement.

MANAGING YOUR ULTIMATE SUCCESS SYMBOL - THE HUGE TRAINING WAIT LIST

We haven't found a simple solution for this one yet. We currently have an average of 1,000 to 1,500 names on the wait list at all times. You can't win for winning; once you've trained them in the Intro to the HP Touchscreen class you get to cross their name off the wait list, right? Right. PLUS now you get to add it 8+ times for WordStar, Executive MemoMaker, Personal Card File, Advanced Link I, Advanced Link II, Drawing Gallery, Charting Gallery, Lotus 1.2.3, etc.

We are attempting creative solutions...

- Even though we advocate small classes, they have little impact on the wait list. So we're trying larger classes--up to 16 students--to bring down the backlog. The problem is the more people in class, the slower the class moves and the less likely people will ask questions.

- We're taking advantage of a temporarily available classroom in our building in addition to our two we already schedule each day in order to accommodate more people/month.

- We're scheduling students into available classrooms at an HP sales office in addition to our three rooms per day. This provides us with additional space and instructors on an as-needed basis and takes 10-16 students off the wait list per class. (Yes, even though we're HP, we still have to pay...we just bill our costs back to the location codes of the students.)

- We're testing some video training for larger groups that was developed by our corporate Office Utilities Group in response to the need expressed by HP's OA people around the world. Utilizing this tool will enable us to conduct larger classes with only one lab assistant, thus allowing us to apply our instructor resources elsewhere.

Ideally, we'd like to take a proactive stance of addressing departmental training requirements and tailor training to specific department needs vs. reacting to the pressures of individual requests on the wait list. We also want to develop job-specific training...training courses specifically targeting secretaries, marketing and engineering professionals, managers.

The sheer volume of PC-based software creates training and support dilemmas...do you attempt to develop a training course

for every word processor that runs on the HP Touchscreen? every spreadsheet? every graphics package? We get calls as soon as a new software package is announced asking when will the class be available. That creates a lot of pressure for the OA staff, not to mention users. You can't realistically expect people to learn a new word processor every several months. People are creatures of habit and don't like to change. If they've found a word processor they like, don't support efforts to pressure them to forsake it and subject themselves to the learning cycle all over again just because it's the latest and greatest. Besides, product-specific training is not like Cup-of-Soup, add hot water and stir. Course development takes time. Our strategy for dealing with this was to set forth a policy statement that our training and support efforts would be directly related to and supportive of HP's Personal Productivity Center software. That means that if a software package is not considered to be part of the PPC offering, we don't develop a class and we don't support user questions. Given the customer base we have and the size of the wait list for PPC products alone, we can't realistically take on more.

MAINTAINING YOUR SANITY

It's Not ALL Bad...

I've been in OA at HP for four years and really don't think often or seriously about moving into another area. I love OA. And so do most people who are involved in OA whom I've met both in and out of HP. Interestingly, many of the OA people I met when I joined HP four years ago were HP's OA pioneers...and most of them are still involved in OA. Those people who have gone in other career directions often tell me they really miss OA. It's really endlessly fascinating and challenging because there are so many facets of office automation and they change so rapidly.

You have the opportunity, as an OA coordinator, to interact with ALL of the people within your organization and not be limited by functional or job-related boundaries. You have the daily personal gratification of receiving lots of positive reinforcement in the form of thanks, smiles, "AHA!"s and occasionally cookies from students and people for whom you solve problems. You have the professional gratification of knowing that your contribution is recognized by others as being valuable to the company and to the people with whom you work.

...But There Are Days...!

There are those days, however, when you'd trade it all in in a nanosecond for any other job. Those days, thankfully, are few. One thing that's probably more helpful than anything else in maintaining your sanity is to develop or join an existing OA network. If you are in a large company with many sites, form or join an existing Office Automation Coordinators group. At HP we have had one for about three years, the Office Automation

Coordinators/Trainers Group, and it has over 100 members worldwide. We have a three-day meeting annually to update our members on new products, new training and support tools, and simply share information and swap war stories. It's an annual treat to experience the enthusiasm and excitement these people feel about their jobs. And it's very gratifying to discover that you aren't the only trainer in the world who's ever fantasized about skewering students on your teacher's pointer. It's a relief to find out you're not the only OA support person who's ever dreamed of ripping the HOTLINE phone out of the wall, or of hitting a user upside the head with the reference guide s/he won't bother to check before calling you. It's truly sanity maintenance. Besides, these OA people are creative problem solvers...they've all devised clever techniques for dealing with problem situations common to us all, and a network of OA coordinators is a great place to learn about those techniques and to share your own. Use the internal network when you have a problem you can't solve; someone's either solved that problem before or is working on it now, and there's no point in everyone inventing the same wheel. Use the internal network to acquire additional training courses or to enhance existing ones or to get feedback on material you've developed or solutions you've implemented or to find out about internal classes and workshops you can attend to develop and improve your skills.

If your company isn't a large one, other companies in your vicinity will have OA coordinators. Invest some of your valuable time specifically for sanity maintenance...call other companies and talk to OA coordinators; if they don't know of any existing groups you can join, I guarantee you they'll be interested in forming one with you. You can take advantage of the enormous amount of knowledge and skills available in a company-independent network, too. Even though they may be using different vendors' software, you can still acquire useful information about how they manage their training and support demands, magazines and books specifically oriented to OA to keep you informed about industry trends, teaching techniques they've found to be effective, outside seminars and workshops that will improve your skills. If for no other reason, it's simply wonderful to get together periodically with other people who are just as crazy about their jobs (and maybe on those infrequent bad days, just plain crazy!) as you are.

3020. Ergonomics and VPLUS/3000 Screen Design

Lance J. Naber
President
L. J. Naber & Associates, Ltd.
119 East Hartsdale Ave.
Hartsdale, New York 10530

Introduction

"Ergonomics" is "the science of making the job fit the worker"[1]. It includes attempts to make computer hardware and furniture more responsive to the needs of the worker. Tilt and swivel screens, adjustable desks, adjustable chairs with special backrests, sculptured keyboards and palm rests are all examples of ergonomic applications.

VPLUS/3000 includes a set of programs which enables the HP 3000 user to access data on the computer by using a computer "screen". This screen has been designed and programmed to meet the needs of the user. This paper will discuss considerations for ergonomic design and implementation of VPLUS/3000 screens. Many magazine articles refer to this quality as a measure of "user friendliness". Since many users are going to spend a considerable amount of time in front of a computer terminal, programmers might as well make it as easy as possible for them.

Components of User Friendliness

Applications described as user friendly share a number of characteristics. First, the novice user should not have a difficult learning curve. Foresight should be used to provide the novice user with the necessary tools to simply learn the system. Second, the intermediate user should not have to go through the same long set of explanations he needed as a novice every time something simple needs to be done. Finally, the advanced user should be able to perform his duties with a minimum of impediments.

A camel has often been described as a "horse designed by a committee". It implies that many compromises were made along the way which provided something for everyone but not enough for anyone. Ergonomic screen design has similar problems. Elements that make a screen just right for the novice user are totally inappropriate after that user has developed some skill using it.

Some components of user friendliness are applicable to all levels of users. For instance, the ability to print the

contents of any screen is quite useful. All users desire the ability to have a simple exit from the system. All users need a way of refreshing the screen if a power failure or other problem has caused it to lose the image. These components are easily satisfied.

Other aspects of user friendliness are completely judgmental. For these aspects one must remember to consider the specific audience and make a decision. Let's start simply and illustrate this point.

Date Validation: The Right Way?

There are many different ways of inputting and validating date fields. For example:

- 1) mm/dd/yy where the user inputs all eight characters.
- 2) mm/dd/yy where the "/" characters are fixed, the user inputs just six numbers.
- 3) ccyyymmdd where the user inputs just eight numbers.
- 4) mm/dd/ccyy where the user inputs all ten characters.
- 5) mm/dd/ccyy where the "/" characters are fixed, the user inputs just eight numbers.
- 6) ccyy/mm/dd where the user inputs all ten characters.
- 7) ccyy/mm/dd where the "/" characters are fixed, the user inputs just eight numbers.

Note: cc = century, yy = year, mm = month, dd = day.

At least a dozen more combinations and permutations are possible. Which method is the best? The method a particular set of users is accustomed to is the best for them. Methods 2, 3, 5 and 7 have the least amount of typing for the user, but method 3 uses a date format unfamiliar to most users. I prefer methods 2 and 5 because they are the simplest way to use detailed error messages and maximize the ergonomic qualities. The other methods allow the programmer to use the VPLUS "MDY" date format and results in a somewhat cryptic error message: "The field must be a valid date, in MDY order". On the positive side, the other methods enable the user to input the date just as it would appear on an input form.

Whichever date structure is chosen, it must be consistent throughout the entire system. If maintenance is being performed on an existing system which uses dates, the programmer MUST maintain consistency. Otherwise, the user may go crazy trying to remember which date format is used in which field on which screen!

The first rule of ergonomics is CONSISTENCY. A poorly conceived screen that uses consistent conventions is still more usable than one that has better concepts but lacks consistency.

Programmable Function Keys

Programmable function keys are used by the programmer to implement frequently used functions or provide special paths through the application. "Programmable" means their values and uses may be defined or redefined as needed. For example, using a function key to always return to the Main Menu is very helpful to the user. This key is "programmed" because its response is translated into a specific action by each program. Remember consistency: if a Main Menu key is defined as f1, keep it same key for all screens.

How do programmable function keys contribute to the ergonomic screen?

Certain basic screen functions are necessary at all times. Providing the user a dedicated way to access these functions will significantly improve usability. For instance, the user always needs a way to EXIT the system. A very simple way to provide this function is to dedicate a specific function key.

The HP Entry program provides a simple convention for dedicated function key assignments. If your installation uses the HP Entry program for data entry, then keeping to these conventions will simplify learning the system for your users. Even if you don't regularly use the HP Entry program, its conventions are useful as learning tools until experience enables you to develop your own conventions. The exit key defined as f8 in the HP Entry program.

When dedicated function keys are used, they must be labelled for clarity and ease of use. Otherwise, you give up some of the advantages of using them. Function key labels are hardware dependent. If the terminals in use are HP 262X compatible, then label the function keys in the VPLUS screen definition area. If the terminals are 264X compatible, use a function key overlay on which you've written the dedicated key assignments. Another way to provide on-screen function key labels for a 264X terminal is to dedicate lines 22 and 23 of the screen for function key labels. This takes up two valuable lines and may not be desirable for certain applications.

Another basic function is REFRESH. If the user screen is erased by a power failure, hard reset or other problem, the refresh key provides the capability to redisplay the screen in its current form. Providing this function is quite simple using the VPLUS intrinsics. Set the NFRAME in the VPLUS COMMAREA to "\$REFRESH" and call VGETNEXTFORM, VINITFORM and VSHOWFORM to perform a



refresh. With the HP Entry convention, f4 is always refresh.

Every user will want the capability to PRINT the contents of any screen. Once again, a VPLUS intrinsic exists to provide this capability. Simply define a function key in the program to call the VPRINTFORM intrinsic and the capability is provided. Using the HP Entry conventions, f3 is always print.

Menus

A "Menu" is the master list of available choices, whether it is found in a restaurant or on a computer screen. One component of user friendliness is having a menu structure to help the user remember the available options and easily use them. An ergonomic menu is easy to read and has a limited number of selections. Why a limited number of selections? A menu of 50 items is too busy and more difficult to read than one with 10 items.

One menu technique is to list the items on the screen with associated numbers or letters. The user selects the desired function, enters the number or letter on the screen and then presses the ENTER key. Another technique is to assign each menu item a programmable function key. This limits the length of the menu to a maximum of eight functions on a Hewlett-Packard terminal. However, since some function keys are usually dedicated to specific uses, the practical number is five functions (PRINT, REFRESH, MAIN MENU or EXIT are frequent values of dedicated keys).

Once again, a choice is needed. Letters, numbers or programmable function keys? Programmable function keys are the easiest to use, but severely limit advanced interaction. The user must always use the programmable function keys to work through menu layers. Selecting from different letters (and using the enter key) is somewhat slower, but more flexible. But how many people can find an "E" on the keyboard easily? My choice is simply to choose a number (from the numeric keypad) and press the ENTER key. Why?

Consider the instance where there is a three level menu. The novice user who selects option 1 on menu 1, option 4 on menu 2 and option 3 on menu 3 needs the titles to remind him of what the choices are. The intermediate user may want to go directly to menu 3 before deciding on a selection. The advanced user could choose the last option directly. How?

The input field on the multi-level menu can be five (or so) characters long. Choosing "1" would bring up menu 2. Choosing "1.4" would bring up menu 3. Choosing "1.4.3" would designate the desired screen and associated program directly. This is not as difficult to implement as one might expect. VPLUS provides the ability to left justify the input and use a "MATCH PATTERN" to validate the input against. The pattern is "MATCH d[.d,.d,d]" for a three level menu structure. The brackets mean the items separated by commas are optional and both valid (the comma represents a logical "or"). Once the input is validated as proper, VPLUS can be used to display the proper screen, without the need for external program screen control. Techniques to control the screen sequence programmatically are used by more advanced programmers.

This example illustrates another point. The programmer may not desire (or need) to add the advanced functions immediately. However, this last design provides the programmer with a "hook" that can be used later if the users demand a way around the multi-level menus. Remember the camel. The programmer can provide different levels of users each with appropriate functionality (and thus build a better camel) with proper planning.

Display Enhancements & Line Drawing Set

Display enhancements are hardware functions provided in the computer terminal which change the way the screen looks. Examples of enhancements are inverse video, half-bright, full bright and blinking. The Line Drawing set is a secondary character set available for a computer terminal which enables forms to be drawn that closely approximate printed forms. The combination of these two hardware elements enable the ergonomic minded programmer to become even more user friendly. Display enhancements are available on all HP terminals. The Line Drawing set is optional and must be purchased separately.

One note of caution. Display enhancements and the line drawing set can help create a truly exceptional form - either excellent or terrible. Care must be taken to consider the aesthetic qualities of a screen every time one is created.

Display enhancements are used to delineate data fields, both protected and unprotected areas. Unprotected areas are fields where data may be entered. Protected areas are not available for data entry. For instance, a data field may be half-bright, inverse video in its initial state. On an error condition, it is changed to full-bright, inverse video by the VSETERROR intrinsic. Another option is to make

the data fields half-bright, underline video normally and flash them to full bright, inverse video on an error. The specific options chosen are a matter of taste. Once again, stay consistent with whichever you choose.

The Line Drawing set is useful for demarcating logical groups of data on a screen. Try not to make the screen very busy or the data entry people will have problems with it. One use for the Line Drawing set is the ability to create a screen that looks exactly like a printed form, simplifying the data entry function. Usually the printed forms must be redone to conform to the limitations of the screen and cursor movement.

Form Families

Before VPLUS/3000 was introduced, certain data validations and protections had to be implemented programmatically. This was a serious shortcoming of V/3000 and required the programmer to cover for it. An example will illustrate it:

A simple data modify function is implemented in two steps. The first step enables the user to retrieve the record to be modified. The second step allows the user who has retrieved the record to make changes and place it back on the data base. After the record is retrieved, we desire to protect the key field to prevent it from being changed. The retrieved key value is stored in the working storage (or linkage section) and compared to the key value after the second step. If it has changed, the programmer puts the original value back, highlights the field in error and displays an error message in the window.

Form families simplify this problem. A form family is a set of forms which have exactly the same field definitions and layout. The differences arise in the editing and protection characteristics for the fields. In the example above, a form family is used to create a second form. The second form is different from the first ONLY in that the key field is now protected. Otherwise, the two screens LOOK identical. This change is transparent to the user (until the user tries to enter data in the now protected field) and operates very smoothly and impressively. Note that you might want to implement a programmable function key with a CLEAR function to enable the user to cancel the change in the second phase and return to the initial screen.

This simple example illustrates the assistance form families offers to programmers. All of the programming

for validating the key value is no longer necessary. The form family concept makes the programmer more productive. Another use for form families is implementing a simple security scenario. People with different security clearances can be shown different screens. Protect the fields the person is allowed to see but not modify. Form families are an important tool to the ergonomic programmer.

Putting It All Together

We have introduced several new concepts that will increase the usability of any system. Use menus to provide simple access to the system. Programmable function keys enable important functions to be dedicated. Display enhancements and the line drawing set enable the programmer to draw useful forms. Form families reduce the amount of code a programmer has to write in specific situations.

Consider the FLOW of the screen. How will the cursor move from field to field? Look at the following screen:

Selection Code [] A=Add, M=Modify, L=List, D=Delete

```

First[.....] Last[.....] Dept[....]
Address[.....]          Group[....]
      [.....]          Area[....]
City[.....] St[..] Zip[.....]
Phones:
  Home[...] [...]-[....]      Flags: A[.]
  Office[...] [...]-[....]    B[.]

```

This screen is easy to read and well organized. But it is not very usable. Why? See how the cursor will move. Left to right starting from the top left corner, then repeating the left to right on each next lower line. The data will be entered into the Selection Code, then First (Name), Last (Name) and Dept fields. Then Address and Group, second Address and Area. The data entry person will have to jump all over the screen. Logically, the data entry person will have to shift his attention every line from name and address data to other peripheral data. The same problem occurs with the Phone fields and the Flags. Look at a better organization:

Selection Code [] A=Add, M=Modify, L=List, D=Delete

```
Dept[....] Group[....] Area[....] Flags: A[.] B[.]
First[.....] Last[.....]
Address[.....]
[.....]
City[.....] St[.] Zip[.....]

Phones: Home[...] [...]-[....]
Office[...] [...]-[....]
```

Note the difference in the cursor movement on the screen. The cursor will start in the Selection Code, move to the Dept, Group, Area and Flags. Then the name and address fields are accessed as one group all together. This is very important! The data entry person will be able to enter the data in a more logical way. The advantage will be evident in use: better screen organization results in fewer keying errors and higher productivity.

One way to ensure the flow of the screen is to walk through it with other programmers and possibly even the users. I always ask other programmers to critique my screens before they are finalized. It is important to organize the screen to maximum advantage, even if it means doing it over more than once. The time spent ensuring the usability of the screen will pay for itself many times over.

Error Messages

The ergonomic programmer remembers to pay attention to details that directly affect the user. The error message "Field must contain all digits" is intimidating to a user, while the message "The Zip Code must be all numbers, such as '10530'." conveys the same meaning and is much more user friendly. Avoid cryptic error messages. If in doubt, test the error messages on the user prior to implementation. Users prefer to provide input before a system is implemented to avoid being totally dissatisfied after implementation.

Do's and Don'ts

DO be CONSISTENT.
DON'T clutter the screen.
DO use the Line Drawing set to create forms.
DON'T make the screen an eyesore.
DO use Form Families to reduce programming time.
DON'T make your screens overly complex.
DO use Programmable Function keys to simplify the application.
DON'T overuse the Programmable Function keys.
DO clearly label your Programmable Function keys.
DON'T forget to provide PRINT and REFRESH Programmable Function keys.
DO design your screens taking care of the data flows and cursor movement.
DON'T rush to create your screens without careful thought.
DO use Menus to simplify use of your system.
DON'T create cryptic error messages.

Conclusions

Ergonomic screen design is the product of foresight and careful planning. The programmer who is aware of the subtleties of screen design has taken an important first step. The ergonomic minded programmer always keeps the user in mind. This care will show in the final implementation and may be the difference between a smashing success or a withering failure.

Footnote

- [1] The New York Times Everyday Reader's Dictionary of Misunderstood, Misused, Mispronounced Words, Edited by Laurence Urdang, Weathervane Books, 1972, Page 102.

3021. 4GL AND REALITY

Wallace M. Snesrud
General Mills
Box 1113
Minneapolis, Minnesota 55440

This paper will discuss the input of the elements of the four stages of integration of a fourth generation language. A case study will then be discussed showing how to implement a fourth generation language in a development environment.

The development of fourth generation languages is a result of the information explosion. The successful companies of the future are the ones who will gather, assimilate, and use all this available information. Fourth generation languages provide these progressive companies a way to master the use of this data in a most cost effective manner.

The concept of system development using fourth generation languages has been discussed for several years. Attempts have been made with varying degrees of success to integrate these languages into the data processing environment.

The new languages have been described at one extreme as the solution to the problem of solving the two year development backlog, to the other extreme of providing managers with control of all their informational needs. Like the sales brochures that promise solutions, there are many ways to implement the languages. The product implementation method will have the largest single effect on fourth generation language acceptance and integration into a company.

Developing a plan to adopt a fourth generation language must take into account the four stages of integration: A.) Conceptualization, B.) System generation within narrow boundaries, C.) Standardization, and D.) General adoption. The implementation of the 4GL must take into consideration the two major areas of MIS development methodology and the skill level of the people using the languages.

The MIS methodologies are divided into three broad categories: 1.) Traditional life cycle development, 2.) Prototyping environment methodology, and 3.) Integrated environment methodology. The opportunities for savings not only in time and effort but in cost effectiveness are dependent on the methodology used and how the fourth generation language is to be integrated.

The traditional life-cycle development does not lend itself to procedures outside the specified controls and functions of the methodology. Retention of the established methods will relegate the use of the fourth generation language to a substitute for the

current third generation language. Such substitution is essentially the streamlining of the production of the code.

Prototyping methodologies provide additional opportunities to increase efficiencies within companies. The detailed design process is shortened by using the iterative program generation process to provide a usable program. Fourth generation languages provide the necessary ability to develop and modify programs with maximum ease and minimum time. Thus the integration of fourth generation languages into the prototyping methodology enhances the pay-off and justification for the use of prototyping.

As with all new technologies the greatest benefits of the 4GL come in application of the technology in a way which matches the capability to the usage. The methodology must go further than prototyping. It must build a working relationship between the professional DP staff and the client. The goal of productivity is to produce the greatest amount of work while expending the least amount of effort. If the system analyst, working with a client, uses a fourth generation language to produce the system as the design process proceeds, the desired system will be generated.

The categories of methodologies must be reviewed in light of each of the four stages of implementation:

A.) CONCEPTUALIZATION:

The system definition within the life-cycle methodology narrows conceptualization to the process of writing the programs. Proper training of staff will allow breaking out of the mold of a third generation language or the "unlearning" of COBOL. This will allow the program flow to follow action order logic rather than COBOL dictated program logic.

Prototyping methodologies provide additional opportunities to expand the conceptualization process. With prototypes there is the opportunity to envision the entire process, with the prototype providing the proof of functionality unhindered by the constraints of program logic. The feasibility of this uninhibited conceptualization contributes to the productivity and creativity utilized within the entire prototyping process.

The integrated environment methodology requires the ability to conceptualize the system in a way the end user of the system will utilize it. The cooperative method of development by the programmer and system end user requires that the concept of the system ultimately be generated and understood in the terms of the user within the framework provided by the programmer.

B.) SYSTEM GENERATION WITHIN NARROW BOUNDARIES:

This portion of the integrated process can be classified as the "shakedown" period. During this timeframe the functionality, usability, and integrity of the fourth generation language system must be evaluated. Although productivity is not very high during this period, the staff is gaining the necessary experience to use the system. The learning curve is an inescapable part of the shakedown project and can be minimized by proper and complete training on both the computer system and the fourth generation language software. Requiring the staff to learn both the hardware and the software results in a learning curve which is longer than the individual curves would be. The process of defining standards for the new system begins at this time, as the staff finds ways to adapt the new product to their current environment.

The shakedown project will be eclectic with techniques for both management and development drawn from all three major development methods. Since the motive of this stage is to learn about the 4GL product, a careful evaluation of direction within the company must be made. A careful analysis and definition must be performed drawing on the lifecycle methodology. When the analysis is completed and the concept of the end result decided, it must be frozen. The shakedown cannot afford the luxury of a moving target! The uncertainty of what can be ultimately done with the 4GL product brings into the shakedown the creativity and innovativeness of the prototype methodology. The functions are first blocked out, with trial and error determining what can be adopted. Considerable time must be spent determining the limitations of the 4GL product. It is essential to stress the different components within the 4GL chosen by the company, both to determine impact on the computer system and the development method.

The necessity to determine the viability of the end user tools brings into play the methods of the integrated environment. If possible these tools should be evaluated by an end-user. With this person on the team, the true usability of the end user products can be determined.

The members of the shakedown team must be flexible and open minded. Many difficulties will appear if team members are not receptive and open to change. The documentation will not be as straight forward or understandable as it should be. The software won't be able to do what you want in the way you want, or it may have bugs in it. Add to this the frustration of the learning curve to use the product, and an open mind is a necessity.

As the shakedown progresses the knowledge level will increase. As the mass of knowledge grows, the standards needed for full integration of the product will begin to

take shape. The basis for the next step has been established.

C.) STANDARDIZATION:

Each of the methodologies will have unique standards. But fundamental to all methodologies is a common basis of information and accessibility. The variations will come not in how the data is accessed or modified, but rather in the way the programs are written to do the access and modification. The data elements and their relationships must be established in the dictionary. Since the 4GL systems have provided accessibility to large amounts of data to a large number of non-programmer trained professional people, the ability to control that access becomes very important and more difficult to manage. As such the addition of data items and their relationships must be done in a controlled way, preferably by data administration. The guarantee must be such that the information placed in a data source by a program written by a programmer will be the data ultimately expected by a financial analyst using his PC to generate code to extract and transfer that information for his own use.

During the shakedown process most of the idiosyncracies and difficulties of the particular 4GL product should have been discovered. The number of idiosyncracies will be determined by how non-procedural the 4GL is. With this information available to the data administrator, standards can be defined which will minimize the problems arising from these idiosyncracies and provide a high probability of accuracy and efficiency. Standards of naming and usage can be defined by the programmers and analysts which will minimize the effects of the idiosyncracies on computer resources, response time, and program generation.

The final area of concern is support and maintainability. With fourth generation languages there will be three distinct groups generating programs; professional programmers, generalists on the DP staff, and end users. Because ownership and support of systems may change from one development to another, the support responsibility will also fluctuate. In order to minimize the costs and delay of support functions, programs must be generated to standards.

D.) GENERAL ADOPTION:

The final step is moving the language into general adoption. This process requires training and assistance for all three methodologies and all categories of users. The programmers, prototypers, and end-users must all have training in the use and applicability of the product. Although the type of support and assistance will differ with each group of users, each must have an expert in 4GL with which to work and explore ideas. The programmers must have product support as well as intensive technical support. The generalists

generating the prototypes must have high level product support. Finally the end users will need assistance in the use of computers and of the languages that are available to them.

The cost of implementation can be measured in two ways, monetary and internal change within the company. The monetary costs consist of purchasing the product, computer system upgrades to handle the load generated by the product, and additional staff to provide the training and support necessary to make the integration of the system successful.

The cost of the change internal to a company is much more difficult to quantify. The integration of a 4GL system will result in a radical departure from established patterns within current MIS departments. Management has to be aware of and prepared for the expected decrease in productivity during the learning curve.

This start-up time allows for a broadening of tasks and a merging of roles across what may not be traditionally related company departments. It will be necessary to develop high levels of human relations skills and a sensitivity not only to the apprehensions of traditional non-users, but also to a perceived threat to the traditional role of the MIS department. The center of interaction will shift from the technical discussions within the MIS department to discussions of solutions and new innovative report generation within the end user community.

Successful implementation of 4GL's yields a service environment which draws on a range of development methods providing the most cost effective solutions to the problems. The successful solutions bear the mark of creativity, with the results oriented toward answering problems rather than the application of methods. This end result shows itself externally by an innovative, progressive company able to expand its market share because of its flexibility and quickness of response. It shows itself internally to the company by creating a more unified staff willing and able to work together because each understands the other's responsibilities. The positive advantages of a fourth generation language allows the company to position itself such that expansion is possible without major cost and additional staff positions.

SNOWFARM ENTERPRISES, INC.

Snowfarm Enterprises, Inc. has a \$100M wholesale marketing group that is currently using a HAL 999 mainframe computer system. The model 13 processor is a 1.5 MIPS computer with 2 gigabytes of storage. Changing trends during this last year have forced the marketing group to have much better access and control of sales

and cost information on a timely basis to better develop a competitive sales approach. This department has generated 786 requests to the MIS group for various programs, reports, and databases. The rapid growth of the company as a result of these sales has caused the financial/accounting group to keep an even closer watch on the corporate cash flow and company investments. As a result they have generated an additional 326 requests to the MIS department.

These requests have created an estimated backlog of 2 years and 4 months of work for the computer department. Being a company with enlightened management, the chairman of the board has directed the Director of MIS to find a solution to the backlog. The director formed a team to evaluate and suggest solutions. The team consisted of a consulting planner from corporate services, a system programmer, a marketing/sales specialist, and a financial analyst. After considering several solutions to the backlog of requests, including a significant expansion of staff people throughout the company, the team decided to recommend to the Board the purchase and implementation of a fourth generation language. After a review of products available for their HAL 999 computer, they decided to recommend implementation of the INFOTRIEV 16 software system. INFOTRIEV 16 was chosen over many available fourth generation languages because of its four major component parts. They were:

1. A programmer productivity system allowing generation of on-line and batch programs.
2. An information retrieval/report writer which is menu driven to allow the end user of the system a way to generate their own reports.
3. A dictionary system allowing definition of elements, concepts, and relationships. The security of the software is also built into the dictionary.
4. A module allowing personal computers to access the information files, run the system as a terminal, or act as an independent computer. With the PC acting as an independent computer, the mainframe module communicates with the PC module allowing automated generation of data requests and extracts and ultimate transfer of data to the PC. Automated loading modules can load the data into any of a number of PC tools.

The recommendation was approved by the Board and the team was assigned to implement the system. The project team developed a comprehensive plan to introduce INFOTRIEV 16 to the MIS department and the end users. While developing the plan it was determined that the attempts to resolve the backlog had resulted in three development processes being used by the department. They were traditional life-cycle development, prototyping, and integrated environmental methodology. The Accounting Department demanded the thorough analysis and controls of the life-cycle methodology. Because the Sales/Marketing Department wasn't sure if the proposed systems would give them the information they needed, they requested the use of prototyping. Attempts were

made by the MIS department to use the integrated methodology, but it failed because of inadequate tools.

This major change in the development process within the company forced the MIS director to establish a training program for both the computer services staff and the end user. This was necessitated by need for the establishment of communications between various company departments, the lack of understanding of these internal relationships, and to facilitate education on the use of hardware and software.

At the suggestion of the MIS Director, three new staff members were hired and introduced into the 4GL process. A data administrator who would establish, track, and control data elements and their associated relationships. A product support specialist would provide technical support for INFOTRIEV 16. And finally, a staff consultant was hired to assist both the system programmers and end users in the use of the 4GL tools and to provide the necessary educational support services.

A core group was chosen to participate in a "shakedown" project designed to allow the staff to determine how best to use the product, find its strengths and weaknesses, and to develop a foundation to be used in establishing standards.

The project team, now consisting of 8 members, (a consulting planner, a product support specialist, a data administrator, a staff consultant, a project leader/programer, a financial analyst, a marketing specialist, and an experienced programmer), was provided with the vendor recommended one week of training. The shakedown project chosen was a re-write of a small information retrieval system which reported product change history for the marketing specialist who was on the team.

All members of the team applied their own individual talents and skills to the shakedown project with some unexpected results for the company. The consulting planner learned enough about the products to make recommendations for changes in their end use. The system programmer determined the possible effects of the project on computer system performance. The marketing specialist learned the use of 4GL tools for the extraction of information from the data base and how to down load it to his PC for personal departmental use. The data administrator determined how to set up data items and generate efficient relationships across all informational boundaries. The product support specialist developed a list of the most asked questions from all the team members and provided their answers to the programming staff and the data administrator. The staff consultant generated the training program. The project leader formulated a management philosophy taking into account the capabilities of the 4GL tools available to the end users. The programmer became very knowledgeable not only about the INFOTRIEV 16 system and its place within the MIS Department, but also gained tremendous insight

into the various functions of the other involved departments and the company as a whole.

Finally, the project team analyzed the results of the shakedown and developed standards for the use of its tools. The standards created were such that they moved from the least restrictive environment (integrated development) to the most restricted environment (life-cycle development). A determination of the enforcement of the standards was incorporated into the report.

With the experience and standards in place and potential problems resolved, the INFOTRIEV 16 system was released for general use within the Snowfarm Enterprise Company. It could now begin its attack on the tremendous company backlog of requests from accounting and marketing. The major support systems which traditionally provide data to the corporate data bases and which require significant audit control functions, were developed by the life-cycle methodology. The use of the 4GL facilitated the ability of the company to react rapidly to external changes and stimuli by quickly providing solutions for informational requests.

The prototypes generated by the team resulted in several modified systems being put into production within the company. The process allowed them to be supported by the MIS department or absorbed within the departments of the end users for their own use and support, removing that part of the production process burden from the MIS department. It ultimately also allowed for shared use of the end reporting processes across several functionally different departments further reducing the number of requests for reporting support to MIS. The prototyping resulted in the demise of several systems which were not providing the necessary information to the right departments.

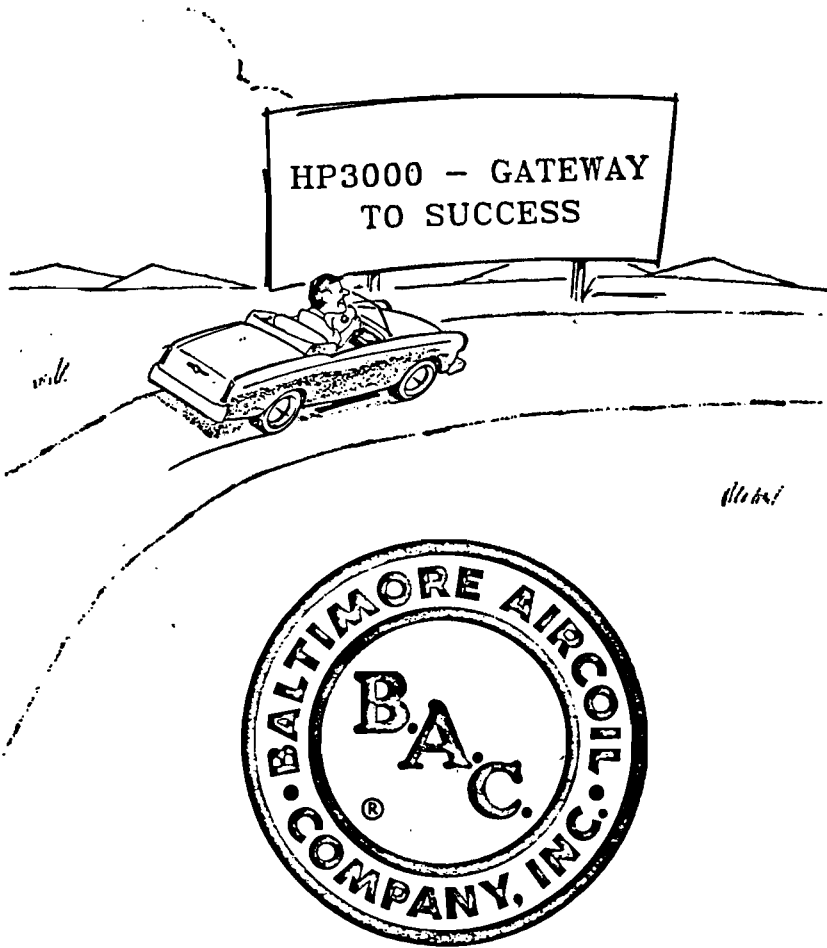
With the data now properly cataloged and easily accessible to the end user within their departments, a marketing strategy was formulated. The strategy and data were made available to them on their PC's in a form they both understood and could use. The accounting department could also generate their own reports and yet maintain security of sensitive financial records within their own department. Shared reports on sales figures and projections were now available not only to sales and marketing, but also to corporate advertising and shipping, allowing future projections to be made and shared with all departments finally putting them in synch one with the other.

Because management of Snowfarm Enterprises was open minded and progressive enough to look outside their company for an answer to their increasing backlog problems, the company increased its flexibility to meet demand in the marketplace. The integration of the fourth generation language not only brought departments closer together allowing better understanding of their separate functions, it rapidly and efficiently reduced the backlog of old requests and further reduced the added responsibility of new

requests by placing part of the burden of report generation on the end user. This was possible because the fourth generation language made computer usage accessible and understandable to the non-professional programmer.

3022. HP3000 - GATEWAY TO SUCCESS
Copyright 1984, Baltimore Aircoil Company, Inc.

John T. Skrabak
Baltimore Aircoil
P.O. Box 7322
Baltimore, Maryland 21227



INTRODUCTION

Baltimore Aircoil Company, Inc. (BAC) specializes in the design and manufacture of evaporative cooling products used in the building trades and industrial markets. With fourteen plants world-wide, BAC found itself in 1982 with multiple brands of expensive hardware and multiple versions of outdated software which marginally supported the organization. Today, BAC has a structured hardware/software architecture -- much of which is already in place in North America coupled with a two-year plan for extension world-wide.

From a data processing perspective, we have an exciting story to tell -- going from an HP3000/40 in November 1982 to four HP3000's networked across North America today to thirteen-fifteen planned world-wide by 1986. Today's case study/methodology is targeted to first-time users and/or potential users. A number of topics covered will interest seasoned users as well.

Today's agenda, then, will include:

I. HARDWARE SELECTION

- . Define your requirements.
- . Maximize your flexibility.
- . Consider current and future market trends.

II. TELECOMMUNICATIONS STRATEGIES

- . Coping with the AT&T Divestiture.
- . Remote printing at high speeds.
- . STAT MUX point-to-point vs. MTS.
- . Cost optimization.

III. PREPARING FOR A FULL-APPLICATIONS PORTFOLIO

- . Why packages?
- . When to start?
- . Project boundaries.
- . System/software architecture.

IV. ORGANIZATIONAL IMPACTS

- . Projects.
 - . People.
 - . Level of support.
 - . Information Center.
- Copyright 1984, Baltimore Aircoil Company, Inc.

I. HARDWARE SELECTION

In April, 1982, BAC had a "31 Flavors" of hardware, including the IBM 4341, IBM 8100, IBM 3775, IBM System/34, DEC PDP-11, NCR, Fujitsu, WANG, Altos, Apple, and Radio Shack. The software was equally diverse. But, the mainstay was a 10-year old Honeywell package, converted to IBM, highly customized, sparsely documented, misunderstood, and impossible to support. With hardware costs running 20% over budget and data processing turnover very high, something had to be done!

With Top Management support (MIS Steering Committee--including the President, the Vice Presidents, and the MIS Director), a "no-holds-barred" hardware study was launched -- but we needed the answers fast! The key to a successful study has a series of requirements:

A. It must be timely;

data processing managers are not paid for attendance on the job; we are paid for results.

B. Yet, it must be comprehensive;

there is nothing more disheartening to Top Management than to have their Chief Computing Officer fumbling over seemingly simple questions.

C. The study's report must be understandable;

avoid jargon -- it hurts rather than helps.

D. Yet, it should have all the answers.

E. The report must be short and to-the-point;

remember, you are selling ideas, not writing a "Whodunit" novel.

Most executives do not enjoy reading and/or do not have the time.

Put all your ideas "up front" and follow that with as many attachments and exhibits as make sense. The result is that your ideas are read (first few pages) and your recommendations are less likely to be challenged. The report thickness suggests you have done your homework and has available detail for the Executive who may be interested in further details.

BAC's hardware study had two reports:

REPORT	# OF PAGES	# OF PAGES OF ATTACHMENTS/EXHIBITS
INTERIM	5	155
FINAL	3	160

Please note that over 90% of the attachments were photocopies of trade magazines and technical journals.

- F. Should cover a specific period of time with approximate time tables of events. For example, our hardware study considered a 5-year horizon.

The study took two months; the proposal was accepted in one Steering Committee meeting, and the result was a \$240,000/year reduction in hardware costs -- a whopping 40% cut. In addition, it positioned BAC with a viable hardware architecture capable of supporting applications into the 1990's.

The methodology employed was, again, SIMPLE. In a separate but related task, MIS was included in the Company's Long-Range Plan. These points (written by the MIS Steering Committee) are listed below:

- A. Adjust MIS hardware to a cost-effective combination of IBM 4341 mainframe plus remote units to obtain optimum systems for company needs.
- B. Upgrade and replace software programs using proprietary software packages as a timely and cost-effective means of providing properly functioning up-to-date systems.
- C. Provide the computing capability in mainframe, minicomputer, and microcomputer systems and an environment of availability necessary to support line and staff management decision-making needs.
- D. Improve office productivity by the introduction and use of modern, integrated word processing, mail, microfiche, and reproduction equipment and methods.
- E. Upgrade telecommunications systems to provide economical voice and data transmission of satisfactory quality.

With these "givens," the team began the study by developing a list of "business assumptions." The purpose of an INTERIM REPORT was to test these assumptions. If Top Management agreed, these assumptions would become functional requirements. Otherwise, they would be "adjusted" and the team would be given clearer direction.

The assumptions that became our functional requirements are listed below:

BUSINESS ASSUMPTIONS AFFECTING THIS STUDY

- A. That a computing environment will be required at plant locations encompassing most, if not all, of the following areas:
- . The printing of reports generated from Baltimore.
 - . The entry of data to be processed at the plant and/or Baltimore.
 - . QUERY capability against local data and/or corporate data.
 - . Word processing and electronic mail throughout BAC.
 - . Personal computing power equivalent to, or better than current microprocessor usage.
 - . Computer-Aided Design and/or Computer-Aided Manufacturing at each plant.
- B. That the equipment chosen provide leading-edge capabilities yet, is mature, stable, and well-received by the marketplace.
- C. That the vendor provides the reliability, support, and upgradability to avoid yet another equipment conversion during this five-year period.
- D. That the vendor be strong financially with a strong long-range strategy -- making for a viable long-range business relationship.
- E. That, if possible, each element in assumption #A can be run on similar equipment. In this way, start-up costs are minimized. Flexibility is provided later. As we continue to grow, we can choose between upgrading a particular machine vs. a second machine of like-size for redundant capacity.
- F. That packaged software be available, if required, to reduce the lead time on new applications while improving the overall system quality.

Based on the assumptions, the entire spectrum of commercially available computers were entertained. Research materials included:

- . Trade periodicals in the areas of Manufacturing, Engineering, Office Automation, and Data Processing.
- . Hardware/Software surveys and references (Auerbach and Datapro manuals).
- . Literature available through the American Production and Inventory Control Society (APICS).

Since BAC was forced to make a hardware decision prior to software, we leaned on package availability as the measure of machine flexibility and market acceptance. For the areas of Manufacturing, Finance, Word Processing, and CAD/CAM, we gleaned our research material for software packages and its applicable hardware.

As an example, ASK/MANMAN is one software package executable on two hardware offerings -- HP3000 and the DEC VAX.

In this manner, 43 manufacturing software packages were reviewed that ran on 66 different hardware selections. The "package" was defined in accordance with Ollie Wight's (a leading authority on manufacturing systems) definition of a "closed loop system." Hardware that accommodated less than 4 different packages (the average) was eliminated.

Financial packages were defined as an integrated offering of General Ledger, Budgeting, Accounts Receivable, and Accounts Payable. 85 Financial software packages were reviewed that ran on 139 different hardware selections. Hardware that accommodated less than 7 different packages (the average) was eliminated.

Listed below are the details behind package availability:

HARDWARE AND ASSOCIATED SOFTWARE PACKAGES AVAILABLE

Vendor/Model	# of Manufacturing Packages	# of Finance Packages	# of Word Processing Packages	# of CAD/CAM Packages
Apple	----	----	6	----
Basic IV	1	1	2	----
Burroughs	2	8	1	----
CDC	----	1	----	----
Data General	7	12	13	----
Datapoint	1	4	1	----
DEC PDP-11	6	19	10	9
DEC VAX	1	4	3	5
Honeywell	6	13	2	----
HP3000	10	16	3	11
IBM Series/1	1	3	4	----
IBM System/3	3	8	1	----
IBM System/34	3	11	6	----
IBM System/38	1	5	3	----
IBM Mainframes	12	14	9	4
IBM 8100	1	1	1	----
ICL	1	1	----	----
Microdata	3	4	----	----
NCR	----	6	3	----
Prime	7	2	3	----
UNIVAC	2	5	1	2
WANG VS	4	2	3	----

Note: With emerging software, this table becomes obsolete quickly. Yet, BAC should acquire proven products from proven companies.

Software surveys similar to Manufacturing and Finance were conducted for Word Processing and CAD/CAM. The results support prior findings -- that software vendors are gravitating to certain configurations.

In this fashion, over 80% of the hardware offerings were eliminated from the study. The table below shows the qualified vendors at this point in the study:

Vendor	Available Manufacturing Software	Available Financial Software
Data General	7 pkgs	12 pkgs
DEC PDP-11	6	19
Honeywell	6	13
HP3000	10	16
IBM 4341	12	14

Next, the Financial strength of the vendors was considered, along with projected volumes through the 1980's.

Strength and projected growth are important for two reasons;

- . suggests the probability of vendor support.
- . vendors of future software packages would be writing code for "the winners."

The table below shows the top 10 hardware vendors for 1981 through 1983 and projected to 1990, according to Datamation magazine. Notes to be made include:

- . The continued strength of IBM and DEC.
- . The growth of Hewlett Packard.
- . The elimination of Honeywell, NCR, Sperry, and Xerox from the top 10.
- . Data General is not on the top 10 listings.

THE LEADING HARDWARE VENDORS

-- DATAMATION MAGAZINE

Top Ten	1981	1982	1983 1990
IBM	1	1	1	1
DEC	2	2	2	2
Control Data	3	4	4	6
NCR	4	5	5	----
Burroughs	5	3	3	7
Sperry	6	6	6	----
Honeywell	7	8	9	----
Hewlett Packard	8	7	7	5
Xerox	9	10	10	----
WANG	10	9	8	4
Storage Technology	----	----	----	3
Electronic Data	----	----	----	8
AT&T	----	----	----	9
Japan, Inc.	----	----	----	10

Data General and Honeywell were dropped based on this financial data, narrowing the field to IBM, DEC, and HP. The final study phase covered the following topics:

- . CPU and terminal design.
- . Software
 - operating system
 - support software; viz., programming languages, communications facilities, and data base
- . Vendor strategies, personnel, and policies

Although the IBM mainframe proved to be an acceptable corporate resource, it appeared unworkable in BAC as a distributed processor, due to the personnel, environmental, and financial support requirements.

DEC vs. HP was a "win-win" decision; and we chose HP.

II. TELECOMMUNICATIONS STRATEGIES

Like most companies in the good old "pre-divestiture" days, BAC had a star network of leased lines and dial-up lines. Extending from Baltimore we had seven lines with the following destinations:

4	-	Northern California
1	-	Illinois
1	-	Ontario, Canada
1	-	Delaware

The equipment was exclusively BELL. Reliability was satisfactory, but the price was out-of-sight. (Of course, since 1/1/84, most data processing managers would pay a small fortune to achieve the reliability of "pre-divestiture" days.)

Alternatives such as Federal Expressed microfiche, MCI mail, and public networks were eliminated because these lines supported batch, mainframe applications.

With a target of cutting the phone bill as well as monitoring the network, Bell modems were replaced with CODEX equipment, including their DNCS (Distributed Network Control System). DNCS tells us, in advance, of potential line problems. We can relinquish a poor circuit to the AT&T Test Board and utilize CODEX's Dial Back-Up facility. The traffic of four former circuits, with a combined capacity of 26.4Kbs, are combined onto one circuit multiplexed at 12Kbs. In addition to better service, data communications costs dropped 30%.

An unanticipated challenge in the world of HP Telecommunications came in the area of remote high-speed printing. BAC's requirement was for 350+ LPM. The HP2608 printed at 400LPM in the computer room; but never reached much more than 200LPM when combined with MTS in the field. Prior to the introduction of the HP2563 and the HP293X family, we were in big trouble.

To the rescue came Digital Associates of Stamford, CT, with their Remote Line Printing System (RLPS). With one microprocessor, the RLPS converts HP print data from parallel mode to serial, compresses and blocks the data, and transfers it to the modem for transmission. At the receiving site, another microprocessor reverses the process to a DataProducts B600 printer. The HP3000 and the printer "think" they are in the same room while, in fact, they are miles away -- in some cases at BAC, as much as 3000 miles away. The following benchmarks pointed to 4.8Kbs as the ideal line speed for BAC.

Line Speed	Print Rate
9.6Kbs	600+LPM
4.8Kbs	600+LPM
2.4Kbs	200 LPM

Increased demands for information, changing applications, and additional sites will cause us to reconsider the Data Communications network.

The technologies to be considered for use by BAC over the planning period are already commercially-available. Influences that will set our direction and timing include:

- A. The declining cost of hardware.
- B. The declining per unit cost of public networks and satellite communications.

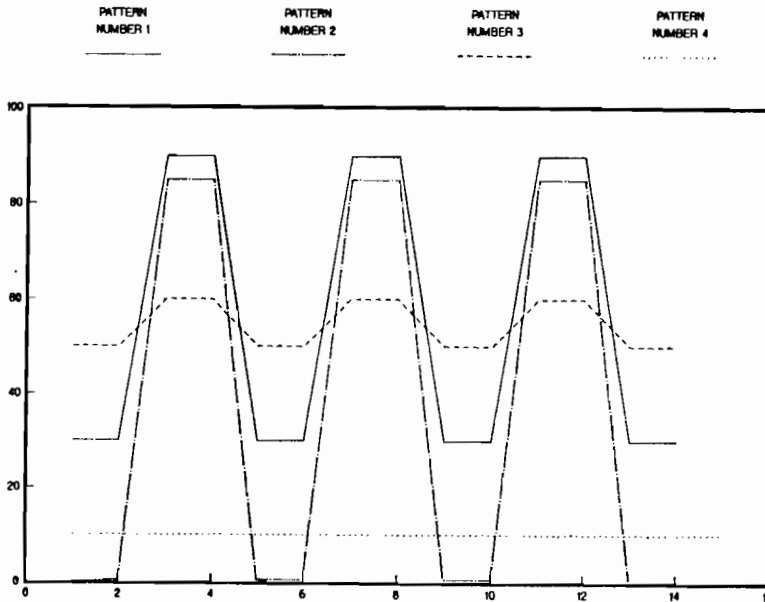
Higher volumes will allow vendors to lower their pricing. Volume increases come through:

- More information needs
- More companies going on-line, allowing them to enter the "per-unit" market
- Competitive pressures brought on by the AT&T divestiture

- C. The rising cost and declining reliability of our private network.
- D. The emergence of on-line applications that will reduce the volume of data transmissions.
- E. The proliferation of on-line applications beyond our North American plants into Company representative offices and International sites.

Listed below are Data Communications Patterns and typical solutions to address each pattern. The key to a successful, cost-effective network is the accurate measurement of data flows and volumes -- current and projected. This data must be compared with the fixed and variable costs associated with each potential solution.

DATA COMMUNICATIONS PATTERNS



1. Remote CRT's requiring full-time access to a computer coupled with periodic high-volume batch reports.
2. Periodic high-volume reports but no CRT data communication requirements. Data Entry on an intelligent terminal and transmitted in a batch.
3. Remote CRT's requiring full-time access to a computer with higher traffic than #1 due to on-line inquiry. However, high-volume reports are decreased because of these inquiries.
4. Very low-volume, sporadic usage.
 - . Planned example: Customer/Vendor electronic mail.

TYPICAL DATA COMMUNICATIONS
Solutions by Pattern

SOLUTION/PATTERN	1	2	3	4
A. Leased Lines Fixed Cost/Month	YES	OK	OK	NO
B. Dial-Up Pay by connect time ignoring volume	NO	YES	NO	OK
C. Value-Added Networks Pay by volume with slight connect time change	NO	NO	YES	YES

NOTES: 1A is BAC's current implementation to support existing systems.

2B is the planned implementation to support our remote HP3000's short-term.

2B, 3C, 4C appears to be the longer-range direction.

C could actually include A or B to connect a BAC site to a vendor's network mode.

C is commercially available today through MCI, AT&T, RCA, GTE, & GEISCO.

III. PREPARING FOR A FULL-APPLICATIONS PORTFOLIO

Purchased software packages are being utilized to address user needs. Proven packages can shrink lead times, reduce risks, and improve the quality over custom-designed systems. The cost of packages is also lower when considering the MIS cost of programming coupled with retaining personnel knowledgeable in our customized code.

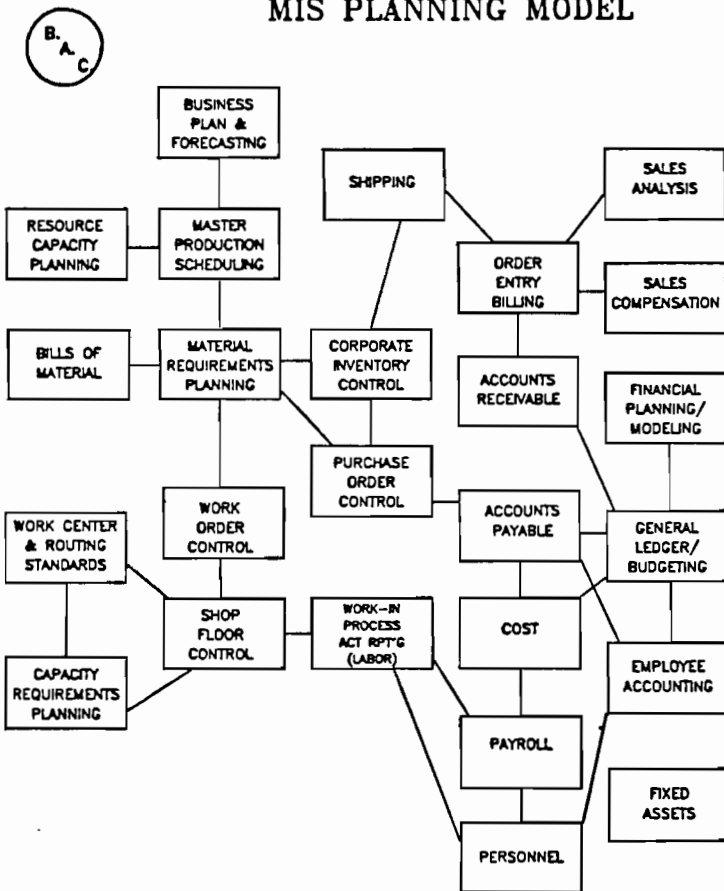
Package acquisition will follow a standardized methodology:

- A. Determine organizational needs.
- B. Define inter-organizational requirements.
- C. Select a leading package based upon match.
- D. Install the key system followed by satellite and other interfaces.
- E. Modify only if absolutely necessary.

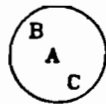
To determine where to start, the support of an MIS Steering Committee is, again, essential. The attached MIS Planning Model is a structured view of most commercial applications (some of the blocks may not apply at your company and some lines may need modification). Without getting involved in a discussion of structured analysis, imagine each line is a highway and each block is an intersection. If the MIS Steering Committee is the Traffic Planning Commission -- Where do they install the traffic lights?

With this simple analogy, General Ledger, Order Entry, and Material Requirements Planning became the priorities. Ranking these top 3 will be a function of each company's particular needs.

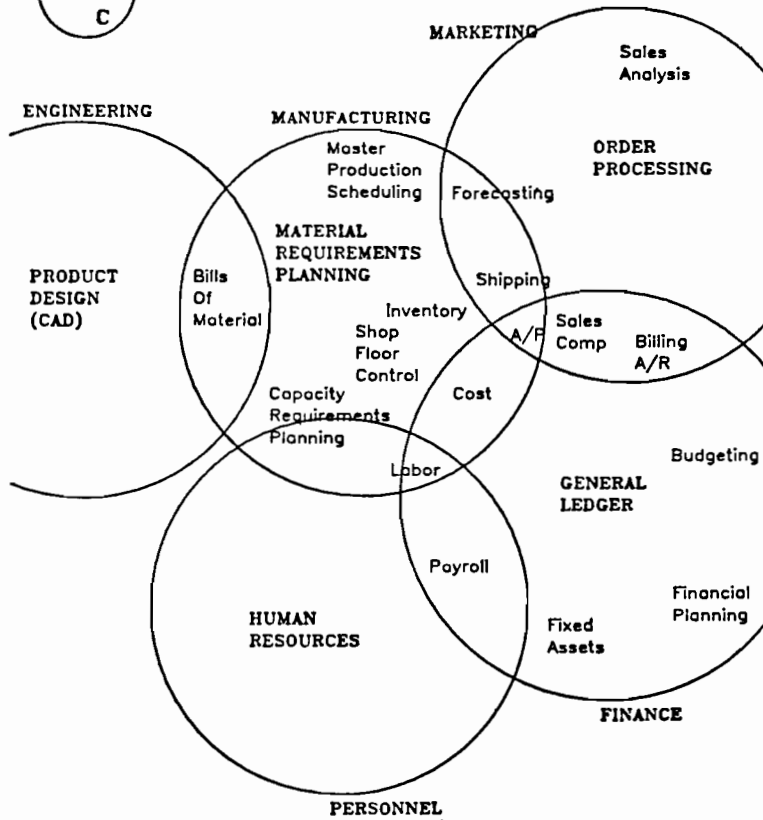
MIS PLANNING MODEL



In order to appropriately define project scope, a second Planning Model is required.



MIS PLANNING MODEL



UPPER CASE = KEY SYSTEM LOWER CASE = SATELLITE SYSTEM

Manufacturing, Engineering, Finance, and Marketing organizations, business flows, and impacts upon MIS are shown. Each organization has a key application system with many satellite systems. The path to success then is satisfactory integration, not only between key and satellite systems, but also key system to key system across organizational bounds.

For example, the key Finance system is General Ledger with Billing/Accounts Receivable considered a satellite system (see table below). Order Processing is Marketing's key system with Billing/Accounts Receivable considered a satellite. A successful implementation would involve the ability to integrate Order Processing through General Ledger. When defining system requirements, key systems, satellite systems, organizational and MIS interfaces must all be considered.

Any leading package may address 80+% of your organizational needs. If not, perhaps you should ask questions about your own company's policies and procedures -- not a vendor's functionality. No package will answer all the questions. Report Writers and applications generators come into play to augment the purchased logic. At BAC, the COGNOS product line has been very successful and tools like COBOL and FORTRAN are swiftly becoming antiques.

MARKETING		FINANCE
ORDER PROCESSING	Billing	GENERAL LEDGER

	Accounts Receivable	

Key systems are listed below by organization.

ORGANIZATION		KEY SYSTEM
Manufacturing	-	Material Requirements Planning
Engineering	-	Computer-Aided Design
Finance	-	General Ledger
Marketing	-	Order Processing
Personnel	-	Human Resources

IV. ORGANIZATIONAL IMPACTS

Many of the directions set forth in this plan have MAJOR implications to the kind of services coming out of MIS, as well as the people that both provide and utilize that service. These are highlighted below.

- A. Software -- The software packages described in this plan will reduce lead times for projects; but will require stronger commitments, project leadership, and up-front analysis to achieve success.

The success of "package projects" suggests MINIMUM package logic changes. The successful organization will change its procedures to match the package.

- B. MIS Personnel --

Clerical - Shrinking Data Entry
- Higher-caliber Computer Operators

S & P - Less emphasis on design and coding
- More emphasis on analysis and interpersonal skills

Information Center - A new entity requiring strong technical and communications skills

Overall - Quality will improve with sound hiring and training practices

- C. User Skills -- More is expected than ever before.

- D. Media -- Batch reports and microfiche/film will be replaced by on-line inquiries via CRT's.

- E. The level of MIS Support must change as described below:

1. Areas of Full Support

- a. Development of new systems, such as A/R, A/P, G/L, etc., will be fully supported. MIS will work with a user project leader to determine and meet system requirements.
- b. Maintenance of installed systems will be fully supported. As our number one priority, when a system breaks down it will be repaired. In addition, as business or legal requirements change systems will be changed appropriately.

Note: Normal business changes to price, product, territory, etc. will be a user responsibility. If programming could aid in large-volume changes, these will be addressed on an individual basis.

- c. Enhancing installed systems under certain circumstances will be fully supported. If input procedures, storage requirements, or modifications to purchased software is required, MIS will fully support these projects.

New reports and changes to user-written reports will receive consulting support only.

- d. Data Center Operation and Security will be fully supported.
- e. Security will be a growing concern as more on-line systems are employed to manage the business. Very specific guidelines and policies for compliance have been generated due to the high risk of lost or stolen data. Areas that are to be addressed include:

- . Controlled physical access to the Data Centers.
- . Controlled logical access into the HP3000 computers. This will be accomplished in three layers:
 - Security access onto the computer itself.
 - Security access into a particular application.
 - Security access into certain fields and transactions.

For example, a password is required to log onto the HP3000. Another password gains access into MANMAN. Then, only Engineering is allowed to change Bills or Accounting to change costs.

- . Data backed-up/archived in three layers:

Media/Location	Potential Problem
Tape/Computer Room	Disk Down
Tape/On-site Vault	Computer Room Disaster; e.g., fire/water damage
Tape/Off-site	Building Damage

- . The Data Center backed-up in two layers:
 - a. Preventive - including specific fire & power protective devices.
 - b. Recuperative - in the event the Data Center is damaged.

2. Areas of Partial Support
 - a. Enhancements as described above.
 - b. Information Center. The tools, training, and consulting will be provided to support the office systems previously described, along with any user-developed applications not considered a business application. Examples in this area include the many Engineering routines developed by Engineering for product design, selection, and optimization. The role of MIS training is described below.

The scope of MIS training will change significantly during the Plan period in response to internal and external forces associated with computer technology. There will be an increased level of computer-based technology throughout the Company. User staffs will require training in order to understand how to apply and use new office technology and productivity tools. MIS staffs will likewise require more training in order to be kept current with new applications development technology that will be brought into the Company at a more rapid rate than in prior years.

The goals of the MIS training program are:

- . To improve the ability of MIS to design, develop and implement high quality information systems for the Company through the maintenance and enhancement of state-of-the-art technical skills within the staff.
- . To provide the necessary basic skills for the users to effectively utilize the MIS supported data processing environment.

Non-professional trainers will be called upon from all MIS staffs, as well as user areas, to be teachers in subject areas where they are resident experts.

It is expected that MIS training will become part of a broader program of human resource management within MIS during the plan period. This will involve a more formal effort in the areas of skills planning, career development, performance management and recruiting and hiring.

- c. For Personal Computers (PC'S), selection and security guidelines will be provided, along with communications interfaces to the HP3000 if appropriate. Support above that will be very limited.

3. Areas of No Support

- a. Data Entry will be phased out as we migrate to on-line systems. At system start-up, data entry support may be available - to be addressed on an individual basis. Once installed, data entry will be a user responsibility. Peak loads can be addressed with temporary clerks within a user's budget.

No MIS headcount is to transfer to a user group. As discussed in the Personnel Strategy, this resource will be re-applied within MIS to achieve this plan. Any data entry load being absorbed by a user group should be outweighed by project benefits OR the project should not be entertained.

- b. Reports/Special Forms - As on-line systems are installed, the Company's reliance on paper will shrink. Low-cost, low-speed printers will be installed in user areas for the small listings still required. Larger volume reports will continue to be printed on centralized, high-speed printers and available for user pick-up just outside the secured Data Center.

Special forms will be printed on the user-area printers. As such, the stocking, reordering, and payment responsibility will remain with the user, although form design will be supported by MIS as part of System Development.

Newer technologies, such as lasers or inkjet, may provide further cost savings.

V. CONCLUSION

Baltimore Aircoil Company, Inc. is well underway in its commitment to improved productivity and Company profitability -- with MIS playing an integral role.

Good people using good tools is the deciding factor of our progress. We are proud of our people and pleased with our HP3000 - The Gateway To Success!

ABOUT THE AUTHOR

John Skrabak is MIS Director for Baltimore Aircoil Company, Inc., in Baltimore, Maryland. In this position, he provides MIS strategic planning and directs Systems Development, Telecommunications, Office Automation, and Data Center activities. He assumed his current position in April, 1982. Mr. Skrabak received a BS degree and MBA from Drexel University. He is a Certified Data Processor (CDP), as well as a Certified Production and Inventory Manager (CPIM).

3023. THE USE OF IMAGE TRANSACTION LOGGING IN A MULTI DATA BASE,
MULTI MACHINE CONFIGURATION TO ACHIEVE A NON-STOP, FAULT
TOLERANT HP3000 SYSTEM.

Roger W. Lawson
Proactive Systems Ltd
110 New Bond St, London
Tel: 01144-1-408-1601

INTRODUCTION

The continuing success of many organisations depends on 100% availability of their computer systems. Many applications such as sales order processing demand that the computer system is never out of action for more than a few minutes if customer service is not to be seriously disrupted. Many HP3000s are also used in organisations where the systems must be available 24 hours a day for 365 days a year (for example in shipping ports, airline offices and similar areas). Moreover as computer hardware becomes more reliable the tendency is to design application systems that depend on that reliability - thus systems that a few years ago might have been implemented by a batch processing approach will now tend to be on-line updating systems. This paper discusses the problems of implementing high reliability systems in an HP3000 environment and describes the approach that was taken to solve these problems in several installations with which the author has personal experience.

PROTECTING YOUR DATA BASES USING TRANSACTION LOGGING

Although an HP3000 computer is now very reliable (for example the CPU is likely to fail less than once per year) it still does not match the need of certain applications. When it does fail it may still take many hours to repair. Also as most users implement their systems using IMAGE, even if any failure is only temporary the IMAGE data bases may be corrupted.

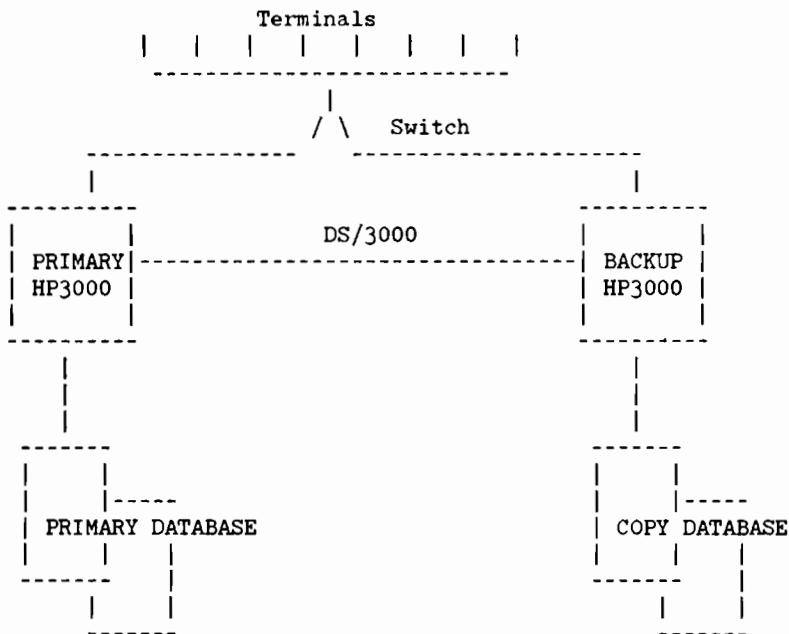
The use of transaction logging is the only absolutely safe way to protect your data in an on-line updating environment. Unfortunately it has not proved to be very popular with HP3000 users to date for the following reasons:

- a - It is often thought that it will seriously impact system performance when in practice most users would not notice the overhead.
- b - If you log to a serial disc or tape (which are the only safe methods) then you either need a spare disc drive or you have to dedicate your tape drive to logging. There are few installations who do not occasionally use their tape drive during normal working hours so either you have much inconvenience or you need a second tape drive.
- c - The erroneous belief that transaction logging is difficult to understand and to implement.
- d - Even if you use logging, when you have a failure and do a roll forward recovery it takes much too long. For example on a system with a moderate transaction load if you logged from 9 am to 2 pm and a failure then occurred, by the time you have restored the data base and rolled forward then the normal working day would probably be over.

The most common reasons for breakdown are a system failure, power supply problem or temporary disc fault. Most users who have run high availability systems have found that in most cases they can restart the system and the chance of having a badly corrupted data base is small - you can always patch it up later and this may be preferable to having top management shouting at you because the system is out of action - don't try explaining why you need to spend 3 hours doing a roll forward recovery to the company president in this situation! Now thankfully HP have released Intrinsic Level Recovery and Roll Back Recovery (with Turbo Image) to solve the last problem so that transaction logging is certainly now more attractive. However that still leaves other types of failure such as disc drive hardware problems requiring roll forward recovery. Moreover transaction logging does not solve the major problem that if you have all your processing dependent on one HP3000 and it breaks down with a major hardware fault then you have no user system until the machine is repaired.

PROTECTION AGAINST HARDWARE BREAKDOWN

There are several HP3000 installations who have considered the current limitations of hardware and software and taken a new approach. Obviously the only possible way to have a fault tolerant HP3000 system is to have more than one processor and to mirror the disc files in some way on the second system (this is how TANDEM and the other non-stop system suppliers attempt to produce such a system). Of course many HP3000 users already have more than one machine linked by a data communications line and DS3000. So the solution that has been arrived at by several people can be represented in the following diagram:



Any transactions relating to the primary data base are passed down the DS line and used to update the mirrored copy of the data base on the second HP3000.

This approach can provide:

1. Minimum down time. If the primary HP3000 breaks down then terminal users can be switched over to the secondary system in seconds and they can continue immediately on an up-to-date data base.
2. Zero down time for data base back-ups (eg DBSTOREs) on the primary HP3000 - they can be performed on the secondary machine.
3. Performance improvement. Read-only programs (eg. reporting systems) can be off-loaded from the primary system to the secondary.

A POSSIBLE SOLUTION

One solution that has been used by a major HP3000 user to implement the above is to replace the IMAGE calls in application programs so that data base modifications are made on both primary and secondary systems. The problem with this approach is that there will be a major performance impact on the primary system (the primary system has to wait for the remote IMAGE processing to be completed for each physical transaction before it can proceed). Also the problem of logical recovery of either data base if a failure occurs is not at all straightforward.

A BETTER SOLUTION - HOW IT WORKS

The major technical problem in achieving a mirrored data base is that of picking up the changes to the primary data base, passing them down a DS line and updating them in real time on the second machine. Also if the primary system fails or the link fails then one must be careful that only complete logical transactions are processed on the secondary data base (any logical transactions which have been only partly passed across the link must not be processed otherwise the secondary data base will not be logically correct).

The chosen approach to this of several users and software suppliers (including my own company in it's own software product in this field) has been the same. The approach is to use transaction logging on the primary system to create a log file on disc. The log file records are then copied immediately down the DS line and used on the second system to update the secondary data base. Note that this updating is done in the same manner as the DBRECOV program operates so as to ensure that logical transactions are treated properly. Effectively you need a record transport process running on both systems and a data base updating process on the secondary systems (these are typically programs running in background batch mode similar to the way HPMAIL is run). If there is a temporary failure of either HP3000 processor or the communication line fails then the processes that run on each system must be capable of automatically realigning themselves when the systems are restored so that transaction records are neither lost nor duplicated.

You also need control functions so as to be able to start/stop or otherwise manage these processes. For example on the secondary system you may want to take a back-up copy of the data base without halting the primary system (or risking transaction loss if the primary system fails during back-up). Therefore a command process exists which communicates with the updating process on the secondary system via a message file which instructs it to simply hold the transactions on disc until the data base back-up is complete.

PERFORMANCE IMPLICATIONS

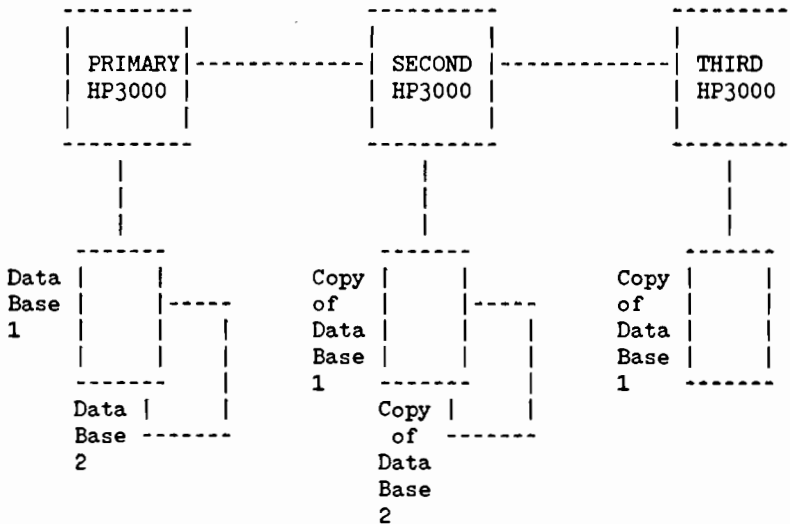
What is the performance impact of such an arrangement? On the primary system there is the overhead associated with the use of transaction logging (which, as already mentioned, is small). There is the impact of DS on both systems. On the secondary system the load can be very small as the data base updating can be run as a low priority background activity (it does not matter if it occasionally falls a minute or two behind the primary system at peak transaction loading - the peaks rarely last long). If the secondary system is dedicated to the back-up activity then it can be a small low-cost HP3000 - it does not need to be as powerfull as the primary system.

As already mentioned it is possible to transfer low priority data base activity such as reporting from the primary to the secondary system (you can also off-load such activities as program development).

Therefore a dual system can considerably improve the response and throughput on the primary system.

A DISTRIBUTED DATA BASE

The same scheme that is described above for one data base on two machines can be extended to any number of data bases on any number of connected HP3000s. The following is a diagram of such an arrangement:



Copies of a data base can be maintained on multiple HP3000s and they will always be kept perfectly in step. In addition it is possible to simply maintain a particular data set rather than the whole data base if that is preferred by being selective in the records taken off the primary log file. By using such tools it is possible to set up and maintain a truly distributed data base system.

MORE FLEXIBLE LOGGING

In the implementation by my company of the above scheme we have also provided for logging to a remote system if the user does not wish to maintain a copy of the data base in real time. In addition the log file is switchable between devices (both locally and remotely). Note that unlike the CHANGELOG implementation which results in fragmented log files, in our system you can switch from tape logging to disc logging

and back again and the transactions temporarily written to disc will then all be written out to the tape - this is much more useful than CHANGELOG if you simply wish to temporarily use your tape drive for some other purpose. These extra facilities are easy to add if you have the log file transport and control mechanisms.

DISASTER PROTECTION

It is worth pointing out that the secondary system does not necessarily have to be back-to-back with the primary system. It can be remote with the DS link supported by a modem line. Note that a 9600 bps modem link will support quite a high data base transaction volume - the link does not necessarily need to support a peak transaction load. By locating your secondary system some distance from the first you can protect against the risk of fire, flood or other catastrophe. If your terminal users are on the primary site and the primary processor breaks down then you can switch over the DS line to support at least some of the terminal users via a stat-mux on that same line using the same modems.

CONTROLLED OPERATIONAL CYCLE

If you have used transaction logging already then you will realise that although very simple to implement, it does impose some extra operator responsibilities. Logging has to be stopped and started at times appropriate to when data base back-ups are done. Multiple data bases that are logically related have to be kept on the same logging cycle. This is even more important when transactions are being transmitted from one HP3000 to another and where multiple copies of the same data base are being maintained. For this reason we implemented our system so that the back-up/logging cycle can be pre-set with minimal operator intervention. The details of the data base mirroring and/or logging are stored in a configuration file which can be easily updated by the use of a configurator dialogue program. So as to maintain ease of operator control we have simply extended the MPE logging commands (eg. :LOG) with additional options where relevent - this is done by overriding the MPE commands by a UDC command that runs a program.

TECHNICAL DETAILS

Our implementation of the above system was written in SPL. No PM features are used and it works on any version of MPE so that the the HP3000s it runs on can be on a mix of operating system fix levels.

CONCLUSION

By using the techniques described above it is possible to configure a dual-processor HP3000 system that is cheaper than a TANDEM or STRATUS system. The HP3000 application software does not require changing in any way to take advantage of this method. Of course if you already have more than one HP3000 as many users do then it is an elementary matter to install such software.

Moreover the use of this type of software is now becoming well established. For example I know of at least two users who have been running this software for over 18 months. In one case a disc hardware failure occurred on a production HP3000. The company operated a round the clock work schedule and could not sustain down time of more than a few minutes without massive costs from loss of business and disrupted work arrangements. The terminals were switched to the secondary processor and the users were soon able to access the "current" data base - total down time was less than 5 minutes, a saving of nearly 4 hours over their previous method of data recovery.

About the author:

The author, Roger Lawson, is the Managing Director of Proactive Systems Ltd - a software company with offices in London and New York. He has a first degree in Production Engineering and a Masters degree in Business Administration. His work experience includes over 10 years of using HP computer systems in a commercial data processing environment.

3024. THERE'S GOT TO BE A PONY HERE, SOMEWHERE

M. G. Cornford
Manager, Data Communications & Operations Support
Northrop Corporation, Electronics Division
Hawthorne, California
(213)600-4217

This paper is about the experiences of a medium sized user dealing with a computer network that includes several HP 3000's plus CPU's, disc drives, and tape drives from several vendors. It primarily concerns the use of PBX's and LAN's and other communications systems. It is a story about the availability of lots of words and little data.

HOW DID A NICE GUY LIKE ME GET HERE?

I've always been a person who looks on the bright side. I discovered and practiced the idea of positive thinking long before I ever heard of Norman Vincent Peale. I think of myself as a naturally optimistic person.

You know what an optimist is, don't you? Let me tell you about the wealthy family that had two sons. One of these boys couldn't be satisfied no matter what. His motto was "every silver lining is surrounded by a dark cloud." The other kid was happy and carefree, no matter what. The father recognized that neither of the kids was properly prepared for the real world that they would have to face eventually. He called in a behavioral scientist - a shrink, if you will - to try to temper the attitudes of these two boys.

They took one of their rooms and filled it up with every conceivable type of toy: computer games, electric trains, everything. They put the pessimist into this room and shut the door. In another room they spread about a two-foot layer of horse manure and left the shovel propped against the wall. They put the optimist in this room and shut the door.

After a couple of hours the behavioral scientist and the parents looked in on the kids. The pessimist, sure enough, had cleared a corner of toys and was sitting there crying, facing the corner. He wanted to read a book, and there weren't any books.

When they opened the other door, they saw horse manure flying every which way. The optimist was shovelling away, whistling happily. When they asked what he was doing, he replied, "With all this manure, there's got to be a pony here, somewhere."

I will confess right at the start of this paper that I got into this whole thing by accident. Soon after I transferred into our

data processing organization, my boss forgot the reason for the transfer. He had had a rather simple installation with a couple of HP Series III CPU's and 50 or 60 terminals attached to the computers by cables running through the area in the ceiling above the acoustic tiles. A tape unit and a few disc drives attached to each unit completed the scene. There were dark clouds on the horizon, though; a Series 44 was on order with another bunch of terminals. Would I please help him out by taking the responsibility for this new installation? I picked up my shovel and started in.

I was awestruck when I attended my first users' group meeting. All of these people around me knew so much and I knew so little! I laughed politely when I heard Ross Scroggs say that the paper he was presenting on interfacing of peripherals contained no facts. As he went on in the next 45 minutes to dazzle me with all sorts of brilliant observations about data communications and hooking things up to Hewlett Packard computers, I was impressed!

After listening to one after another of these speakers with so much knowledge and so much experience, I was wondering how I had ever gotten into such a position. Being the optimist that I was, I grabbed the shovel and started scooping.

In the years since, I have been responsible for obtaining and installing several more computers and many more terminals. All of these things began to be hooked together and get so complicated that it was difficult to keep track of everything. We began to find that we had an awful lot of wires running everywhere in the facility. Suddenly there were a lot of pieces of equipment that had to be running reasonably well or the telephone started ringing with a voice on the other end saying, "My computer doesn't work." The potential for failure rose in a geometric progression.

I have had to hire people to help me because the whole thing just kept getting further and further out of hand. When it came time to listing the qualifications that I wanted in these people, I ran into problems. Naturally, I wanted these people to be experienced in the field of data communications. I wanted them to know all about how to hook up various pieces of equipment so that they all ran and did what we wanted them to do. I wanted people who were results oriented and didn't know the meaning of the word "can't". I had the perfect job openings for optimists.

Being an optimist qualifies one to work in the field of data communications more than any other single factor: more than educational background, more than experience, more than specific knowledge. In fact, as I proceeded on this quest for people and this quest for answers to my many questions, I discovered a very important fact; no one knows anything about data communications!

There is no useful knowledge of any kind in the field of data communications. If there were, would we have the multitude of

terms such as space, mark, odd, even, 0's, and 1's for parity? We're talking about a simple one character field that can only contain a 1 or a 0! Why else do you believe that IBM, with all of its store of highly trained and experienced people, can't make a simple decision as to what sort of technique to use for networking, coax or twisted pair, packet switching or something else?

In this field, the real expert is the person who can keep track of whether he is dealing with a DCE or a DTE. The best experience that one can look for is the ability to use a breakout box to solve an interface problem. I hope you all know what a breakout box is. It's a small box, usually blue, that data communications people don't go anywhere without. It has little lights that don't mean too much. The most important things that it has are a bunch of jumpers and switches. Using these you can keep trying different things until something works! (It should be called a TAE box for Trial And Error.)



WHERE AM I, ANYHOW?

Before I go too far, I had better set some boundaries on that part of the field of data communications that is being addressed in this paper. There is, of course, a field of data communications that addresses itself to the flow of information within the computer itself. There is another world that addresses itself to "the big picture", the world of protocols and specifications and the such.

Then, there is my world of data communications which is that world that must address itself to providing users with a needed ability to get the computer to do something that they can't do very well without it. This is a world that never has enough time or enough resources but is long on needs. It is a world that doesn't always have time enough to plan ahead more than days or weeks. It is a world in which "it can't be done" is an unacceptable answer.

My world is concerned with terminals and printers and computers, disc drives and tape drives. These are the facts of the equation; I wish I could say they are the constants of the equation, but they change too much. My world also contains cables, modems, multiplexers, switches, controllers, PBX's, and LAN's. These can hardly be called facts; about the time you think you can do something with them you find that all of the things you thought you knew just aren't so. Yet, these are the only tools you have and they must be manipulated and twisted until a solution is found to the problem.

This is a world of contradictions: there are no answers; there are a lot of answers. All of the answers are right; all the

answers are wrong. You can be certain that about the time you get a problem resolved, you and everyone else will have a crystal clear knowledge that there is another, simpler answer.

My world is a world of lots of things. There wasn't much need for a person like me when computing was simply one computer with its tape and disc drives plus a few terminals. I became needed when that world expanded to include a number of computers, a number of disc drives, a number of tape drives, a number of facilities, a number of printers, and a whole lot of terminals.

My world is a world of frustrations, both to me and to those who are calling on me for solutions. I am constantly being asked to tell someone the day and hour when I will have a solution for a problem that still isn't even defined. I must have an installation complete within two months even though Hewlett Packard isn't able to guess when they will have those products that are needed to solve my problems.

The one word that best describes my world is "networks." A collection of components becomes a network when they can be made to work together in an intelligent and efficient manner. An expectation is that the network will perform significantly better than the pieces did before they were integrated with each other.

Early in this paper I said that no one knows anything about data communications. You could very well ask, then, why I am presenting this paper. That is the reason! I am presenting this paper because there is an appearance that there is a lot of knowledge, lots of facts, and simple solutions available for the asking. These things simply aren't true. I believe that one is far better prepared to deal with an uncertain world when he knows in advance just how uncertain that world is.

This paper, therefore is a sharing: a sharing of experiences, a sharing of trials, a sharing of successes, a sharing of failures. Most of all, it is intended to be a sharing of goals and hopes.

By now you must have observed that I speak from the view point of a person who tries to put together a bunch of components in such a way that we end up with something that performs useful work closely approximating that which we want done. Because of that, I take a simplistic and limited view of knowledge. I don't care whether or not knowledge really exists unless that knowledge is useful to me. By useful I mean that it must have the ability to provide answers that can be applied to the solution of real problems in a current time frame.

LET'S PUT THINGS INTO PERSPECTIVE

In order to give perspective to our discussions of data communications networking, the following is a brief overview of the data center that we now have.

We have put together a system of computers and peripherals. The system works. The computers run and they produce the data that is expected of them. Most of the time the people who operate the terminals get connected with the computer. At first glance, it would seem that my definitions for knowledge have been satisfied. I feel, however, that we are still far away from real solutions. No one is satisfied with the limitations on what we can and can't do.

We began our experiences with HP computers when my company embarked on an ambitious plan to modernize our manufacturing and material control procedures. By the time the first part of this program was implemented six years ago, we had two HP3000 Series III's with 40 terminals. Five disc drives stored a maximum of 390 Kb of data. Each of the Series III's had a low speed, medium density tape unit. One of the CPU's was used exclusively by the analysts and programmers who were enhancing and maintaining the system, and the other was the production machine which ran the system itself.

Over the six years since this initial implementation, this system has undergone considerable expansion. There have been a number of significant program expansions, and the data bases themselves have grown several orders of magnitude as a result of an explosive growth in company business. From that original system of two HP3000 Series III's with 40 terminals, this manufacturing system has grown to where it now utilizes one Series 42, one Series 48, and two Series 68's with a total of about 320 terminals. It requires 17 disc drives with a total capacity of 5724 Kb. Backups have required that we utilize the highest speed, highest density tape units available.

Parallel with these developments, the organization has picked up added responsibilities for word processing, computer graphics, financial data processing, and program management support. Along with these responsibilities an IBM 4341, an HP 2680 laser printer, and three Star "supermicro" computers have been added to the computer room, and the Series III has been utilized to support non-manufacturing programs.

Physical locations have expanded from a single building to four separate facility complexes separated by as much as five miles. The work schedules have expanded from a single shift, five days a week, to three shifts, virtually seven days a week.

With the physical expansion came a new telephone system, the Northern Telecom SL-1 digital voice/data system. In our planning to that point, we had decided that the PBX was to be our choice

for a networking tool, so it took very little arm twisting to get us to embrace this new system. More about this later.

Peering into the future, we find no respite from this pattern of continuing growth and expansion. Production rates are scheduled to multiply significantly over the coming months. Manufacturing activities at a remote plant are scheduled to increase dramatically and to make use of the automated systems that are currently in use in our Southern California plants.

We can predict considerable increase in usage of the data from this system as others in the company realize that the system contains data that they need in the performance of their jobs.

Office automation is a very real factor in our future; it is virtually non-existent in the present. Personal computers are just beginning to be a factor in our environment. While it is known that distributed processing must be incorporated to a far greater extent than it is at present, severe software problems exist.

PLANS AND ACCIDENTS

I am not going to pretend that we have done everything right. We didn't have time to do that even had the knowledge and the facts been available. There is a saying that I am sure you have heard all too often about not having time to do something right but always having time to do it over. We haven't had time for either, so anything that was done wrong but works anyhow is still wrong.

We are where we are as a result of a combination of planning and accident. Planning made clear our needs and the path that we wanted to follow to satisfy those needs. Finding those tools that made possible the satisfaction of any of those plans can only be classified as accidents. Properly combining these two elements, planning and accident, is exciting and fruitful; I commend it to all.

Growth that has been experienced has, for the most part, been forecast and planned for. We have known that there will continue to be many more users on our network in the future. We know that our computer system will more and more involve multiple CPU's in the storage, updating, retrieval, and reporting of information. We know that there will be networks within networks. It would be foolhardy to say that we know how all of these are going to be satisfied. That's where we must rely on accident.

Growth in number of users presents multiple problems. There is, of course, the simple problem of providing the cables or other

hardware for connecting all of the desired users to the computers. There is then the problem of providing computer ports for the users. Lastly, there is the problem of controlling and servicing this network of users.

Interconnection of computer systems and the use of multiple CPU's within a single process or transaction presents software, hardware, and networking problems. While the principal intent of this paper is to concentrate on networking, it will not be possible to avoid discussing other aspects.

In attempting to find ways of handling the growth of our computing system, we have looked hard at all suggested methods of networking. We knew that we had to have the most efficient possible method for adding users to our network. We also knew that a continuing relocation of terminals was a fact of life with which we would have to deal. We needed methods for passing data between computers. We need solutions that will permit growth of the overall system without degrading performance. Above all, we need methods that will come as closely as possible to providing our users their 24-hour, 7-day availability of the processors for on-line activity.

The PBX vs the LAN

In searching for answers to the problem of connecting terminals to the computer, our attention kept getting drawn to the telephone cabling plant. After all, there is a great deal of similarity between the telephones and the data terminals. In their simplest terms, each performs the same tasks: they send and receive information between themselves and other devices at a distance from themselves. Performance objectives are also virtually the same: data must be passed accurately and with a high degree of reliability by the system.

When we first began looking at the cabling plant, the biggest obstacle to its use was the fact that it was installed, operated, and controlled by the telephone company. The telephone company has never been particularly thrilled to have anything connected to their system that they didn't make, own, and get paid each month for its use.

Even before we installed the Northern Telecom voice/data digital switching SL-1 system, we had looked at other systems which would allow simultaneous traffic of voice and data messages on the same wiring. Systems were on the market which permitted data terminals to be connected to the telephone circuits and which provided the ability to separate the data traffic at the main switchboard.

We had been watching the progress of Local Area Networks for several years. These had been widely touted for their capability of transferring data at high rates between computers and between terminals and computers. The proponents of these systems were

confident that they could compete with the cost of twisted-pair wiring, but we never took that too seriously.

My main hang-up in using LAN's for terminals, work stations, or personal computers was the problem of wide-spread installation of coaxial cables. We had had all of that experience that we wanted in connecting IBM terminals to their controllers with coaxial cables. In our new facility, all wiring must pass through floor troughs. Openings to these troughs had razor sharp edges that had stripped insulation from more than one wire.

We also found that mice or rats apparently used these troughs as a sort of playground. With an apparent shortage of other food, they turned to our coaxial cable, eating the insulation as well as the braided wire shield. I thought my technician was kidding until he showed me the tooth marks on the insulation.

Of course, the principal obstacle in the use of LAN's with the Hewlett Packard or IBM worlds was the lack of communication software that would support communication with LAN's. (Until HP released their LAN earlier this year.) Even the third-party LAN vendors wanted nothing to do with the HP systems. Most of the suppliers never returned or called again after assuring me that they were almost positive that some of their customers used HP computers and they would get right back to me with more information.

The advantage that had been quoted for the LAN was its ability to pass data at much higher bit rates than twisted-pairs could handle. The RS-232 specification limits communication rates to 20,000 bps, and HP's DS3000 communications techniques limited communications to 56 Kbps. LAN's, we were told, could handle up to 50 Mbps; well 10 Mbps, anyhow.

What does this mean, however to the average terminal user? Do a little arithmetic assuming a typing speed of 60 5-letter words per minute and you will find that 50 bps will handle the load quite well. A fast 180 cps printer will only require 1800 bps. HP's terminal controllers don't handle more than 9600 bps even on the return. It would be startling to have a whole screen painted in 1/500 of a second, but I doubt if any real improvement in productivity would be achieved. File transfer to PC's may lend weight to the use of a LAN for connecting PC's to the network should that really become desirable.

After considering all aspects of the data communications world within existing technologies, we based our planning on the use of PBX technology for handling communications for terminals and work stations. Up to now, we still believe that the LAN may have a place in the field of communications between CPU's or between CPU's and discs or between CPU's and tape drives or laser printers.

Matrix Switching and Port Contention

We also could forecast that computer hardware limitations and sheer economics would not permit a continued luxury of having a computer port for each terminal. As terminal prices continue to fall and as labor costs rise, it becomes cost effective to have terminals more liberally distributed. In fact, we forecast now that there will be some sort of a terminal on most desks in our facilities by the end of the decade.

Some form of port contention will be necessary to provide efficiency in the operation of the computers. The average terminal user other than the full-time data entry user, will undoubtedly never achieve greater than 25% usage of a terminal. This would say that a minimum ratio of 4:1 for terminals to computer ports would be cost effective. I'm sure that experience will raise this minimum to a ratio of 8:1 or 10:1 without interfering with user productivity.

Another need that is rising in importance is for a user to be able to work with more than a single CPU. Our IBM computer is used primarily for accounting, labor and payroll functions, personnel transactions, cost estimating, and for maintaining a number of data bases for drawing release, cost estimating, reliability, logistics, work standards, and others. Several of the HP's operate the manufacturing and material system while others are used for graphics, word processing and other data base systems. The Star computers are used for program planning and control.

Even a cursory thought about these needs indicates a desirability for interaction between these systems. Labor collected on the HP system must feed the payroll and timekeeping system on the IBM. Procurement data in the HP system is important to the cost estimating system on the IBM and to the program planning and control system on the Stars. Needs for interactive access to more than one CPU at the same time are outpacing any plans for hardware and software development to support these needs.

The use of some sort of matrix switch whereby a user can easily sign on to any selected CPU would be far better than using DS facilities. Such a switch would also make it easier to switch "home" CPU's than using patch cords or shifting of cable connectors. Some matrix switches also can handle the port contention problem.

I have mentioned previously that one of the needs that had been set forth by users was to have 24-hour, 7-day availability of the computer for record updating, interrogating, and reporting. No time was permitted for mundane chores such as system backup or servicing. Our programming staff is looking into a relatively new software package now called Silhouette that would permit us to maintain identity of data sets on two different CPU's at all times.

Assuming that all of the problems can be resolved with the use of the software and that this solves the backup problem, there is still a very real problem of service and maintenance work. Even HP products do occasionally fail, and they do need preventative maintenance as well as time to load new software changes. Silhouette makes possible the use of the alternate, or slave, machine during those periods so long as users may be switched as required. A matrix switch would allow such changes of connectivity to be done programmatically.

The PBX is a form of matrix switch. It can handle port contention within limits. Terminals have the ability to dial into any CPU. In a rather small installation, this may be a totally satisfactory solution. In an installation such as ours where we have nearly 5000 voice stations and more than 450 data lines serving two separately controlled computer centers, this cannot be recommended. Voice demands prevent or inhibit the free use of the switch programming. I am sure, also, that design objectives of the PBX switch and design objectives for a data matrix switch are enough different to cause difficulties.

PERILS AND PITFALLS

The area of data communications is perhaps one of the most dynamic and rapidly changing areas of technology in the information handling world today. Protocols are still under development. The IEEE has been attempting for years to develop IEEE 802 standards. Various schemes have risen and fallen in popularity. Hewlett Packard accepted the standards of IEEE 802.3 in the LAN3000 recently released. General Motors has been spearheading the development of their system of Manufacturing Automation Protocol (MAP) which is identified in the 802 standards as IEEE 802.4. With all of the support that is being generated for this, I would have to place my bets on this system.

Commitments either have not been made by computer vendors and by communications vendors or the commitments are changing. IBM has not yet come out with a LAN interface or with packet transport software. HP committed to support PBX systems and set up a program to certify them. Soon after, they stopped or slowed down their activity in support of system interface with these systems to concentrate their efforts on an interface with AT&T's System 85.

Northern Telecom recently announced their LAN working at 2.56 Mbps on twisted-pair wiring. This announcement, coupled with other rumors that I have heard from other vendors, indicates that some advances in twisted-pair technology may reduce the requirements for coaxial cables.

With such a state of technological flux (another way of saying what I originally said about the lack of knowledge) the advice to

put off a commitment to any networking scheme as long as possible is very good advice. Unfortunately, some of you may have requirements that do not permit such procrastination just as we have had. For those, my hope is that you have a commitment to optimism.

Based upon our experiences, anyone who enters into this area is very much of a pioneer in spite of the fact that data concerning such networking has been in the literature for some time. Your journey may be likened to that of the original pioneers setting out in their covered wagons for the far West. The journey is guaranteed to be a bumpy one, there will be little concrete information on which to base your decisions, and there will be few people with experience with whom to speak.

When we installed the new telecommunications switching equipment we found that there was no way to run IBM terminals through the switch since the NTI coax eliminator system was still under development. (Our first equipment was delivered and installed this past June.) This forced the decision to use ASCII terminals and protocol convertors. None that we could find had yet been thoroughly checked out through the SL-1. We chose to work with ICOT for two reasons: they had terminals operating with RS422 protocol and their convertor was programmable.

We now have a fairly stable system with the ICOT terminals working through the switch. We also had an installation where these terminals were connected to their protocol convertor using the telephone cable plant. The signals were picked off at the switch room before going through the switch. This technique that was used in order to get an installation running while we were having trouble with the convertor/switch interface is a potential way around switch problems.

I mentioned the use of RS422 protocol. This is a fairly new option available for terminals. It can be very important economically when the PBX system is used since the terminal with this protocol connects directly to the phone system. With an RS232 terminal connection, a data module must be used. The data module costs approximately \$350.

In spite of the supposed certification of the SL-1 switch for use with HP equipment, we could not get written information from either vendor about the interfaces of the terminals with the SL-1 devices or the SL-1 output with the CPU's. Our old favorite, Trial And Error, came through again. We did manage to obtain communication through the switch. HP finally produced a wiring diagram for the cabling (marked for hooking up to an Infotron switch).

Some months after we were up and running with the switch, we were finally able to obtain some information from HP about the hookup to an SL-1. It was a little late to tell us that the use of an SL-1 with a Series III is not supported. Fortunately for us,

supported or not, terminals do communicate with a Series III through the switch.

This sort of withholding of technical data has been one of my frustrations with HP. There is a wealth of good technical information available within HP that would be most useful to the beleaguered data processing manager who is trying to build and operate systems that do things a bit differently than the simple installation. The trick that I have not totally mastered is to get one's hands on the publications. When you do manage to find one, the odds are about even that it will be marked company confidential. It makes you feel like some sort of industrial spy to read such material.

One must tune his ears and eyes very carefully in listening to sales presentations and watching the product announcements. The world of advertising has many years of experience in convincing you of something without really saying so. I sat in on a seminar for a new product release as I was preparing this paper. The vendor was touting his new LAN system that was part of his total system product. He showed the connecting device that was required to connect between his terminals and the LAN. On the other part of the LAN he showed CPU's connected directly. To the unsuspecting or to the one that hadn't tried to get such a system to work, the absence of a connecting device between the LAN and the CPU similar to that shown for the terminal might go unnoticed.

The real message that was being presented was that this new system would be ready when the computer manufacturers provided the capability that only they could provide for the LAN connection. By next year it may be fair to ask this vendor and HP about when we might expect the missing pieces to become available. By then there may be a bit of market pressure that will cause the questions to be asked of the technical people in the labs. I don't doubt for a minute that the technical guys will have already looked into it and will be ready to come up with an answer very soon.

That's as good a line as any to lead me to a couple of basic criteria that we use as sort of lifelines when we have to buy equipment even though we know that a new development next year or the year after will make everything different. First, I tend to stay away from any product that can't be used with other products. Second, I always try to choose products that are software controlled and driven.

If a vendor's products are designed to work with other vendor's products there is a high likelihood that rather standard interfacing methods are used. His product will be that much more likely to work with new developments. I simply don't feel smart enough to predict which vendor will come up with the solutions that I need; I prefer to be in the best position possible to work with them all.

My experience tells me that software driven products are more easily updated than products that are controlled by wires and switches. The ICOT protocol convertors are excellent examples. The individual channels can be programmed to operate with different types of terminals. There's a high probability that they will be able to keep up with much of the future developments even if we can't predict what they will be. They will even be able to keep up with IBM's changes, perhaps.

One last constructive criticism for Hewlett Packard. Please do whatever you can to encourage other vendors to interface with your systems. No vendor would attempt to put a product on the market without answers to the question of how it will interface with IBM. A fairly large number of products are listed as VT-100 or VT-200 compatible. I find that I can weed out prospects fairly quickly by asking them how much knowledge they have of their product working with HP. The statement that they can work with any ASCII system simply isn't enough.

When we were looking into LAN's, it seemed that there were any number on the market which would satisfactorily handle IBM and DEC equipment, but none would handle HP until HP put their own system on the market earlier this year. This fact was not always easy to determine. Generally, it was implied that if the HP system had an RS232 connection the hardware could connect. The hardware vendor could not, however, be responsible for the software that would be necessary to handle the packet aspects.

Knowing what I know now, I wonder if those systems really did handle IBM or did they simply permit IBM output to come out of an IBM software system such as MRJE without the packet handling features? One must learn what questions to ask in order to get at the truth. The right questions become a sort of shovel that one can use to dig out the truth.

By golly, hand me that shovel. There has to be a pony here somewhere!

3025. PERFORMANCE SELF-ANALYSIS

Brian Duncombe
CAROLIAN SYSTEMS INC.
Toronto, Ontario, Canada

INTRODUCTION

This paper deals with the subject of system performance - what it is, how we can measure it and what we can do to improve it.

As a consultant specializing in performance of the HP3000 computer system, I have had experience looking at a wide variety of system configurations and application loads. From this perspective, I have come to the conclusion that much of the written material in this area is too analytical in nature and not enough of the focus is on the practical things that anyone can do.

In this paper, I try to provide a technique for analysing any system. The treatment is that of a common sense approach rather than an in-depth technical discussion and as such should be applicable to both the technical as well as the management-oriented user. The material presented requires neither great knowledge of MPE internals nor of the details of how to go about monitoring them. It shows that the subject is a simple example of supply and demand.

WHAT IS GOOD PERFORMANCE ?

In order to discuss a subject, we must first agree on its definition. One definition of performance is "What you want, when you expect it." Most of us could agree with this but we could probably discuss the CAUSE and CURE of the problem forever. In discussing the subject, many approaches and ideas should come to light but the most important thing that should emerge is a firm understanding of the variables of performance and a general idea of how they interact.

WHAT CAUSES DELAY? GETTING THE COMPUTER'S ATTENTION (priority)

With a multi-user system such as the HP3000, many users are often competing for the available processing power. Since there is really only one central processor, each user must compete with every other user for this sometimes scarce resource. In order to provide a reasonably equitable sharing of the CPU resource, MPE uses a system of priorities. These are adjusted according to a set of parameters specified by the system manager with the TUNE command and are dependent upon the scheduling queue that a particular user has been assigned to. In this manner, competing users are favoured or penalized depending on the queue that they are running in (BS, CS, DS or ES) as well as how much CPU

resource they have consumed. By applying these rules, MPE may greatly favour one user over another so that even though one user is getting quite good response times, another user is being almost totally ignored. Of course the total resource available will also delay some or all users if the demand exceeds the supply.

PROCESSING INSTRUCTIONS (central processor speed)

Even when a particular user is able to get the attention of the central processor, the instructions that form the program that is helping him solve his particular task require some finite time to execute. We all hear the claims of computer manufacturers regarding the thousands or millions of instructions that their processor can execute in a second of time (MIPS - millions of instructions per second). A sobering fact is that what appears to be a simple task to a user at a terminal can actually require millions of instructions to be executed. The traditional HP3000 application is usually considered to be low on CPU resource and high on disc I/O requirements but many tasks are processor intensive. In addition, when many users are sharing the system, their relatively small individual requirements can quickly add up.

GETTING MAIN MEMORY SPACE (memory management)

Just as all users must compete for the CPU, all users must also compete for the available main memory resource. Operating systems such as MPE recognize that main memory can be used more efficiently if the main memory size is made to appear very much larger by a system of allocating the less-used memory areas to disc memory (virtual memory management). The theory here is that at any point in time, only a small portion of users are really doing anything and therefore requiring real main memory resource space. This mapping is of course made almost invisible to the various users of the system and it is only when the demands for actively used memory space exceed the available real memory that the scheme breaks down due to the very much slower access time associated with disc storage. When this happens, the delays associated with getting data from and putting data to the virtual memory appear as sluggish performance. At this point, users that make efficient use of main memory and follow some simple guidelines may very well compete unfairly with the other users of the system. They may not suffer noticeable delays even when the majority of the system community does. This should point out the possibility that if more users followed the guidelines, they and possibly the whole community might gain similar improvements.

WAITING FOR I/O TO COMPLETE (disc & terminal speeds)

HP3000 applications are normally characterized as I/O intensive. By this, we mean that the applications access large volumes of data while requiring a relatively low volume of processing of the

data. The most popular devices involved in data transfer are terminals and disc drives.

With terminals, data can be transmitted at rates of up to several thousand characters per second. While this might seem adequate, many applications use block mode and large formatted screens containing several thousands of characters of data to be transmitted. What this means is that a user might very likely grow impatient during the second or so that the data takes to move between the terminal and the computer. Depending on the nature of the user and his expectations, he may see the transmission time as part of the overall response delay or he might consider that once the data begins to move, he is seeing progress and is therefore satisfied. How the user views this transmission time can have a great effect on perceived response. Of course, slowing down the data flow with modems and sharing the transmission capacity by using line-sharing (multiplexing) can worsen this perception.

Disc I/O is often the single largest bottleneck on the HP3000 computer system. The data transmission rate between disc and the computer is in the range of one million characters per second. In addition, some hardware allows simultaneous transmission of data between the computer and two or more disc drives. What slows down this whole process is the mechanics of disc accessing. The physics of the matter is that data is stored on circular tracks on a spinning disc and depending upon where the platter is in its rotation, it can take a significant time to wait for the media to rotate into position for transmission. The construction of most disc drives also provides for moving heads to allow more than one circle of data to be stored on concentric rings of the disc platter. The requirement for head movement also slows the process down. The sad fact is that the average transmission is about 500 bytes of data and the transmission time for this is insignificant when compared to the rotation delay (latency) and head movement time (seek). The net result is that a single disc drive can average only about 25 physical accesses per second.

Viewed in the light of data transfer limitations, the typical HP3000 application can present a real problem to overall system performance.

INDICATORS OF PERFORMANCE

Based on the definition of performance presented earlier, the signs that indicate relative performance to us are elementary. Since it all reduces to users getting what they want when they expect it, the meters of performance are time measurements and expectations.

RESPONSE TIME AT THE TERMINALS

Terminal response time is usually what an end-user is looking at in attempting to measure performance. As discussed previously,

there is some room for interpretation as to what components are actually part of the response delay. Generally response delay is measured from the time the user presses the return/enter key until the first character of data is returned to the terminal. This measure is usually adequate for non-key-entry users of a system but for heads down data entry, the measure is usually considered to be from the time the return/entry key is pressed until all data has been returned to the screen and a new transaction can be begun. On a system such as the HP3000, this terminal response can vary dramatically during the day depending on what else is happening on the system. For this reason, it can sometimes be frustrating to investigate a reported slow response and get there just as the system suddenly picks up speed again.

RUN TIME OF BATCH JOBS

As far as the operations personnel of a computer centre are concerned, the time it takes to run the various batch jobs that must be run is the real measure of performance. It is one thing to have some interactive users whining about response times but when the batch processing components of a system start to run behind, things become very hectic. In the event that nightly batch runs are not able to be completed by the next morning, the system is usually not available to the interactive users until later in the day. The ultimate in poor performance is not being able to access the system at all. Unfortunately, batch processing performance problems seem to creep up on you slowly until all of a sudden you are faced with a severe problem.

FREQUENCY OF PHONE CALLS

An excellent early indicator of deteriorating system performance is an increase in the phone calls that operations receives. These usually start out with the less popular users calling to mention that things are slow and the situation worsens until users start to call to ask if the system has crashed. By developing a habit of "managing by wandering around" the computer room area, you can often tune in on the user community feelings about system performance.

APPROACH TO IMPROVEMENT

THERE WILL ALWAYS BE A BOTTLENECK

No matter what system you look at, there is always a bottleneck that is limiting the performance of the system. No matter how many bottlenecks you relieve, another one will appear to take its place. Realizing this, you can approach the subject as a never-ending one. It is up to you to determine how far you will go to "solve" the performance problems of your system. At some point, the gain will not warrant the effort involved and at this point you should stop for a while. The definition of this point is a subjective one that only you can determine.

APPLY EFFORTS TO AREAS OF MAXIMUM RETURN

Since the resource involved in the current bottleneck is the one that will yield performance improvement when the demand for it is reduced, it makes sense to work on this as the primary target. If performance is currently limited by the processor speed, reducing demand for main memory will have little effect on improving performance. Even within this scope, there will be several areas that you can work on to reduce demand for the critical resource. There will always be areas for which a relatively small effort will yield large results while other areas will exist for which it seems that no amount of work will yield a noticeable result. If you apply some common sense to choosing the area to work on, you should be able to maximize the return on your effort. Clearly, it would be better to tune a program that accounts for 2 percent of all run time on the system than to chose a program that is only run at month end.

KEEP IT SIMPLE

Naturally, you will usually find that there are more areas to improve than there are resources to work on them. In my experience, the techniques that have consistently yielded the best long term benefits have been those that involved a simple change in approach to the data processing problem. As the tuning becomes more and more "elegant", the possibility of introducing a new problem or complicating future maintenance efforts rises dramatically.

ENCOURAGE GOOD HABITS

Most tuning effort is focused on the study of existing systems to see how they can be improved. This is a never-ending task but one that is unavoidable. If in addition to this, you begin a campaign to develop the use of more efficient techniques in the development effort, you will get it right the first time on new system development and reduce the effort required to tune it in the future. By doing this, you not only get helpers working for you in your tuning effort but also you will break the circle of generating new systems to be tuned at a later date. By instilling in everyone the habits of doing it the right way, you have not increased their workload but you have gone a long way towards reducing the number of obvious contributors to poor performance.

YOU ARE NEVER FINISHED

As I mentioned earlier, there will always be a bottleneck to be removed. At some point, you must at least rest from this tuning effort. Only you can define the point at which additional effort is not worthwhile. Even when you feel you have reached this level, you should plan to periodically review the situation to see if you have slipped below this line. The line itself will usually move depending on whether you have more processing

resources than you really need (or more dollars to spend!) or whether additional applications have been pushing your resources to the point where you must do something to carry you over until more processing resources can be acquired. Remember, it is much less painful to spend some time for periodic review than to be suddenly faced with a serious problem that appears from nowhere.

INDICATORS OF A PROCESSOR BOTTLENECK

CURRENT INSTRUCTION REGISTER / ACTIVITY LIGHT

All of the HP3000 central processors provide a visible indication of CPU activity. In the case of the Series II/III/64/68 processors, there is a 16 LED display that normally shows the bit pattern for the machine instruction currently being executed. If this instruction is not changing very rapidly or there is a visible fixed pattern to the display, the system is not busy all of the time. If the pattern is constantly bright and changing, then the processor is probably running at or near capacity. The current instruction register (where available) is the first thing that I look at upon entering a computer room.

With the Series 39/40/42/44/48 processors, there is no current instruction register display. If you open up the front panel of the processor, you will find an activity light which is a single LED that flashes as the processor is executing instructions. While you can't get as good an indication of what is going on, you can easily spot a very busy or very idle system.

BATCH JOBS GETTING NOWHERE

If your installation regularly runs batch jobs during the daytime, the ability of these jobs to make progress is a good gauge of the reserve capacity of your CPU. Since batch jobs will by default reside in the DS subqueue, their priority will be less than that of all interactive processes. If the CPU becomes saturated, the dispatcher function of MPE will totally ignore these less important batch jobs and they will make little or no progress. If you see this happening at particular periods of the day, it is a very good indication that the CPU resource is saturated.

INDICATORS OF A DISC BOTTLENECK

DISC ACCESS LIGHTS

With the exception of the HP7911/12/14 disc drives, all of the disc drives contain an LED that shows when the disc is physically being accessed. By observing this light on each of the drives, you can not only see how busy the various disc drives are but also how evenly the disc accessing activity is spread across the

available disc drives (assuming you have more than one). This is one of the first things a seasoned performance specialist will look at upon entering the computer room.

FALLING PACK COVERS / TAPE SEALS

Many users store the empty pack covers for the various disc packs in use on the top of the disc drive that the pack is currently in. In addition, many people place tape seals on top of disc drives while the tape is mounted on the tape drive. If your operations people are frequently retrieving pack covers and tape seals from the floor then there is a good chance that the disc drive is shaking as it is being accessed (a hand placed on top of the disc drive would give you a clue also!). This shaking is caused by the movement of the arm-access mechanism as the read/write heads are rapidly moved across the surface of the disc. Since this seek movement is a major contributor to slow disc access times, the vibrations might point to a situation where two files on the same disc are being accessed as a set. If one of these files could be moved to a drive with less activity, there is a real potential for disc subsystem performance improvement.

DRIVES THAT MOVE

When the vibration of the arm-access mechanism movement becomes very pronounced, it is common for the disc drive to actually move slightly on the floor. If the feet of the drive are not screwed down to take the weight off the casters, this movement can happen quite easily. If you are experiencing disc drives that "ramble", you should first check to see that the feet are screwed down and if this is the case, you have a very severe disc subsystem access problem. If the seek activity has caused this much vibration, it is undoubtedly also causing slow data access on the drive where the vibration is happening.

TIME TO TEXT IN A GIVEN FILE

A very simple test to measure the capacity of the disc subsystem to handle access requests is the time it takes to TEXT a file into the editor. If you build (or choose) a file containing 50 or 100 records and measure the time it takes from when you hit ENTER for the text command until you get the next editor prompt, you have a simple test. By recording the time when you are the only one on the system, you get the optimum value for your system. When you repeat the test during normal operations, the resulting time when compared to your stand-alone timing gives you an indication of disc subsystem performance. This test can be extended by creating an identical file on each disc drive and repeating the test on each drive.

EXCESS ACTIVITY ON VIRTUAL MEMORY DRIVE(S)

When the system is having trouble keeping the required segments in main memory for all active users of the system, it begins to swap segments for one user out in order to make room for the next user even though the segments will be needed very soon. The state of rapidly swapping segments in this manner is referred to as THRASHING. Once this state occurs, performance of the system deteriorates very rapidly since you are replacing main memory access times with the relatively slow disc access times. If virtual memory is configured on only some of your disc drives, the excess activity on these drives can signal a main memory shortage.

VIOLENT VIBRATION OF VIRTUAL MEMORY DRIVES

Even when the accessing activity is fairly well balanced on the drives, if the drives containing virtual memory are shaking more than the others, you may have main memory shortages. When you start your system and load it initially, disc space is allocated first to system overhead including the directory and then to virtual memory space. After this space is allocated, your own data files are positioned on the disc media. This results in the directory and virtual memory being positioned on the edge of the disc and your data spread across the surface of the platters. When excessive directory or virtual memory accessing begins, the read/write heads must be continually moved from your data files to the directory/virtual memory and back again. This usually results in much longer than average seek distances and more pronounced shaking of the disc devices.

VERY SLOW RESPONSE IN THE EDITOR

A simple test that can highlight main memory shortages involves making simple changes to a few lines in a text file using the editor. To do this, text in a file and then modify several lines at least 50 lines apart. This will require very little CPU resource and only one or two disc accesses. If a noticeable delay occurs between the two changes, the cause is likely to be that between the two activities much of your working set was swapped out to disc and the delay was in making it present again.

THE BIG 10 PERFORMANCE KILLERS

#1 - EXCESSIVE FILE OPENS AND CLOSES

Perhaps the most common cause of poor performance is the situation where programs are opening and closing files frequently during their execution. The overhead of this activity is very heavy and for much of the process it requires exclusive access to system resources such as the directory which locks

other users out. Many programs exist in which file opens and closes are part of a commonly used subroutine or procedure and it is not apparent that this activity is going on.

#2 - INSUFFICIENT IMAGE BUFFERS

IMAGE/3000 by default assigns a number of buffers to be shared by all users of a database. By default this number varies according to the number of users who currently have the database open. In the first place, the defaults are usually not high enough and IMAGE performance suffers because of this. In the second place, as the number of buffers varies, the data segment containing the buffers will require a change in size which often involves swapping it out and back in again. My recommendation is to specify a fixed number of buffers for all numbers of users. This can be done by using DBUTIL and specifying "SET basename BUFFSPECS=32(1/120)". This suggestion should be tempered by the number of databases, the amount of available real main memory and the number of users sharing the database.

#3 - NOT BLOCKING SERIALY ACCESSED FILES

A very common habit on the HP3000 is to allow the file system to assign the default block size to a file. In reality, the file system does not do a particularly good job of this and you should always specify it explicitly. A good rule of thumb is to block all files so that the block size is about 4K bytes. By doing this, you gain a large blocking factor and at the same time do not create buffer segments that are unusually large and therefore difficult for the memory manager to keep in main memory.

#4 - NOT ALLOCATING COMMONLY USED PROGRAMS

One of the most disruptive tasks that the operating system is required to perform is that of making a program file ready to be executed. The many facilities of MPE that make code sharing so convenient also extract a high toll at the time that you RUN the program. The work required to setup the segments and resolve external subprogram linkages is a monumental task and its effect on the system is magnified by the requirement to "own" many of the MPE system resources during the effort. By ALLOCATING the programs that are often run but seldom shared concurrently, you can drastically reduce both the delay to the user trying to invoke the program as well as the other people trying to get work done on the system.

#5 - EXCESSIVE USE OF UDCs

The use of UDCs is very much a mixed blessing. When used in moderation, they provide a great convenience that makes the system much easier to use. Many installations, however, get

carried away with this facility and implement their own operating system interface. In addition, many of the commands defined for the benefit of development staff find their way into the end-user UDCs. Whenever a command is issued, it must first be searched for in all active UDCs for the user. If it is not found there then it is passed to the system command processor. By having a long string of UDCs, you make this scanning more complex than necessary. In addition, many shops develop a standard of a system wide UDC file, an account UDC file and possibly a user UDC file. For each file, an MPE file is kept open for the life of the job/session. This is made worse by naming more than one file at a particular level.

#6 - NOT USING "OUTPUT DEFERRED" WHERE POSSIBLE

This capability of IMAGE/3000 is not very well publicized. What it does is turn off the facility of IMAGE that always keeps disc files current with what is in the IMAGE buffers in main memory. By selecting deferred output, you tell IMAGE that disc writes should only be done when absolutely necessary. By doing this, you can decrease the disc accessing required for a given task by a factor of 2 in many cases. There is a cost for this, though. In the first place, you must be the only accessor of the database (open mode=3). In the second place, you should be prepared to recover and rebuild your database since in the instance of a program or system failure, the data base is guaranteed to be not only logically incorrect but also structurally unsound. For these reasons, the technique is usually reserved for batch processing when you have a good backup and can save all transaction input.

#7 - EXCESSIVE LOGONS

The hardest task that can be requested of MPE is to logon a new user. The process makes exclusive use of many of the system tables and resources thereby preventing others from using them for much of the duration of the logon. While people must log on in order to use the system, it is not at all uncommon to see a user logging on to one group or account and then off and onto another repeatedly. When you investigate the reason for this, you often find that by changing the security on some files or accounts you could eliminate this logon activity or at least reduce it substantially.

#8 - BACKUP

With a few exceptions, system backup is a function for which the system must be used exclusively. The time that it takes is lost time as far as the users of the system are concerned. Even when the system is backed up late at night when no users desire to use it, if batch processing is to be done it will be held up.

Any steps that can narrow the window during which the system is off the air for backup will tend to improve perceived system performance.

#9 - DATABASE DESIGN

The design of the databases that are used by applications can have a serious impact on the overall performance of the system. When chained paths are not provided for commonly utilized searches, the resulting serial scans of the database place an onerous load on the system resources. The pitfalls of database design are a topic unto themselves and you would be well advised to spend time doing a periodic checkup of your data base designs and their usage.

#10 - GETTING ON THE WRONG SIDE OF THE USERS

As our definition of performance points out, the topic is very subjective. If a user is unhappy with the services provided by the computer system, there is probably nothing that can be done to eliminate the comments directed at the system performance. Cases where the user expected a 'keypunch like' response and was delivered ENTRY/3000 to batch files is usually a no-win situation. By setting expectations up front and making a long term commitment to work at user satisfaction, this situation can be largely avoided.

SOFTWARE MONITORS

While there are a great many tools available as part of MPE and many performance measuring techniques that can be used to evaluate your system's performance, there comes a time when you want more. The most obvious tools that come to mind are the so-called software monitors. These tools are programs that have been designed to sample data relevant to some aspect of system performance and report it to you in some fashion. These programs fall into two categories that are differentiated most obviously by their cost. In most instances, increased cost includes support, better documentation and regular enhancements.

CONTRIBUTED

Many programs are available on a contributed basis from either the IUG contributed library tape or the HP TELESUP account. Some of the more popular ones are listed below.

SOO~~~This program exists in many forms (SOO⁴, MOO, GOO, etc.). In general, the program scans the various processes active on the system and reports on CPU usage for each.

TUNER~~~This program reports on the configured size of many of the system tables and in addition shows the current and maximum utilizations.

LOGREPT~~~This program actually consists of several programs that analyse the file close entries in MPE log files and report on the files that have been accessed with some statistics.

SURVEYOR~~~This program monitors the system using the MPE measurement interface and reports various statistics concerned mainly with process level data. In addition, it reports much of the data that TUNER does.

PRODUCTS

OPT/3000 (Hewlett-Packard)~~~This is the tool that all the others are traditionally compared to. It provides a great deal of data about both overall system data as well as process level data. It includes many different display screens designed to highlight specific areas of performance concern.

APS/3000 (Hewlett-Packard)~~~This tool is aimed at analysing use of the CPU from the viewpoint of the code segment containing the instructions. It does this by using a statistical sampling technique to investigate the code in use at any point in time. It is useful for determining what code is the most frequently used and therefore the best candidate for tuning efforts.

TINGLER~~(TEXET)~~~This tool is an excellent tool for analysing where a program is spending its time. On the surface it appears to serve the same function as APS/3000. It uses, however, a technique of modifying a temporary copy of the program to be studied so that it can trap all calls. In practice it is probably the best tool for the serious investigation of code utilization and resegmentation.

LOOK/3000 (Tymlabs)~~~This tool is aimed at the COBOL user. It attempts to provide a tool similar to APS/3000 and TINGLER. It provides a very good feedback facility between the code being executed and the source program and makes it easy to highlight the busy code in a program.

SYSVIEW (Carolian)~~~This program provides facilities very much like OPT/3000. It is oriented towards displays that are aimed at studying various aspects of system performance. It has been developed from the performance specialist viewpoint to make the displays as convenient as possible and to highlight relevant data easily. Its ASYST function allows automatic system monitoring and reports any deviations from user-set permissible ranges.

3026. THE INFORMATION CENTER: IMPLEMENTATION USING HP150 / HP3000

Ellie East
Media General
Richmond, Virginia

Media General is a diversified communications company headquartered in Richmond, Virginia, with primary interests in newspaper publishing, newsprint manufacture, and network and cable broadcasting.

Since December of 1983, Media General's data processing department has established company-wide use of the HP150 computer by approximately one hundred fifty executives, managers, accountants, and secretaries. The ongoing project is the responsibility of an Information Center group, which is part of the data processing department.

CURRENT OPERATING ENVIRONMENT

Media General owns subsidiaries in California, Tennessee, and along the East Coast. All computer system development is done in Richmond by the data processing department, which is a centralized, corporate function. The systems then run on seventeen HP3000s, which are located in corporate offices, and in subsidiaries throughout the country. All seventeen computers are linked together by a network of leased lines.

The HP3000s are used for normal data processing functions such as accounting, payroll, personnel, and inventory. In addition, they run specialized newspaper systems, like classified ad processing and newspaper circulation processing, and some unique applications in the newsprint mills. General purpose packages, such as Listkeeper, HPDraw, DSG/3000, and QUIZ, have been added to the systems as they have become available.

PROBLEM: LACK OF AUTOMATION IN THE OFFICE

The HP3000s carried their intended loads well, but our office workers, especially those at the higher levels, had little, if any, access to computing.

Executive secretaries, in most cases, were using typewriters for their work, which included standard letters, memos, and longer documents. More time-consuming, however, were the involved spreadsheets they were typing! Worse yet, the spreadsheets were originally prepared by executives and managers using pencil, paper, calculator, and eraser. Controllers and general managers had access to the HP3000 accounting systems, but they had no what-if analysis capability. Department heads did their budgets with pencil and paper. Accountants used a mainframe spreadsheet program, but it was slow and cumbersome. The end result, in all these cases, was a waste of expensive white-collar time.

ENTER THE MICROCOMPUTER AND END-USER SOFTWARE

The availability of powerful personal computer software, and HP3000 general-purpose software, both usable by non-technical people, provided an answer to these problems. By taking advantage of the new products, we could give computing power to those people who were doing almost all of their work manually.

BENEFITS DESIRED / CREATION OF INFORMATION CENTER

The goals that we wanted to reach were: time-savings, reduction of errors, fast what-if analysis capability, reduced load on the HP3000s, and reduced load on the data processing department.

In order to achieve the desired benefits, we created an Information Center within the data processing department. We started out cautiously, initially allocating only one person to the Information Center "group". This person had peripheral support from computer operations and technical writers. The group grew to three people, who handle over one hundred people using personal computers as standalone machines, or as terminals attached to HP3000s.

RESPONSIBILITIES OF THE IC GROUP

The Information Center group has eight main responsibilities:

1 - Knowledge.

Because we had to be aware of the various alternatives available to our potential users, self-education was our first and foremost responsibility. We needed knowledge of three major areas.

First, we had to be thoroughly familiar with HP3000 end-user software. Second, in order to help the personal computer user download his mainframe data, we had to know our own in-house systems and databases. And third, we had to learn enough about microcomputer hardware and software to choose the microcomputer most appropriate for our company.

2 - Choice of a microcomputer.

We wanted to avoid the situation which would be created by multi-vendor computers in our offices. We knew that our potential users had very limited technical knowledge, so we wanted a computer with a user-friendly interface. Since we wanted to use HP3000 software when appropriate, we needed a computer which could easily work as a terminal. Last, but certainly not least, we needed sufficient software for word processing, spreadsheets, graphics, and databases.

After looking at several brands of personal computers, we chose the HP150 because it best fit our needs. The table below lists our computer requirements, and the HP150 solutions.

REQUIREMENTS	SOLUTIONS
Easy access to the HP3000.	"Terminal" key
User-friendly operating system	PAM
Sufficient software.	Word processing, database, spreadsheet, graphics
Graphics capability.	Built-in

Extra benefits included a small footprint, standard 256K of memory, standard dual disc drives, and good support in the call-in service of HPCOACH.

3 - Determining specific user needs.

Even though we knew generally what our potential users would need to do, we had to determine specific needs for each individual. This was really a systems analysis function, and usually took the form of initial discussion meetings. We tried to find out what the users were currently doing, what their most time-consuming tasks were, and how they expected the computer (either the HP150 or the HP3000) to help them.

This was a vitally important, but sometimes difficult, stage. Often, because the users had had so little exposure to computing, they were unaware of what a computer COULD do for them, and therefore did not know what to expect or what to ask for. It was up to the Information Center group to get enough information from them to make appropriate software and hardware recommendations.

It was essential that these first contacts be conducted in a professional, respectful, and enthusiastic manner. These meetings could either excite the potential user, or turn him off completely.

4 - Hardware / Software recommendations.

We have recommended the standard HP150 system for all our users on their initial purchase. In some cases, we have subsequently suggested upgraded memory for running large spreadsheets.

Printers have been the most difficult hardware to select. The decision is easy for secretaries needing letter quality output, but other users often cannot predict their output needs. On software purchases, we recommended initially buying only

what was necessary to solve the most pressing problems. We knew that it would probably take the user longer to get comfortable with his computer system than he expected. If he bought more

software than he could learn to use quickly, the least-needed packages would gather dust until he found time for them. In the meantime, because the industry changes constantly, a better software solution may have come along.

5 - Installation, configuration, and initial help.

After the computers were delivered and installed, it was the IC group's job to configure hardware and software, so that system components worked together properly.

Then, the IC group made sure that the user was comfortable enough with his system to get started using it. The users' biggest problem was understanding the data processing operations (formatting, copying files, etc.) associated with running a computer, so we made backup copies of software for them, and supplied formatted work discs. Our goal was to help them avoid these tasks initially, leaving them free to solve problems with their computer, and thereby start to realize the potential benefits the computer offered.

6 - User training.

This is probably our single most important task. We have organized our own training program, which consists of four formal, hands-on classes: HP150 Introduction, Introduction to LOTUS 1-2-3, Advanced LOTUS 1-2-3, and Volkswriter Deluxe/Speller. Training is done in a room equipped with four computers, where we teach eight people (two per computer) in each session.

Response to this program has been overwhelmingly positive. Our users have shown tremendous interest in learning to use their computers well, and have strongly supported our training sessions. This has been true of all levels of personnel throughout all our companies. We have trained about two hundred people, locally and in our subsidiary locations. In addition to non-data processing users, we have trained our own programmers and operations staff.

7 - Post-installation support.

This support falls into four categories. First, we continue to help the users with formatting, copying files, and so forth, for as long as necessary.

Second, we handle miscellaneous problems that arise, such as bugs in software, defective discs sent from the manufacturer, and any hardware problems that occur.

Third, we help with application work when necessary. For instance, we might suggest which software to use for a given problem. We sometimes help create initial databases, and spreadsheet or word processing models. We have had very few

needless questions from our users. As a rule, they have requested help only as long as they really needed it, and have become independent very quickly.

Lastly, as a result of our own continuing self-education, we recommend new hardware and software as it is available. We are often able to suggest new products which enable greater productivity from the computer.

8 - Keeping users informed.

This responsibility is really part of on-going support. Besides normal telephone help and one-to-one help, we have two other means of distributing information.

The first is a newsletter containing general information on the HP150, do's and don't's on computer usage, how to buy software, where to get supplies, and pointers on how to user particular software packages. We periodically send updates as we gather new information.

The second area is that of furnishing step-by-step instructions on formatting discs, copying files, installing software, and other necessary operations. This saves the new computer user from having to dig through manuals. We distribute these instructions in newsletters, class manuals, and on an as-needed basis.

FACTS AND FIGURES

Our first HP150 was delivered in December of 1983. Now, eighteen months later, we have about 150 HP150s company-wide.

The usage of the computers is as follows. Over 90% are connected to an HP3000, so that they can be used as terminals if necessary. Almost 90% use LOTUS, and about 85% use Memomaker, which is a simple word processing package. Volkswriter Deluxe is used by those needing more sophisticated word processing. The percentages drop to 28% for DSN/Link (lately Advance Link), a data communications package, then to 18% for Personal Card File, which is a simple data management software package.

THE BOTTOM LINE

Our original goals were: time-savings, reduction of errors, what-if analysis, reduced load on the HP3000s, and reduced load on the data processing department. We have definitely begun achieving the first four goals. The last - reduced load on the data processing department - is just now beginning to show. We hope to provide more personal computing ability to our end-users, which should reduce the data processing load even more.

3027. Using Intrinsic in COBOL Programs

Brett Clemons
5119 Gateway Dr.
Tampa, Fl 33615

Abstract. MPE intrinsic may be used in COBOL programs to enhance the programmers utilization of system resources. All intrinsic may be called from COBOL programs, with the exception of those using procedures, labels or long parameters. The file system intrinsic are indirectly used by most COBOL programs, yet COBOL verbs do not allow reverse reading of files, or adequate error handling. File system intrinsic allow the COBOL programmer to perform advanced file handling procedures such as timestamping files(by using FREADLABEL, FWRITELABEL and DATELINE), error handling(using FCHECK, FERRMSG, and PRINTFILEINFO), even access to KSAM files chronologically(using FREADC). The COMMAND intrinsic presents the capability to perform many Command Interpreter commands programmatically, allowing access to many useful features of MPE, such as Job Control Words, file commands and job streaming (:STREAM command). Process handling intrinsic can be used to run programs from other programs. Extra data segment intrinsic allow accumulation of large amounts of data that might be too large to access directly in a program, or they may be used for interprogram communication. The sort intrinsic (SORTINIT and SORTEND) and merge intrinsic (MERGEINIT and MERGEEND) may be used for interactive programs when the sort parameters not known at run time, or may change during the execution of the program.

Introduction. MPE intrinsic are the basic parts of all programs, regardless of the programming language. In some, such as SPL, intrinsic calls are explicit, whereas in others such as COBOL, the compiler will generate the appropriate intrinsic calls that may be needed for you to use files or get the date and time, for example. You can write COBOL programs without ever calling intrinsic yourself, but you can also call intrinsic explicitly in your program to enhance the functionality of a COBOL program. Most intrinsic may be called explicitly from COBOL programs. There are a few exceptions, namely those that use long, label or procedure parameters in the call. Other intrinsic, although they accept compatible parameters, cannot be called because their name contains embedded apostrophes, such as PRINT'FILE'INFO. All other intrinsic may be called from a COBOL program. "Native COBOL", for the purposes of this paper, shall be used to refer to COBOL programs that do not call intrinsic explicitly.

Parameters for intrinsic.

Before we can even start using the intrinsics, we must first find out how to call them from our program. Intrinsics need parameters so that can do what we tell them to do. Parameters can take one of two forms: arrays or integers. Arrays are passed to intrinsics in the form of group or elementary data items defined in the DATA DIVISION as USAGE IS DISPLAY, either explicitly or implicitly. For example :

```

01 FILENAME                PIC X(37) VALUE "MYFILE.PUB".
01 FILENUMBER              PIC S9(04) COMP.
.
.
.
      CALL INTRINSIC 'FOPEN'          USING FILENAME;
                                      0;
                                      0;
                                      GIVING FILENUMBER.

```

FIGURE 1
FOPEN intrinsic example

In this example, we see that FILENAME is an elementary display data item; further, FOPEN expects its first parameter to be a byte array if it is specified. FILENUMBER is an integer, which brings us to our second parameter type. When passing an integer parameter to an intrinsic, the data item must be defined as PIC S9(04) COMP (see Figure 1). This data type is a single word integer, a binary number with 16 bits. Double-word integers, or double parameters are defined as PIC S9(09) COMP. COBOL also allows parameters to be passed by value, as the second and third parameters in the example show. The compiler will generate the necessary code for the program to always pass the second and third parameters with a value of zero. Sometimes, though, it is inappropriate to use integers by value. Most file system intrinsics expect the file number of the file to be an integer by reference(i.e., using a data item). Since we can think of these as being true variables, and unknown at run time, we simply must pass some parameters by reference. We can also omit parameters from our call if they are optional. The way to do this is to replace the parameter with "\\" in the parameter list:

```

01 USER-NAME                PIC X(08).
.
.
.
      CALL INTRINSIC 'WHO'          USING \\\;
                                      \\\;
                                      \\\;
                                      USER-NAME.

```

FIGURE 2
Parameter list example

The WHO intrinsic tells us information about the user running

our program. In this example, we need to know the name of the user running our program, but not really anything else. We omit the three parameters before the one we need, and WHO cordially returns the user's name. The WHO intrinsic also has optional parameters after the USER-NAME parameter, but we don't need those either - so we terminate the parameter list with the last parameter that we need. The general format for calling intrinsics from our COBOL program is

```

CALL INTRINSIC 'intrinsicname' [USING { data item } ;..]
                                \\  

                                literal  

[GIVING data item].

```

FIGURE 3
Intrinsic general format

If the intrinsic uses the GIVING phrase in its call, it's referred to as a "typed" procedure. The FOPEN intrinsic is a typed procedure, and it's important to include the GIVING phrase in the call if the intrinsic is expected to do what we want it to do. When we call an intrinsic, and it has done its task, we can test whether it has done everything okay. Intrinsics employ a condition code to relay the status of the call for us to test. The condition code is deeply rooted in machine instruction logic, but for our purposes, all we need to know is that there are three condition codes: CCE, CCG, and CCL. The condition code is compared to zero, the mnemonic representing the condition it tests. Generally, the condition codes represent the success or failure of the intrinsic call, but may relate other information as well:

- CCE - your call has succeeded. Now check to see if it did what you wanted.
- CCG - your call has succeeded, but something wasn't just right. Check everything real carefully.
- CCL - your call failed. Time to do something drastic, like a STOP RUN or the use QUIT intrinsic.

Remember, the specific meanings for the condition code vary from intrinsic to intrinsic, and the explanation above is just a guide. For your COBOL program to check the condition code, you must define what you want to call the condition code in your program. The proper place is in the SPECIAL-NAMES part of the ENVIRONMENT DIVISION. For example :

SPECIAL-NAMES.

CONDITION-CODE IS INTRINSIC-STATUS.

We may call the condition code anything we want, INTRINSIC-STATUS is just an example. Checking the condition code is done immediately after the call to the intrinsic, as in:

```

CALL INTRINSIC 'GETDSEG'          USING EDS-INDEX;
                                   EDS-LENGTH;
                                   EDS-ID.

IF INTRINSIC-STATUS = 0
  NEXT SENTENCE;
ELSE
  IF INTRINSIC-STATUS > 0
    NEXT SENTENCE;
  ELSE
    IF INTRINSIC-STATUS < 0
      DISPLAY 'Could not get an EXTRA DATA SEGMENT';
      DISPLAY 'GETDSEG FAILURE : CCL ';
      CALL INTRINSIC 'QUIT' USING EDS-INDEX.

```

FIGURE 4
Checking the condition code

In Figure 4, we are trying to get an extra data segment to our program. When the intrinsic returns control to our program, we test how successful the call was. Note that in the comparison tests against INTRINSIC-STATUS that the literal constant '0' was used, and not the figurative constant 'ZERO'. Since the condition code changes for each execution of a machine instruction, the following condition code tests would be invalid:

```

IF ( INTRINSIC-STATUS > 0 OR INTRINSIC-STATUS = 0 )
  NEXT SENTENCE;
ELSE
  DISPLAY 'COULD NOT GET THE EXTRA DATA SEGMENT';
  STOP RUN.

```

The conditional test on the first line of the IF statement might cause the DISPLAY statement to execute, even though our call to get an extra data segment was successful. Parameter type conversions in COBOL programs are shown in Figure 5.

If the intrinsic calls for parameters of type -----	then pass a parameter defined as USAGE IS -----
LA,IA,BA,BP,RA or DA	DISPLAY, PIC X(nn)
L or I	COMP, S9(1) - S9(4)
R or D	COMP, S9(5) - S9(9)

FIGURE 5
Parameter conversion for COBOL programs

File System Intrinsic.

When you write a COBOL program and use COBOL verbs to access files, the compiler generates code to invoke the file system intrinsics required for your I/O. These calls are

transparent to you, and you have no control over many attributes that you wish your file to have(if it's a new file) or the way you access it. This is because the COBOL language itself is restrictive, and HP just writes the compiler to perform within a set of specifications, namely, the ANSI standards. Figure 6 shows the coorelation between the COBOL verb and the analogous file system intrinsics that are invoked when a COBOL verb is used to access a file.

COBOL verb and modifier -----	Cooresponding intrinsic -----
OPEN filename	FOPEN
CLOSE filename	FCLOSE
DELETE recordname	FREMOVE *
EXCLUSIVE filename	FLOCK
READ filename	FREAD
READ filename KEY IS dataitem	FREADBYKEY *
READ filename NEXT RECORD	FREAD *
REWRITE recordname	FUPDATE
START filename KEY IS dataitem	FFINDBYKEY *
UNEXCLUSIVE filename	FUNLOCK
WRITE recordname	FWRITE

FIGURE 6

COBOL verbs and their intrinsic counterparts

* denotes these intrinsics are for KSAM files only

In some cases, the file system intrinsics serve our needs better than their COBOL counterparts and, more importantly, they may serve our needs better when we call them from our program and override the compiler-generated calls. File system intrinsics are easy to find in your favorite MPE Intrinsics Reference Manual: they all start with the letter 'F'. To initiate access to a file, we must begin by using the FOPEN intrinsic. For example, to open our \$STDLIST file all we need do is pass FOPEN the necessary parameters:

```
01  STDLIST-FNUM                PIC S9(04) COMP.
.
.
.
CALL INTRINSIC 'FOPEN'          USING \;
                                %614;
                                %1;
                                GIVING STDLIST-FNUM.
```

And presto! Our \$STDLIST is now open and we can now use other file system intrinsics to access it. The first parameter is optional, so we omit it entirely(FOPEN interprets this as meaning that the file is to be a temporary, nameless file). The second parameter is the file options parameter, and this value informs FOPEN to open a new temporary file, named \$STDLIST. The third parameter(the access options parameter) tells FOPEN to allow only write access to the file (you can't

really read from \$STDLIST, that's what \$STDIN is for). FOPEN returns the file number of the new file, and we're ready to go to work. Reading from a file is done with the FREAD intrinsic. This intrinsic requires the file number returned by the FOPEN intrinsic, as well as the buffer area where the information is to go and the number of words or bytes to transfer. FREAD is only one of the intrinsics used to transfer information from an opened file to our program. For example, to read 30 bytes of information from a file, we could use

```

01 FILE-NUMBER          PIC S9(04) COMP.
01 FILE-BUFFER          PIC X(30).

CALL INTRINSIC 'FREAD'  USING FILE-NUMBER;
                           FILE-BUFFER;
                           -30.

```

The third parameter tells FREAD to transfer only 30 bytes to the area in our program called FILE-BUFFER. The negative sign is used to specify bytes; using a positive value would transfer 30 words of information to FILE-BUFFER. To write information to our file, we could use the FWRITE intrinsic. The first three parameters are the same as for the FREAD intrinsic, but it has a fourth parameter, the CONTROL parameter, for use when writing records that you may wish to use carriage control directives. There are a whole lot of values that can be used for this extra parameter, but it is required, and when writing to a disc file without carriage control options, just pass a literal of zero for this parameter. Error handling using native COBOL can get cumbersome when the program just returns a '9' in the first byte of the file status. Luckily, the file system intrinsics know exactly what is wrong with the I/O to that file, and we can use them to divulge that privy information to us. Four intrinsics can be used to provide copious amounts of file information for us: FCHECK (or CKERROR, if native COBOL verbs are used) initiates the sequence by providing the numeric equivalent of the error, which is used by FERRMSG to translate that number into a user readable message; PRINTFILEINFO is then called to provide a descriptive 'tombstone' of all information about the file that is in error. Appendices A and B give examples of error handling in native COBOL mode. Appendix C gives examples of error handling using non-native mode file I/O.

KSAM files.

KSAM files present special problems for the COBOL programmer; not in terms of accessing the files, but in accessing the file in the way that is most efficient and

least logically confusing for us. Some programmers use the COBOL procedures written for KSAM, such as CKOPEN or CKREAD, feeling comfortable that these procedures are in some way superior or more efficient than their equivalent COBOL verbs. Figure 7 shows the correlation between the COBOL verbs and COBOL procedures.

COBOL procedure	Cooresponding COBOL verb
-----	-----
CKOPEN	OPEN filename
CKCLOSE	CLOSE filename
CKDELETE	DELETE recordname
CKERROR	no COBOL equivalent
CKLOCK	EXCLUSIVE filename
CKOPENSHR	OPEN filename
CKREAD	READ filename NEXT RECORD
CKREADBYKEY	READ filename KEY IS
CKREWRITE	REWRITE recordname
CKSTART	START filename
CKUNLOCK	UNEXCLUSIVE filename
CKWRITE	WRITE filename

FIGURE 7
COBOL procedures vs. COBOL verbs

The figure shows that the COBOL procedures function the same as the COBOL verbs. So why use them? The only exception might be CKERROR, which can still be used since its parameters are not dependent on the file accessed, but the status of the return. The COBOL procedures themselves call the file system intrinsics to perform their tasks; but some file system intrinsics that can be used for KSAM files are not even referenced as a COBOL procedure. Namely, four file system intrinsics specifically for KSAM files are listed in Figure 8, along with their functions.

File system intrinsic	Function
-----	-----
FPOINT	Position to relative record number in chronological order
FREADC	Read next record in chronological sequence
FSPACE	Position record pointer forwards or backwards in key sequence
FFINDN	Position record pointer to record number in key sequence

FIGURE 8
File system intrinsics not referenced by
COBOL procedures

These intrinsics allow the programmer to access the KSAM file in ways that indexed sequential files cannot be accessed using native COBOL. For program examples of these useful intrinsics, refer to Appendix A and Appendix B. These COBOL programs show how to read a file in reversed key sequence (which is really decending sequence), and how to read a KSAM file by specifying a relative key sequence number. These methods could be useful in online inquiry programs where users browse a file, forward or backward in keyed sequence. You may also note that any key may be specified for these intrinsics, especially the FSPACE intrinsic, which just spaces forward or backwards a specified number of records. The program example in Appendix A also shows how a COBOL program can be enhanced using just a single intrinsic. The open and the reading of the file is done using the standard COBOL verbs. To tell the FSPACE intrinsic which file to reference, the file NAME as defined in the DATA division is passed in the call, and the compiler generates the necessary code such that the intrinsic references the correct file. This very nice feature allows us to use many file system intrinsics without having to FOPEN the file to get the file number. Native COBOL cannot read KSAM files in chronological sequence, i.e., the sequence in which the records are written to the file. To illustrate, lets assume we have a small KSAM file with three records in it:

	physical record 1	physical record 2	physical record 3
key is	DOG	CAT	ANT
	logical record 3	logical record 2	logical record 1

Using the COBOL verbs while reading the file sequentially would yield three records with keys in ascending sequence: ANT, CAT and then DOG. Using the intrinsics FPOINT and FREADC, or FREADDIR would give records in the sequence they were written to the file, i.e., DOG, CAT and then ANT. In instances where a KSAM file must be read chronologically, these intrinsics are the only method that can be used. KSAM files can also be defined to use special records called user labels. This is easily done with KSAMUTIL, by specifying ";LABELS = n" (where n is an integer between zero and 254) when the file is built. Once this is done, the intrinsics FREADLABEL and FWRITELABEL may be used. To access these user labels, open the KSAM file, and then read or write your user labels using these intrinsics before the first read of a record in the file. User labels exist before any records in the file, and can only be specified at the time the file is built. These user labels can contain any pertinent information that you may deem necessary for processing, such as

date and time of last access, what user last accessed the file, security information, etc. User labels are similar to records in the sense that the first user label is zero, not 1.

Extra Data Segments.

Many COBOL programmers have written very large programs, only to be greeted with a message like

```
ABORT COBOLPRG.PUB.USER;%0.%320  
PROGRAM ERROR #24 :BOUNDS VIOLATION  
PROGRAM TERMINATED IN AN ERROR STATE (CIERR 976)
```

or

```
ABORT COBOLPRG.PUB.USER.%2.%2012  
PROGRAM ERROR #20: STACK OVERFLOW  
PROGRAM TERMINATED IN AN ERROR STATE (CIERR 976)
```

What happened? Chances are, if your program has a very large DATA DIVISION, you have exceeded the physical limits of your stack. You now have two choices - make your program reference less data (by limiting its functionality) or use extra data segments to increase the amount of data your program can reference. Many utilities on the HP use extra data segments(EDS). KSAM uses them to keep track of data buffers and pointers, and IMAGE uses them to keep track of user security information. So what's so special about KSAM and IMAGE? Nothing, really. You can use extra data segments in any COBOL program for any purpose you wish, from storing data to passing information to other programs. To use EDS's a program must be prepared with ";CAP=PH,...". This now brings the program under the classification of using special capabilities, which may scare some people. Rest assured, special capabilities does not mean difficult nor forbidden or even magical. Surely it can be interpreted to mean enhanced! Isn't that what we're looking for in designing optimal programs? On the HP, your program stack is a data segment to MPE, allocated for your program's use until terminated. But your program stack can only have a maximum size of 32,767 words to store data. This is another tradition that HP has that is just a carryover from earlier system architectures. But wait, the maximum size for each EDS is also 32,767 words! If you need to store lots of data in an EDS, you may need multiple EDS's to do the job. Luckily, you can get as many EDS's as your program needs, the only limitation being the system configuration; but don't get carried away or you could bring your system to its knees in a hurry. The intrinsics that are needed for using EDS's are easy to use and not difficult to understand. GETDSEG is the intrinsic used to get an EDS for your program. DMOVIN moves data from the EDS to your stack. DMOVOUT moves data the other way - from your stack to the EDS. FREEDSEG releases the EDS from your program

once you're done using it. Starting with MPEV, calls to the EDS intrinsics must be consistent in the mode in which they are called. If GETDSEG is called in privileged mode, all calls to DMOVIN, DMOVOUT and FREEDSEG must also be in privileged mode; the reverse is also true for user mode. The GETDSEG intrinsic has three parameters and requires all three when you call it:

- *index - initialize this parameter to zero. The number returned in this parameter will be used in subsequent calls referencing the EDS. Since the parameter is an integer, define it as PIC S9(04) COMP.
- *length - here's a tricky one. The value here is represented in words. What if you want an EDS that is 30,000 words long? Since this parameter is an integer, it is also defined as PIC S9(04) COMP. To COBOL, that means its value range is +/- 9,999. Figure 5 shows a program example using an EDS that is 30,000 words in size.
- *ident - this is what we will call our EDS. A value of zero is used as the ident for a private EDS, one that can only be used by the process that created it. Any other value makes the EDS a sharable EDS that can be used by any other process in the job/session.

```

01 EDS-INDEX                PIC S9(04) COMP VALUE 0.
01 EDS-LENGTH-2W           PIC S9(09) COMP VALUE 0.
01 FILLER                   REDEFINES EDS-LENGTH-2W.
   05 FILLER                PIC S9(04) COMP.
   05 EDS-LENGTH-1W         PIC S9(04) COMP.
01 EDS-IDENT-X2            PIC X(02) VALUE IS 'DS'.
01 EDS-IDENT-1W            REDEFINES EDS-IDENT-X2;
                           PIC S9(04) COMP.

```

```

MOVE 30000                  TO EDS-LENGTH-2W.
CALL INTRINSIC 'GETDSEG'    USING EDS-INDEX;
                           EDS-LENGTH-1W;
                           EDS-IDENT-1W.

```

```

IF INTRINSIC-STATUS < 0
  DISPLAY 'Intrinsic GETDSEG failure (CCL)';
  CALL INTRINSIC 'QUIT'    USING EDS-INDEX.

```

FIGURE 9
GETDSEG Intrinsic example

The two intrinsics used for moving data to and from the EDS's are DMOVIN and DMOVOUT. They move chunks of data between the EDS and the stack of your program. As a precaution, they also do some bounds checking for you, just to insure that you aren't trying to move data past the limits of the EDS or your stack. Who want's to mess up someone else's stack or a system table? Both intrinsics use the same parameters in the same order; further, all 3 parameters are required:

- *index - this is the parameter returned by the GETDSEG intrinsic. This is the number that MPE uses to keep track of your EDS.
- *disp - this is the offset into the EDS where you want to start referencing the data. Displacements start in position zero. If you need a displacement of more than 10,000 words, use the trick that was used with the length parameter of the GETDSEG intrinsic, or pass a literal as this parameter.
- *number - the number of words to be transferred to the data area in your program stack. But caution here is advisable. Moving a single word twice will cost as much as moving 1000 words once(1).
- *location - this is where the data is being moved from or to in your program. It must be defined in your program as having the same length as the number parameter above, or a bounds check will fail and and no data will be moved.

```
01 EDS-DATA-AREA
```

```
PIC X(2500).
```

```
CALL INTRINSIC 'DMOVIN'
```

```
USING EDS-INDEX;
```

```
0;
```

```
1250;
```

```
EDS-DATA-AREA.
```

FIGURE 10
DMOVIN program example

The last of the EDS intrinsics we'll look at is FREEDSEG. It releases the EDS from your process and deletes it unless another program in your job/session has referenced the same segment and has not done a FREEDSEG for that segment(in that case the condition code is CCG). The parameters for the call are:

- *index - this is the same index returned by the GETDSEG intrinsic.
- *ident - this is the ident parameter that was initially passed to GETDSEG to get the EDS for your program. FREEDSEG just wants an extra check to make sure that you really want to release this EDS. If this ident doesn't match the one you passed when you called GETDSEG, FREEDSEG rewards your confusion with a return code of CCL, and the EDS is not released.

Issuing MPE commands from a program.

Have you ever wished you could execute an MPE command from your program without having to end the program? Well, there's good news for you; the COMMAND intrinsic is just what you've been looking for. Practically any MPE command can be issued programmatically with the COMMAND intrinsic. The potential is virtually unlimited; you can issue file equations, STREAM jobs, PURGE files, even BUILD files, if you want. A real neat trick is to build a custom help catalog for your users by invoking MAKECAT.PUB.SYS and then issuing two calls to the COMMAND intrinsic as shown in Figure 7. The COMMAND intrinsic has 3 parameters:

- *comimage - this is the MPE command itself. It should be syntactically correct and contain all parameters needed. Figure 7 shows 2 MPE commands; both are group items that are terminated by a numeric value of 8205 (this is the numeric equivalent of "return, space". Don't prefix the command line with the MPE colon prompt.
- *errcode - If your command is formatted correctly, but MPE cannot execute it, this parameter will contain the Command Interpreter error(CIERR).
- *errparm - This parameter return the index of the bad parameter in your comimage if MPE cannot execute your command. If your command won't execute because of a file system error, this parameter will contain the cooresponding FSERR to use if you want to call the intrinsic FCHECK.

01 FILE-COMMAND.

05 FILLER

PIC X(38) VALUE IS

'FILE CICAT.PUB.SYS = HELPCAT.PUB.USER '.

```

05 FILLER PIC S9(04) COMP VALUE
8205.
01 HELP-COMMAND.
05 FILLER PIC X(07) VALUE IS
'HELP'.
05 FILLER PIC S9(04) COMP VALUE
8205.
.
.
.
CALL INTRINSIC 'COMMAND' USING FILE-COMMAND;
COMMAND-ERROR;
COMMAND-PARM.

IF INTRINSIC-STATUS = 0
IF COMMAND-ERROR NOT = ZERO
DISPLAY 'COMMAND intrinsic for FILE failed';
CALL INTRINSIC 'QUIT' USING COMMAND-ERROR;
ELSE
NEXT SENTENCE;
ELSE
DISPLAY 'CCL or CCG for FILE failed';
CALL INTRINSIC 'QUIT' USING COMMAND-ERROR.

DISPLAY '<< Entering HELP mode.. type E to exit >>'.

CALL INTRINSIC 'COMMAND' USING HELP-COMMAND;
COMMAND-ERROR;
COMMAND-PARM.

IF INTRINSIC-STATUS = 0
IF COMMAND-ERROR NOT = ZERO
DISPLAY 'COMMAND intrinsic for HELP failed';
CALL INTRINSIC 'QUIT' USING COMMAND-ERROR;
ELSE
NEXT SENTENCE;
ELSE
DISPLAY 'CCL or CCG for HELP command';
CALL INTRINSIC 'QUIT' USING COMMAND-ERROR.

```



FIGURE 11.
COMMAND intrinsic program example

In figure 11, the error handling at the end of each intrinsic call stops the program if an error is returned, as in

```

COMMAND intrinsic for HELP failed
ABORT HELPPROG.PUB.USER.%0.%552
PROGRAM ERROR#18: PROCESS QUIT PARM=1800

```

This is all nice and fine if we know how the program works (wait, didn't we write it?), but what is CIERR 1800? We can use another useful intrinsic, GENMESSAGE, to interpret our

errors for us, and tell us in plain English what went wrong. GENMESSAGE uses the message catalog, CATALOG.PUB.SYS, to interpret specific errors and return the text associated with that error. MPE uses GENMESSAGE for all textual display. To use GENMESSAGE, FOPEN the file with foptions of %5 and aoptions of %420, or issue a file equation(:FILE CATALOG.PUB.SYS,OLD;NOBUF;MR) and OPEN the file using the COBOL verb. For more specific information about message catalogs, see the System Operation and Resource Management Reference manual(3). If we replace our error handling found after the calls to the COMMAND intrinsic in figure 11 with the code in figure 12, we have a little more user friendly system.

```

01 CATALOG-FNUM                PIC X(16) VALUE
    'CATALOG.PUB.SYS'.
01 CATALOG-FILE-NUMBER        PIC S9(04) COMP.
01 CATALOG-MESSAGE           PIC X(80).
    .
    .
    .
IF INTRINSIC-STATUS = 0
    IF COMMAND-ERROR NOT = ZERO
        PERFORM CALL-GENMESSAGE;
        CALL INTRINSIC 'QUIT' USING COMMAND-ERROR;
    ELSE
        NEXT SENTENCE;
ELSE
    DISPLAY '(CCL) or (CCG) from COMMAND intrinsic';
    CALL INTRINSIC 'QUIT' USING COMMAND-ERROR.

CALL-GENMESSAGE.

    CALL INTRINSIC 'FOPEN'      USING CATALOG-NAME;
                                %5;
                                %420;
                                GIVING CATALOG-FNUM.

    CALL INTRINSIC 'GENMESSAGE' USING CATALOG-FNUM;
                                2;
                                COMMAND-ERROR;
                                COMMAND-MESSAGE;
                                -80.

    DISPLAY COMMAND-MESSAGE.

    CALL INTRINSIC 'FCLOSE'    USING CATALOG-FNUM;
                                0;
                                0.

```

FIGURE 12
Example of using GENMESSAGE intrinsic

Now our error messages are a little more informative:

```
UNABLE TO OPEN 'CICAT.PUB.SYS' (HERR 1800).
ABORT HELPPROG.PUB.USER.%0.%552
PROGRAM ERROR #18: PROCESS QUIT PARM=1800
```

Process Handling.

Process handling is another one of those special capabilities that so many people seldom use because they feel that process handling can cause problems, degrade response time or just generally wreak havoc with their system. Process handling allows us to run programs, which are really processes, from another program. Several intrinsics are available to use this special capability, notably the CREATE and ACTIVATE intrinsics, the CREATEPROCESS intrinsic, and the KILL and SUSPEND intrinsics which allow us to regulate the processes we create. The processes we create are called our 'son' processes, and the program that creates them is referred to as the 'father' process. Other intrinsics allow us to query how our son processes are performing: the GETPROCID and the GETPROCINFO intrinsics return various information about any or all sons processes that we have created. We can create processes that create processes, such that we create a family tree, in a sense. We can create a son process and become the father for that process, which can then create a son process and thus becomes its father. A new intrinsic appearing in MPEV versions is PROCINFO, which can relate information about all processes if you call it in privileged mode. Creating processes is easy to do. It goes without saying that you need write and execute access to the program file that you are going to create. The program that is going to create the process needs to be PREPped with the proper capability list(;CAP=PH,...). However, the program that is to be CREATED need not have PH capability. Figure 13 shows a program example using the CREATE and ACTIVATE intrinsics.

```
01 SON-PROCESS-NAME          PIC X(37) VALUE SPACES.
01 SON-PROCESS-PIN          PIC S9(04) COMP.
.
.
.
```

CREATE-LOOP.

```
DISPLAY 'Enter program name to RUN>'.
ACCEPT SON-PROCESS-NAME.
IF SON-PROCESS-NAME = SPACES
  DISPLAY '<< Normal End of Program >>';
  STOP RUN.

CALL INTRINSIC 'CREATE'      USING SON-PROCESS-NAME;
                             \ \;
                             SON-PROCESS-PIN.
```

```
IF INTRINSIC-STATUS < 0
    DISPLAY 'Program does not exist, try again.';
    GO TO CREATE-LOOP;
ELSE
    IF INTRINSIC-STATUS > 0
        DISPLAY 'CONFIGURATION MAXDATA used'.

DISPLAY 'Process created..now activating..'.

CALL INTRINSIC 'ACTIVATE'      USING SON-PROCESS-PIN;
                                2.

IF INTRINSIC-STATUS < 0
    DISPLAY 'Could not activate process!';
    GO TO CREATE-LOOP;
ELSE
    GO TO CREATE-LOOP.
```

FIGURE 13
CREATE and ACTIVATE intrinsic program examples

The example shown in Figure 13 shows a string being accepted from the user who specifies the program file to be created and then activated. The parameter passed to the CREATE intrinsic is a byte array, and you can optionally specify the group and account in the call to the CREATE intrinsic. The CREATE intrinsic returns the PIN (the Process Identification Number) number for the new process. Other parameters may be specified for the CREATE intrinsic (for specifics on these, consult the MPE Intrinsics Manual). Once the process is created, the necessary code segments are loaded but the program is doing nothing. To get it going, we must use the ACTIVATE intrinsic. The PIN number returned by the CREATE intrinsic is used in the call to the ACTIVATE intrinsic, and the second parameter tells MPE to suspend execution when the son process starts up and wait until it's done to restart the father process. We could also have specified a value of zero, in which case the father process would continue executing while the son process starts executing. Caution should be used to prevent son processes suspending or aborting and causing father processes to be suspended without any way of resuming execution. We can use process handling in conjunction with other special capabilities. We can pass information to son processes by using EDS capabilities (see Appendix C). We can use other process handling capabilities such as the MAIL subsystem that allows processes in the same job or sessions to send messages back and forth between their fathers and sons. Process Handling also allows us to spread very costly functions to other processes. Costly functions might include programmatic sorts or internal table handling, which use large amounts of stack space, and therefore restrict the functions that the main calling program would be able to perform.

Programmatic SORTS in Cobol programs.

Usually, when we write a program to sort data, we know what sequence we are going to sort the records in, be it phone number, address or name sequence. But what if, at run time, we have no idea of how the user wants to see his data? What if our program is an online query program, and the user doesn't know if he needs to look at customers by name of address sequence? Even more challenging, what if there are many files the user could inquire into? There would need to be one SD defined in DATA DIVISION for each file the user would need to sort. The SORT intrinsics are just the thing we need to satisfy our user's needs. The two sort intrinsics, SORTINIT and SORTEND allow us to sort any records in any key sequence we desire, from any file and place them in any file. These intrinsics allow use to perform the SORT/3000 functions from a program, just as if we had run SORT.PUB.SYS online and input the parameters. A program example using the SORT intrinsics can be found in Appendix C. This program is an example of a freestanding program that is activated by a father process, which passes information about the sort files and keys in an EDS. The MERGE intrinsics are similiar to the sort intrinsics, but the parameters are just slightly different. Consult the SORT/3000 reference manual for the exact parameters of the MERGE intrinsics(4).

Intrajob communication.

Quite often, we have jobs that run several programs within them, and we need some way for one program to communicate to the next program in the job stream that everything has run fine or (oh,no!) it did not. We can use two intrinsics, the JCW (for Job Control Word) intrinsics, to help control the flow of programs in a job stream. FINDJCW and PUTJCW are two of the JCW intrinsics that are used to access the JCW table for your job or session. PUTJCW puts a JCW, that is named whatever you want it to be, into the JCW table for your job/session to use. Neither intrinsic changes the condition code, but instead returns a status flag to indicate the return status of the call. Both intrinsics use the same parameters. The JCW name specified in the call must be less than 255 characters long, and must terminate with a space. By using a '@' as the JCW name, you can set all JCW names to the second parameter, the JCW value. FINDJCW is the intrinsic for getting the current value for the JCW from the JCW table. Figure 14 shows an example of how the intrinsic PUTJCW can be used to set a JCW value.

```

01 JCW-STATUS                PIC S9(04) COMP.
01 JCW-NAME                  PIC X(20) VALUE
    'EXTRACTJCW'
01 JCW-VALUE                 PIC S9(04) COMP.

```

```

IF NUMBER-RECORDS NOT > ZERO
  MOVE 1 TO JCW-VALUE;
  CALL INTRINSIC 'PUTJCW' USING JCW-NAME;
                                JCW-VALUE;
                                JCW-STATUS;

IF JCW-STATUS NOT = ZERO
  DISPLAY 'Cannot set JCW value';
  CALL INTRINSIC 'QUIT' USING JCW-STATUS.

```

FIGURE 14

Program example using JCW intrinsic

Figure 14 shows how to set the JCW value to 1 if the NUMBER-RECORDS for an extract program if no records were extracted. The job for using this program might look something like the example in Figure 15.

```

!SETJCW EXTRACTJCW:= 0
!SETJCW CIERROR:= 0
!CONTINUE
!RUN EXTRPROG
!IF EXTRACTJCW <> 0 THEN
!  IF CIERROR = 0 THEN
!    TELLOP No records to extract..going to EOJ
!    EOJ
!  ELSE
!    TELLOP Program terminated abnormally..check JCL
!  ENDIF
!ENDIF

```

FIGURE 15

Job example using JCW's to control job flow

Conclusion.

Many intrinsics were not covered in this paper, mainly because of space limitations. Hopefully, this paper will serve as an introduction to intrinsics that may be used in COBOL programs, providing a basis for additional trial and experimentation to the many and varied uses of intrinsics in novel situations where COBOL may be inadequate to handle for using just native COBOL. In this sense, those who feel that COBOL can sometimes be a language limited in scope and function, can use intrinsic to unshackle the true power of MPE and use it to their advantage and satisfy the sometimes unreasonable burden place on programmers to achieve better and more efficient programs. Using COBOL in conjunction with intrinsics can at last place this language in the class with the powerful languages that exist today, and enhance your ability to solve problems using COBOL with a new perspective.

Good luck and good programming!

Acknowledgements.

Many thanks to the many people who have helped enrich my knowledge and understanding of the HP computer. Special thanks to Dan Parsons, my mentor, who has taught me all he knows about the HP, his patience and excitement an endless source of inspiration for me.

References.

1. Programming for Performance, by Jim Mays. An excellent paper to serve as an introduction for beginning or the most seasoned of HP programmers. Just chock full of good information on how to make your programs run better.
2. The MPE Intrinsic Manual. This should serve as your guide to using intrinsics. Read from front to back.
3. The System Operation and Resource Management Reference Manual. Applicable section to refer to is the MPE message system section.
4. The SORT-MERGE/3000 Reference Manual, by the Hewlett-Packard co. Of particular interest is the sections on calling the intrinsics from FORTRAN or SPL programs. How come there's not a section for calling the intrinsics from COBOL programs?

Appendix A

This program reads a KSAM file sequentially, mixing COBOL verbs and the FREADDIR intrinsic, which reads a record based upon the record number supplied.

```

000100$CONTROL USLINIT,QUOTE='
000200 IDENTIFICATION DIVISION.
000300 PROGRAM-ID.                SEQREAD.
000400 DATE-COMPILED.
000500 REMARKS.
000600 ENVIRONMENT DIVISION.
000700 CONFIGURATION SECTION.
000800 SOURCE-COMPUTER. HP3000.
000900 OBJECT-COMPUTER. HP3000.
001000 SPECIAL-NAMES.
001100*
001200*                DEFINE CONDITION CODE FOR INTRINSIC CALLS
001300*
001400                CONDITION-CODE IS INTRINSIC-STATUS.
001500 INPUT-OUTPUT SECTION.
001600 FILE-CONTROL.
001700
001800        SELECT KSAMFILE                ASSIGN TO 'KSAMFILE';
001900                ORGANIZATION IS INDEXED;
002000                ACCESS MODE IS DYNAMIC;
002100                RECORD KEY IS KSAMKEY;
002200                FILE STATUS IS FILE-STATUS.
002300 DATA DIVISION.
002400 FILE SECTION.
002500
002600 FD        KSAMFILE                LABEL RECORDS ARE STANDARD.

```

```

002700
002800 01 KSAMRECORD.
002900      05 KSAMRECORD          PIC X(72).
003000      05 KSAMKEY           PIC X(14).
003100
003200 WORKING-STORAGE SECTION.
003300
003400 01 FILE-STATUS          PIC X(02) VALUE SPACE.
003410 01 FCHECK-ERROR        PIC S9(04) COMP VALUE 0.
003420 01 FERRMSG-BUFFER     PIC X(80) VALUE SPACES.
003430 01 FERRMSG-BUFLEN     PIC S9(04) COMP VALUE 0.
003500 01 REC-PTR            PIC S9(04) COMP VALUE 0.
003600
003700 PROCEDURE DIVISION.
003800 PARA-BEGIN.
003900
004000      OPEN INPUT KSAMFILE.
004100      IF FILE-STATUS NOT = '00'
004110          CALL INTRINSIC 'FCHECK'      USING KSAMFILE;
004120          FCHECK-ERROR;
004130          CALL INTRINSIC 'FERRMSG'     USING FCHECK-ERROR;
004140          FERRMSG-BUFFER;
004150          FERRMSG-BUFLEN;
004160          DISPLAY FERRMSG-BUFFER;
004170          CALL INTRINSIC 'PRINTFILEINFO' USING KSAMFILE.
004200 LOOP.
004300
004400      CALL INTRINSIC 'FREADDIR'          USING KSAMFILE;
004500          KSAMRECORD;
004600          -84;
004610          REC-PTR.
004700
004800      IF INTRINSIC-STATUS > 0
004900          DISPLAY '<< End of file for KSAMFILE >>';
005000          STOP RUN.
005100
005200      DISPLAY KSAMKEY.
005300      ADD 1 TO REC-PTR.
005400      GO TO LOOP.

```

Appendix B

This program will read a KSAM file in descending sequence, starting with the last record in the file, and stop at the beginning of the file (try that without using intrinsics!)

```

001000$CONTROL USLINIT,QUOTE='
001100 IDENTIFICATION DIVISION.
001200 PROGRAM-ID. REVREAD.
001300 DATE-COMPILED.
001400 REMARKS.
001500 ENVIRONMENT DIVISION.
001600 CONFIGURATION SECTION.

```

```

001700 SOURCE-COMPUTER. HP3000.
001800 OBJECT-COMPUTER. HP3000.
001900 SPECIAL-NAMES.
002000*
002100*           DEFINE CONDITION CODE FOR INTRINSIC CALLS
002200*
002300           CONDITION-CODE IS INTRINSIC-STATUS.
002400 INPUT-OUTPUT SECTION.
002500 FILE-CONTROL.
002600
002700     SELECT KSAMFILE           ASSIGN TO 'KSAMFILE';
002800                                     ORGANIZATION IS INDEXED;
002900                                     ACCESS MODE IS DYNAMIC;
003000                                     RECORD KEY IS KSAMKEY;
003100                                     FILE STATUS IS FILE-STATUS.
003200 DATA DIVISION.
003300 FILE SECTION.
003400
003500 FD     KSAMFILE           LABEL RECORDS ARE STANDARD.
003600
003700 01  KSAMRECORD.
003800     05  KSAMKEY           PIC X(40).
003900     05  FILLER           PIC X(1960).
004000
004100 WORKING-STORAGE SECTION.
004200
004300 01  FILE-STATUS           PIC X(02) VALUE SPACE.
004310 01  FCHECK-ERROR       PIC S9(04) COMP VALUE 0.
004320 01  FERRMSG-BUFFER     PIC X(80) VALUE SPACES.
004330 01  FERRMSG-BUFLLEN  PIC S9(04) COMP VALUE 0.
004400
004500 PROCEDURE DIVISION.
004600 PARA-BEGIN.
004700
004800     OPEN INPUT KSAMFILE.
004800     IF FILE-STATUS NOT = '00'
004810         CALL INTRINSIC 'FCHECK'     USING KSAMFILE;
004820                                     FCHECK-ERROR;
004830         CALL INTRINSIC 'FERRMSG'    USING FCHECK-ERROR;
004840                                     FERRMSG-BUFFER;
004850                                     FERRMSG-BUFLLEN;
004860     DISPLAY FERRMSG-BUFFER;
004870     CALL INTRINSIC 'PRINTFILEINFO' USING KSAMFILE.
004910*
004920*     START KSAMFILE WITH HIGH-VALUES AND GET AN
004930*     INVALID KEY (WE HOPE).. RECORD POINTER NOW POINTS
004940*     PAST THE LAST RECORD IN THE FILE.
004950*
005000     MOVE HIGH-VALUES           TO KSAMKEY.
005100     START KSAMFILE KEY NOT LESS THAN KSAMKEY;
005200     INVALID KEY
005300         MOVE '23'           TO FILE-STATUS.
005500
005510*

```

```

005520*      SINCE THE RECORD POINTER POINTS PAST LAST RECORD
005530*      IN THE FILE, FSPACE -1 NOW POINTS TO THE LAST
005540*      RECORD IN THE FILE UNLESS THE START WAS SUCCESSFUL,
005550*      IN WHICH CASE FSPACE 0 WILL REREAD THE LAST
005555*      RECORD (HIGHVALUES)
005550*
005600      IF FILE-STATUS = '00'
005610          CALL INTRINSIC 'FSPACE'          USING KSAMFILE;
005700                                     -1;
005710      ELSE
005720          CALL INTRINSIC 'FSPACE'          USING KSAMFILE;
005730                                     0.
005800
005900      IF INTRINSIC-STATUS > 0
006000          DISPLAY '<< No records in the file... >>;
006100          STOP RUN.
006200
006300 LOOP.
006400
006410*
006420*      NOW READ THE RECORD THAT IS POINTED TO. THE COBOL
006430*      READ..NEXT RECORD POSITIONS THE RECORD POINTER TO
006440*      THE NEXT RECORD. THIS ALLOWS SEQUENTIAL READS FOR
006450*      COBOL.
006460*
006500      READ KSAMFILE NEXT RECORD AT END
006600          DISPLAY '<< End of file for KSAM file >>;
006700          STOP RUN.
006800
006900      DISPLAY KSAMKEY.
007000*
007010*      RECORD POINTER SHOULD NOW POINT TO THE RECORD
007020*      AFTER THE RECORD JUST READ..
007030*
007100      CALL INTRINSIC 'FSPACE'          USING KSAMFILE;
007200                                     -2.
007300
007400      IF INTRINSIC-STATUS > 0
007500          DISPLAY '<< Beginning of file, no more records >>;
007600          STOP RUN.
007700
007800      GO TO LOOP.

```

Appendix C

This program is created and activated from an online program that asks a user how he wants his records sorted. The creating program passes sort parameters in an EDS and then this program sorts the records, giving the main more stack space, since it doesn't have to sort the records.

```
000100$CONTROL QUOTE=' ,MAP,CROSSREF,USLINIT
```

```
000200 IDENTIFICATION DIVISION.
```

```
000300-----*
```

```

000400*
000500*          I D E N T I F I C A T I O N   D I V I S I O N   *
000600*
000700*-----*
000800
000900 PROGRAM-ID.          SORTPROG.
001000
001100 AUTHOR.             BRETT CLEMONS.
001200
001300 DATE-COMPILED.
005900 REMARKS.
006000
006100 ENVIRONMENT DIVISION.
006200
006300 CONFIGURATION SECTION.
006400 SOURCE-COMPUTER. HP3000.
006500 OBJECT-COMPUTER. HP3000.
006600 SPECIAL-NAMES.
006700*
006800*      DEFINE CONDITION CODE FOR INTRINSIC CALLS
006900*
007000          CONDITION-CODE IS INTRINSIC-STATUS.
007500
007600 INPUT-OUTPUT SECTION.
007700$PAGE
008300 DATA DIVISION.
008500*-----*
008600*
008700*      W O R K I N G - S T O R A G E   S E C T I O N   *
008800*
008900*-----*
009000 WORKING-STORAGE SECTION.
009100*
009200*          DATA AREA FOR 'FOPEN' INTRINSIC
009300*
009400 01 INPUT-FILE-NUMBER          PIC S9(04) COMP VALUE ZERO.
009500 01 OUTPUT-FILE-NUMBER         PIC S9(04) COMP VALUE ZERO.
009600 01 AOPTIONS                     PIC S9(04) COMP VALUE 5.
009700 01 FOPTIONS                   PIC S9(04) COMP VALUE 2.
009800*
009900*          DATA AREA FOR THE 'SORTINIT' INTRINSIC
010000*
010100 01 INPUTFILES.
010200    05 INPUTFILE-1                PIC S9(04) COMP VALUE ZERO.
010300    05 FILLER                     PIC S9(04) COMP VALUE ZERO.
010400
010500 01 OUTPUTFILES.
010600    05 OUTPUTFILE-1               PIC S9(04) COMP VALUE ZERO.
010700    05 FILLER                     PIC S9(04) COMP VALUE ZERO.
010800
010900 01 OUTPUTOPTIONS               PIC S9(04) COMP VALUE ZERO.
011000
011100 01 SORT-NUMBER-KEYS            PIC S9(04) COMP VALUE ZERO.
011200 01 SORT-KEYS.

```

```

011300      05  SORT-KEY-ENTRY          OCCURS 5 TIMES;
011400                                     INDEXED BY SORT-KEY-INDEX.
011500      10  SORT-KEY-OFFSET        PIC S9(04) COMP.
011600      10  SORT-KEY-LEN           PIC S9(04) COMP.
011700      10  SORT-KEY-SEQ-TYPE      PIC S9(04) COMP.
011800      10  SORT-KEY-TYPE          PIC S9(04) COMP.
011810
011900 01  SORT-FAILURE                PIC S9(04) COMP VALUE ZERO.
012000 01  SORT-ERROR-PARM            PIC S9(04) COMP VALUE ZERO.
012100
012200 01  FCHECK-ERRORCODE           PIC S9(04) COMP VALUE ZERO.
012300 01  FERRMSG-MSGBUF             PIC X(80) VALUE SPACES.
012400 01  FERRMSG-MSGLEN            PIC S9(04) COMP VALUE ZERO.
012500 01  SORT-BUF.
012600      05  INPUTFILENAME          PIC X(14).
012700      05  OUTPUTFILENAME         PIC X(14).
012800      05  SORT-NO-KEYS           PIC 9(04).
012900      05  SRT-KEYS               OCCURS 5 TIMES;
013000                                     INDEXED BY SRT-INDX.
013100      10  SRT-KEY-TYPE            PIC 9(04).
013200      10  SRT-KEY-ORDER          PIC 9(04).
013300      10  SRT-KEY-START          PIC 9(04).
013400      10  SRT-KEY-LENGTH         PIC 9(04).
013500      05  ERROR-RETURN           PIC X(01).
013600      05  FILLER                 PIC X(01).
013700*
013800*      EXTRA DATA SEGMENT WORK AREA
013900*
014000 01  INTERPROCESS-COMM-AREA.
014100      05  EDS-INDEX               PIC S9(04) COMP VALUE ZERO.
014200      05  EDS-LENGTH             PIC S9(04) COMP VALUE ZERO.
014300      05  EDS-ID-X               PIC X(02) VALUE 'DS'.
014400      05  EDS-ID                 REDEFINES EDS-ID-X;
014500                                     PIC S9(04) COMP.
014900
015000$PAGE
015100*-----*
015200*
015300*      P R O C E D U R E   D I V I S I O N      *
015400*
015500*-----*
015600 PROCEDURE DIVISION.
015700
015800 S0000-EXECUTIVE                SECTION.
015900
016000*
016100*      GET THE EXTRA DATA SEGMENT FROM FATHER PROCESS
016200*
016300      CALL INTRINSIC 'GETDSEG'      USING EDS-INDEX;
016400                                     EDS-LENGTH;
016500                                     EDS-ID.
016600
016700      IF INTRINSIC-STATUS < 0
016800      DISPLAY 'GETDSEG FAILURE DURING SORT';

```

```

016900      DISPLAY '*** SORTPROG ABNORMAL END ***';
017000      CALL INTRINSIC 'QUITPROG' USING EDS-INDEX.
017100
017200      CALL INTRINSIC 'DMOVIN'          USING EDS-INDEX;
017300                                     0;
017400                                     41;
017500                                     SORT-BUF.
017600
017700      IF INTRINSIC-STATUS NOT = 0
017800          DISPLAY 'DMOVIN INTRINSIC FAILURE DURING SORT';
017900          DISPLAY '*** SORTPROG ABNORMAL END ***';
018000          CALL INTRINSIC 'QUITPROG' USING EDS-INDEX.
018100
018200      CALL INTRINSIC 'FOPEN'            USING INPUTFILENAME;
018300                                     FOPTIONS;
018400                                     AOPTIONS;
018500                                     GIVING INPUT-FILE-NUMBER.
018600
018700      IF INTRINSIC-STATUS NOT = 0
018800          MOVE 1                          TO ERROR-RETURN;
018900          CALL INTRINSIC 'FCHECK'        USING INPUT-FILE-NUMBER;
019000                                     FCHECK-ERRORCODE;
019100          CALL INTRINSIC 'FERRMSG'     USING FCHECK-ERRORCODE;
019200                                     FERRMSG-MSGBUF;
019300                                     FERRMSG-MSGLEN;
019400          DISPLAY FERRMSG-MSGBUF;
019500          GO TO ACTIVATE-FATHER.
019600
019700      CALL INTRINSIC 'FOPEN'            USING OUTPUTFILENAME;
019800                                     FOPTIONS;
019900                                     AOPTIONS;
020000                                     GIVING OUTPUT-FILE-NUMBER.
020100
020200      IF INTRINSIC-STATUS NOT = 0
020300          MOVE 1                          TO ERROR-RETURN;
020400          CALL INTRINSIC 'FCHECK'        USING OUTPUT-FILE-NUMBER;
020500                                     FCHECK-ERRORCODE;
020600          CALL INTRINSIC 'FERRMSG'     USING FCHECK-ERRORCODE;
020700                                     FERRMSG-MSGBUF;
020800                                     FERRMSG-MSGLEN;
020900          DISPLAY FERRMSG-MSGBUF;
021000          GO TO ACTIVATE-FATHER.
021100
021200      SET SRT-INDX SORT-KEY-INDEX      TO 1.
021300*
021400*          SET UP THE SORT BUFFER THAT SPECIFIES KEYS
021410*
021420      PERFORM S2000-MOVE-SORT-KEYS SORT-NO-KEYS TIMES.
021500
021600      MOVE INPUT-FILE-NUMBER            TO INPUTFILE-1.
021700      MOVE OUTPUT-FILE-NUMBER          TO OUTPUTFILE-1.
021800      MOVE SORT-NO-KEYS                TO SORT-NUMBER-KEYS.
021900
022000      CALL INTRINSIC 'SORTINIT'        USING INPUTFILES;

```

```

022100                                OUTPUTFILES;
022200                                OUTPUTOPTIONS;
022300                                \\\;
022400                                \\\;
022500                                SORT-NUMBER-KEYS;
022600                                SORT-KEYS;
022700                                \\\;
022800                                \\\;
022900                                \\\;
023000                                \\\;
023100                                SORT-FAILURE;
023200                                SORT-ERROR-PARM.
023300
023400    IF INTRINSIC-STATUS NOT = 0
023500        MOVE 1                                TO ERROR-RETURN;
023600    CALL INTRINSIC 'SORTERRORMESS' USING SORT-ERROR-PARM;
023700                                                FERRMSG-MSGBUF;
023800                                                FERRMSG-MSGLEN;
023900        DISPLAY FERRMSG-MSGBUF;
024000        GO TO ACTIVATE-FATHER.
024100
024110*    INITIATE THE SORT.
024120
024200    CALL INTRINSIC 'SORTEND'.
024300*
024310*    CLOSE THE SORT FILES
024320*
024400    CALL INTRINSIC 'FCLOSE'    USING INPUT-FILE-NUMBER;
024500                                0;
024600                                0.
024700
024800    CALL INTRINSIC 'FCLOSE'    USING OUTPUT-FILE-NUMBER;
024900                                0;
025000                                0.
025100
025110    ACTIVATE-FATHER.
025120
025200    CALL INTRINSIC 'ACTIVATE'    USING 0.
025300
025400    IF INTRINSIC-STATUS > 0
025500        DISPLAY 'ACTIVATE INTRINSIC FAILURE AFTER SORT';
025600        DISPLAY '*** SORTPROG ABNORMAL END ***';
025700        CALL INTRINSIC 'QUITPROG'    USING 0.
025800
025810    STOP RUN.
025820
025900    P0090-EXIT.
026000    EXIT.
026100$PAGE
026200    S2000-MOVE-SORT-KEYS                SECTION.
026300
026310    MOVE SRT-KEY-TYPE( SRT-INDEX )
026320                                TO SORT-KEY-TYPE( SORT-KEY-INDEX).
026400    MOVE SRT-KEY-START( SRT-INDX )

```


026500 TO SORT-KEY-OFFSET (SORT-KEY-INDEX).
026600 MOVE SRT-KEY-LENGTH (SRT-INDX)
026700 TO SORT-KEY-LEN(SORT-KEY-INDEX).
026701*
026705* ZERO FOR SORT-KEY-SEQ-TYPE MEANS SORT IN ASCENDING
026707* SEQUENCE
026710 MOVE ZERO TO SORT-KEY-SEQ-TYPE(SORT-KEY-INDEX).
026720
027000 SET SRT-INDX SORT-KEY-INDEX UP BY 1.
027100
027200 P2090-EXIT.
027300 EXIT.

3028. Writing Intellegent Software

Otis Whitehurst
Vermont Housing Finance Agency
P.O. Box 408
Burlington, Vermont 05402

What is "Data" ? What is it useful for ? What kind of operations can I perform on it or with it ? These kinds of questions come to mind as we consider some aspects of what we do or would like to do with the information that we deal with on a daily basis. It is easy to get bogged down in the day to day grind and lose sight of the broader view of what "Data Processing" and information management is all about. Why do we use a computer to manage our information ? What kinds of things do computers do well ? What information does the computer have available about our information ? How can we get a computer to accomplish what we want ?

The purpose of these abstract questions is to determine what it is that we want to do. Before we can determine how to undertake a task, we must define the task. Your answers may differ slightly from mine, but in the abstract the answers seem to fall into several categories. The company I work for has an HP3000 series 40. We use it to store, retrieve, display, sort, and analyze the information that we feed into it. If you use a computer, you probably do very similar things, though your emphasis may be in different areas. One item we share in common, no matter what type or size computer we use. That item is software. No chip or black box can accomplish anything without software. You can write it in COBOL, assembly language or burn it in, but you can not use a computer (except maybe as a paper weight or conversation piece) without software.

When we consider what we want to do with an HP3000 and how to do it we can think of a lot of different areas that can enhance and expedite our needs. One area is "Intelligent Software". By "Intelligent Software", I mean software that acts as an extension of what we want to accomplish. Software that can use the information available on the HP3000 in conjunction with a set of rules and with a limited amount of interaction to perform tasks. As I considered how I do my job and what it is that I do, two things cropped up again and again. First, some of my work is very repetitive. I do it the same way with little variation each time. Second, underlying a lot of my work I found certain inherent assumptions. I assumed certain things almost automatically depending on the context of what I was doing.

The work I do involves producing Reports, interactive on-line screens and the creation or transformation of files. We use screens displayed on terminals to interact with Image databases and other files on the HP3000. The user can add, delete, change, or inquire entries using the interactive screens. Databases and files are updated interactively as transactions are entered or changed. Reports are selected by the user and generated as a job created at the time the report is requested. Most of our users do not create their own files or databases but rely on the data processing department. A few users write their own adhoc

reports, but most rely on the data processing department. All production reports are written by data processing.

The mix or emphasis of the three different areas may vary from site to site, but most HP3000 users probably utilize some or all three of the areas mentioned. In each of the three areas mentioned, both repetition and assumption exist. The idea of repetition may be thought of as a pattern or a type. The idea of assumption may be thought of as context.

Reports when examined generally follow a very narrow pattern. In the simplest form report software reads a disc file and prints the contents on a printer. The pattern is as follows:

- A. Select file
- B. Open file
- C. Read a record
- D. Write a line on printer.
- E. Goto "C".

As the complexity of the report grows, the pattern also grows more complex. Still, the basic pattern remains at the core. Add a database as the source of the information and the complexity of opening and reading increases. If you examine reports that use databases you will again find a very similar pattern as follows:

- A. Open Database
- B. Lock sets or paths needed (maybe)
- C. Read selected sets (Chained, Serial, etc.)
- D. Write composite line to printer
- E. Goto "C".

You may be quite aware of the repetition in your work, but if you are not, it may be discovered by taking some aspect of what you do (writing reports for instance) and outlining in general what is done. After several projects are outlined, compare the outlines and note both the similarities and the differences. Is there a pattern to the differences? Could a computer be given the pattern and a list of the "differences" and produce the same result?

A more subtle concept is assumption. By its very nature assumption is generally not thought of in most work that we do unless we assume something that causes problems. Defaults are assumptions used to simplify tasks. If you are using the HP3000 and do not specify a group and account, your logon group and account are generally assumed. Using reports as an example again, several assumptions are common. Files that are used may not be fully qualified (no account may be specified), so the logon account is assumed. The printing device may not be described in detail, so standard 66 lines a page 132 characters wide, 6 lines per inch are assumed. If you create reports for someone else, they generally assume a great deal. They may only specify what is wanted in general (headings, summaries, items to be reported, etc.) and you must assume (or ask for) the other aspects based on the files or databases the user deals with and what reports have been produced for this user in the past.

The most common and useful application of patterns and assumptions is contained in the concept of "likeness". Someone comes to you and asks for a report "like" one you produced last week and then proceeds to

describe how this new report differs from the old. The assumption is that differences not mentioned are to be the same. The pattern of the old report may be refitted (if the differences are not too extensive) to create the new report. Programmers use a copylib or include files to avoid re-coding sections that rarely change. You may have a favorite form with headings or format that you use as a starter or pattern in creating new forms. I have never created a database schema from scratch. Each time a new schema is created, I text in an old schema and re-form it into what I need.

Now that we have examined some "intelligent software" concepts, what is available that satisfies the need? Software as we know it falls into the three broad areas of packaged software, custom programs, and fourth (or subsequent) generation software. Packaged software is any pre-written software that you buy or receive from outside your site that was written to do the types of things you do, but without knowledge of your specific applications in mind. Custom software is any software written to your specifications with your applications in mind (some users may argue this point). Fourth (and subsequent) generation software is defined in different ways depending on who you talk to, but usually includes some freedom from issuing commands in a procedural form (step by step), a data dictionary, semi-natural language commands, and a report writer. There may be some overlap in the three areas, because the packaged or custom software could use a fourth generation language. For the purposes of our discussion we will examine a hypothetical situation.

You are responsible for an HP3000 or for some area that must get the HP3000 to do meaningful and productive work. Someone comes to you and says "I would like to do such and such with the computer." In applying our three possible methods in the order they were presented the answers might be:

- A. We have a program we bought yesterday that will do exactly what you need (packaged approach).
- B. We will program your task, it will be ready in the morning (custom approach).
- C. You can do it yourself easily with this new software we purchased last week (fourth generation approach).

Though all situations may not be as clear cut, you can be sure that if the program does not do what is needed (or promised), if it is not ready in the morning, or if they cannot figure out how to do it themselves, you will hear about it. Two answers users never like to hear are "the computer can't do that", and "I know I said it would be ready but we ran into problems".

You might say "I don't need to produce for other users, I need to do my own tasks". The problem is the same, only you are both the user of the software and the supplier of the solution. We all know the problem. What would be the perfect solution to the problem? If you could flow directly from the definition of what needs to be done (the result) through the process of getting the computer to do the required work, what would it be like? In a more down to earth solution, what can we imagine that would at least give all the advantages of the currently

available solutions, yet none of the disadvantages ? Intelligent software ?

Several aspects are important in the work that I do. You may put more or less emphasis in some areas, but probably could benefit from the same advantages. Looking at the disadvantages first, packaged software assumes you do business the way they think your type of business operates. If this assumption is not true or your business operation is not common for the HP3000 environment, you are either out of luck or must change the way you do business to conform to the package. The up front costs are greater, but the software is available now for those who can use it. Also, with most packaged software, don't expect enhancements to fit your situation, that falls under custom programming. Custom programming in a traditional language needs a lengthy definition process or repeated prototypes to work well in most cases. Once the problem is defined, the software must include each detailed step to tell the computer what to do. Programs must be debugged, maintained and enhanced. A custom job usually takes longer than either of the other two methods. Fourth generation software disadvantages depend on the software selected, but all require learning a new set of commands, rules, and syntax before starting to solve problems. There is usually a big investment not only in time, but money to start up. The problem to be solved must still be defined well, though prototyping may be easier. Most fourth generation languages that do not translate into traditional source code (some would say not a true 4th generation), sacrifice much flexibility. They may also solve problems in a very inefficient manner or make inherent assumptions that are at odds with the way you would rather work (database locking for example).

There are advantages to each method. Packaged software may be available to do the work right now that fits like a glove and is moderately priced. Maintenance and bugs may be low. Documentation may be great. Spreadsheet packages are an example. For the areas that can use a spreadsheet well there are several very good packages available. Word-processing also lends itself to packaged software. It would be prohibitively costly to develop software for your own use for certain applications, and you would not get many of the "goodies" available in a package.

Custom software can often be written to do exactly what you need in a reasonable amount of time. By using "shell" programs that start with the basics needed for a type of problem, much of the ground work and maintenance is reduced. Copy-libs and includes of source files reduce the repetitive types of coding.

Fourth generation software may be easily maintainable and overcome the limitations of some of the proceduralness and rigidity of traditional languages. Applications may be developed quickly. Data dictionaries can improve the integrity and structure of the files used as well as speeding up generation of screens and reports. If you are a supplier rather than a user, fourth generation software may enable you to shift some application development to the users.

We have not covered all of the advantages and disadvantages in detail, but those we have examined should be useful in looking for a solution to our software problem. The list is as follows: ADVANTAGES
DISADVANTAGES

=====	=====
READILY AVAILABLE	MUST CONFORM TO PACKAGE
FLEXIBLE	MUST SPECIFY EACH STEP
POWERFUL	FULL OF BUGS
LOW MAINTENANCE	MANY ENHANCEMENTS NEEDED
DEPENDABLE	ABORTS REGULARLY
EASY TO USE	CRYPTIC AND AMBIGUOUS
REASONABLY PRICED	EXPENSIVE
REDUCED DP LOAD	USERS MUST LEARN SOFTWARE

You may add other items to your list.

Now that we have some idea of what we are looking for, what does the HP3000 offer that might be used in a solution ?

File equations are a nice place to start. The HP3000 offers the capability of using generic files in the software and assigning a specific file to be used at the time the program is run. One program can be written that can access similar files of any name in any group or account that a user has access to without even needing to prompt the user for a file name. Out put can be redirected to printers or files. The characteristics of a file to be created can be specified without rewriting one line of source code. Two things are needed to use file equations. First, you must know the generic name that the software uses. Second, you must issue the file command during the job or session before the program accesses the file in question. An example of a common use of the file equation is the SYS DUMP software. By issuing the command "FILE SYSLIST=yourfile,OLD" you can redirect the listing to yourfile. File equations override the characteristics defined for a file in a program (as long as the program does not disable this option) so that you have the ability to change characteristics each time the program is run. Of course you cannot change the characteristics of existing files. File equations take effect when issued and stay in effect until the end of the job or session or until cancelled by a reset command. Another nice aspect of file equations is that they are even effective when using databases. File equations may be put into UDC's, can be issued from the MPE prompt or may be issued in BREAK from a program.

Database oriented software can gain a lot from the use of the DBINFO intrinsic. As with most intrinsics, DBINFO requires certain parameters and returns an answer through a buffer. You must specify information that tells DBINFO what database you want info on and what type of information you are looking for. There are five types of information available as follows:

1. Data items
2. Data sets
3. Paths (information about search items and their relations)
4. Logging
5. Subsystem access

If you ask for certain information concerning data items, data sets, or

paths, you must also specify what item, set or path you want info about. A status parameter is also included so that you may determine if the call was successful, and why not if it was not successful. The buffer parameter must be long enough to contain all the answer for the call to be completed successfully.

DBINFO gives four categories of information concerning data items. You may check to see if you have access to an item, and if you do whether it is read only or update (and possibly add and delete) access for an entry in a set containing the item. The number of items and item numbers of all accessible items in the database or a specified data set can be obtained. You can determine the type, size and whether the item is simple or compound, if you know the name or number of the data item. You may also obtain corresponding item names and numbers if you know either the name or number of the data item.

DBINFO offers similar information on the data sets. You can identify all data sets in a database or all datasets that contain a certain data item. Your access to a data set can be determined. The type, size and other characteristics including the current number of entries is available. Paths between master and detail data sets can be listed if the name or number of either the master or detail set is known. The same is true of search items. Information about the current status of logging and subsystem access is also useful. Programs tend to fail when another logged logical transaction is attempted while a preceding one is in progress.

The FGETINFO intrinsic is available for non-database files. A similar intrinsic using a different parameter format is FFILEINFO. FGETINFO requires a parameter to identify the file to obtain information on. The information about the file is returned to up to nineteen parameters that are of the same type and size as the parameters needed to specify the file characteristics. For example, the size of the file may be described in a double integer, while the name of the file is described in a string or byte array. You may include all or none of the nineteen parameters that describe the file. Some readily available useful items are current number of records, the size and type of the file (KSAM, MPE, etc.), file code, file options, access options, LDEV number, and others. Software can obtain a lot of information about files without having to assume anything.

In using forms there are three intrinsics that can be used. The VGETFIELDINFO intrinsic is used for fields on a form, VGETFILEINFO is used for the whole forms file, and VGETFORMINFO is used for forms within the formsfile. These intrinsics use three parameters. The first parameter is the common area parameter used by most VPLUS/3000 procedures. The status area and the area that specifies the size of the common area must be set prior to calling this intrinsic. The second parameter is the buffer used to hold the answer and the third parameter is set by the intrinsic to tell the amount of information returned in the buffer. The buffer should be large enough to contain all the information requested if you want your calls to be successful. VGETFIELDINFO is especially useful and offers the following information:

Field Name
Screen order (left to right, top to bottom)
Field number
Length of field (in bytes)
Position of field in buffer
Field enhancement (underline, inverse, blinking, etc.)
Data type (CHAR, MDY, NUM2, etc.)
Type of processing (display only, required, etc.)

The operation of the VGETFIELDINFO is interesting. You may request information about a specific field if you know the field name, screen order number or field number. If you do not know specific information you may request information either field by field in screen order or for a group of fields. You must specify the number of fields you want information about and the amount (size of entry) of information that you want about each field you request. If you request more fields than are contained in the form, or keep requesting the next field after you have reached the last field in the form, the intrinsic wraps around and starts reporting with the first field in the form again. Software can use field information to put data in the proper form and format before sending it to the form or to put data received from the form into the proper type of variables for handling.

In addition to the mentioned intrinsics, there are other intrinsics that can give information on other aspects of the HP3000 such as WHO, CALENDAR, CLOCK, DATELINE, FINDJCW, etc.

Aside from intrinsics, another area that can be productive is the area of SD or self describing files. QUERY and some other programs supplied by Hewlett-Packard can create and use SD files. These files use userlabels to record information about file contents and record layout. These userlabels act as mini data dictionaries of a limited nature and can be used to intelligently read and process SD files when deciphered. A program (or programs) exist in the contributed software library that will read and display the contents of the userlabel description of record layout in an SD file. With a little imagination, the userlabels could be processed by a properly designed program to enable the records to be used intelligently. As far as I know Hewlett-Packard does not document the layout of the userlabels in SD files so the complete information is available only to Hewlett-Packard software.

Userlabels may have up to 128 words of information per label for disc files and 40 words per label for tape files. Self describing files generally use a header label as the last userlabel that contains the following information: Program and version creating the SD file

- Record Length
- Number of Fields in a record
- Number of Userlabels
- Number of Fields described per label
- Entry Length (per field description)

Each userlabel before the header label describes fields contained in the body of the SD file. The descriptions are listed in order of the occurrence of the fields in the record, with the last userlabel before the header containing the first group of fields. By reading the header

label and then reading the description labels in descending order you can obtain a picture of the record layout. Field descriptions contain the following information:

Field Name
 Field Type (Image types are used)
 Offset from beginning of record in bytes
 Field size in bytes

That concludes the main body of things available on the HP3000 that are needed for significantly more intelligent software. There are other advantages such as process handling and extra data segments that can have a great effect, but for now I would like to take the problems, advantages, disadvantages and available helps on the HP3000 and combine them. Intelligent software is available as a solution right now. Expert systems are becoming more common on the HP3000. Recently I saw an article on "Reuseable Code" that contained some good ideas for time saving. Fourth generation languages are improving.

In looking for solutions to the problems that occur day to day in the work that I do, I have written some programs that utilize the concepts of intelligent software. Hopefully, this can serve to spur others on to thinking of even better software and more effective use of the HP3000 to perform the work we need to accomplish. From the SD file I have taken the concepts of using userlabels to describe the contents of a file. It has great advantages in that it does not require a separate file, creates little overhead, takes up little space and allows access to a file not only on a record level but a field level without having to hardcode a record layout into the source program.

These "User Label Intelligent Files" (ULI) contain a header label always in label zero (0) of the user label and field description labels in the subsequent user labels. The header label layout is as follows:

BYTES	WORDS	DESCRIPTION
8	1-4	Name of the program that created the file.
8	5-8	Version of the program in the form A.01.01
4	9-10	Source or origin of the information in the file (FORM,BASE,ORIG," ")
26	11-23	Name of original source: File.Group.Acct
Double Int	24-25	Date of Origination: (yyymmdd. 851231)
Integer	26	Number of Fields in record.
2	27	Storage format (INTERNAL, EXTERNAL).
Integer	28	Record size (Negative indicates bytes).
16	29-36	Form name or Data set if "FORM" or "BASE".
Integer	37	File type (0 Undefined -1 Data Base, Data Set 312 User Label Intelligent 1035 VPLUS Form 1036 VPLUS Fast Form 1084 Self Describing).
Integer	38	Starting Column for printout.
4	39-40	Printing setup string.
28	41-56	Available for expansion.
72	57-128	Comments or remarks

For each field or item description, there are two parts. The first part describes the internal record storage and the second describes how the item should be displayed or printed. Five items can be described by each userlabel so that the number of labels required a given number of items is equal to $(2 + (\text{number of items} - 1) / 5)$. This includes allowance for the header label. Each item description occupies 24 words of the description label. The item description labels contain the following information on each item:

	BYTES	WORDS	DESCRIPTION
	16	8	Itemname.
Integer	1		Internal format type, follows IMAGE types. IMAGE: I J K R U X Z P TYPE: 1 2 3 4 5 6 7 8
Integer	1		Internal format size in bytes.
4	2		External format follows VPLUS types plus extra. (CHAR, DIG, NUMn, MDY, MONn, PERn, etc.)
Integer	1		External format size in bytes.
22	11		Edit Masks for external formatting.

These label layouts are all that are needed to read and display files. Using our knowledge of the SD files, we can either convert SD files to user label intelligent (ULI) files or read their contents as they stand. By storing the name of the origin formsfile and formname, we can access the file, have the formsfile and form opened and displayed without any prompts or action by the user. By using DBINFO we can read and display the contents of any data set. We can use editmasks and external formatting to format the display however we want. We can redirect the output to screens or printers and have the program determine the method of display depending on the characteristics of the output device. Because we can read and work with the data in files our management of data becomes more fluid and available. Instead of having to write a special program for each file application, we gain the advantages of a rudimentary data dictionary and eliminate hard-coded file descriptions. The greatest advantage is that the barrier between types of files is broken down. If we assume that field names on forms, Data Item names in databases, Item names in SD files and Item names in user label intelligent files (ULI) are equivalent we can move information around at will. We can take it from a form and put it into a database. We can extract information from one Data Set and update another Data Set or file. By using ULI descriptions as templates we can superimpose structure on existing files.

Three cases where we have implemented this concept have given satisfying results. Using a database loading and unloading program similar to DBLOAD and DBUNLOAD, only intelligently, has allowed us to restructure and prototype databases very easily. A forms data entry program similar to ENTRY has allowed users to design a form and create a useable full function file linked to a form that can be interactively inquired and updated. If later a new item needs to be added, the user adds it to the form, creates a new file and intelligently moves the information from the old file to the new. Finally a file manager program has allowed files to be VIEWED, MOVED, SORTED, CREATED (like an existing file), REPORTED, and UPDATED all from one intelligent piece of software. Data in one format in a file can be easily moved or translated to another

format in another file by using the same item names. The position or order of an item in a file or on a form is irrelevant as long as you know the item name. You can even convert items to other types and sizes so that a file with columns of numbers typed in ASCII can be translated to integer or real.

The above software was developed as a generic answer to the types of problems I encountered in my work. I am sure that these only scratch the surface of what can be done. These programs were written in FORTRAN for the HP3000, but the concepts are applicable to any language or hardware.

3029. The Anatomy of a
True Distributed Processing Application
Michael A. Casteel
Vice President
Computing Capabilities Corporation
4465-A Fairchild Dr.
Mountain View, Ca 94043

Introduction

In early 1984, we at CCC began developing an application to run on a network of HP3000 systems. This application wasn't just to run separately on each of those systems, but needed to continuously monitor and co-ordinate processing on all the different computers. Logically, it would be considered a single, distributed application.

Hewlett-Packard's DSN/DS facility is the obvious basis for such an application, and we set out to design this system using DS. The first thing we discovered was that the DS manual contained little or no information which would help us design a real distributed application. Its main focus was how to access remote computers while sitting at a terminal. For the sake of other users who make the same discovery, this article summarizes some of our recent experiences in distributed application design.

This article describes the architecture which we developed for our application, and presents some of the reasoning behind it. It is clear that different applications would require changes to some aspects of the design. The governing factors are specific application parameters such as data base inquiry and update characteristics, transaction volume, and response time requirements. The parameters of our application are presented in the next section.

Sadly, the present DS manuals leave the application system programmer as much in the dark as the designer when it comes to actually using DS. A companion article will present some implementation details regarding programmatic use of DS and Remote File / Data Base Access.

The Application

The application described here is a multiple-CPU batch job scheduler and controller, now a CCC product by the name of Maestro. The function of this application is to determine which batch jobs are to be run each day on each CPU in the network, then initiate and manage the processing of those jobs. Since, in general, the initiation of one job may be dependent on the successful completion of another job which runs on another CPU, it is necessary for Maestro to monitor processing on all CPUs in the network together.

Figure 1 summarizes the characteristics of this application. The data base is (conceptually, at least) quite simple: Identify which jobs are running, have been run or need to run. The obvious transactions against this data base, such as "Job Initiated" or "Job Succeeded", are not likely to occur in very high volume, probably no more than a few hundred per hour, even over several CPUs.

Since it is not an on-line application, it doesn't matter very much if the "response time" to these transactions is several seconds. Which is a good thing, since network performance is hard to guarantee.

Some of the principal design considerations with respect to distributed processing were:

1. Where does the application program run?
2. Where does the data base reside?
3. How do the transactions make their way through the network?
4. How does the application view the network environment?
5. What happens when something breaks?

Distributing the Application Program

We quickly determined that the program which governs this application should run on every CPU in the network. If the control program were to run on a central computer, relying on remote access to other CPUs, a shut DS line would stop processing on the remote CPU. This would be especially onerous in a switched network, where several remote CPUs might be connected by dial-up but cannot all be connected at once. In addition, failure of the central CPU would stop processing on every CPU in the network.

By running a copy of the application on each CPU, the programs can continue to function even if communications with the rest of the network break down. This also gives the program local access to MPE on every CPU, which helps it do its job, that is, to track all the batch jobs in execution on that CPU and to initiate (STREAM) new jobs.

Distributing the Data Base

Following similar reasoning, we chose to maintain a local data base on each CPU in the network (see Figure 2). All the jobs to run on a CPU are contained in the data base on that CPU, and can be initiated and tracked by the Maestro program on that same CPU. Again, this means that a CPU which is isolated by some fault in the network can continue to function. But, what happens when jobs on this CPU are dependent on jobs run on another CPU, or vice versa?

Here is the main "Distributed Processing" aspect of this application: The interdependencies between jobs which run on different CPUs. A job which runs on CPU "A", for example, may need to wait until a particular job has completed on CPU "B". This means that the Maestro program on CPU "A" needs to know the fate of a job which ran on CPU "B", in order to decide whether to run one of its own jobs.

According to Figure 2, the program on CPU "A" would have to look at the data base on CPU "B" and check the status of the prerequisite job. If it has completed OK, the job can be initiated on CPU "A". If not, the program will have to check again later, perhaps repeatedly. All in all, the program on CPU "A" may have to look at the data bases on each of the other CPUs again and again, often enough to avoid undue delays in initiating dependent jobs.

We decided instead to copy the entire data base onto each CPU. For example, the data base on CPU "A" will contain a record for each job on every CPU. When Maestro needs to check the status of a particular job, such as the one on CPU "B", it simply looks up that job in its local data base. In this way, any number of inter-CPU dependencies can be quickly resolved.

As a side benefit, the operator can use a terminal on any CPU to display the status of every job on every CPU, simply by looking in the local data base.

This scheme implies that updates to the data base must be posted to every CPU. Figure 3 illustrates this situation, with each CPU looking in its local data base to control processing. When a transaction such as "Job Complete" occurs on a CPU, that transaction must be sent to every CPU to update every data base.

Distributing the Transactions

In order to complete Figure 3, we really need to add six more arrows: CPUs "A" and "C" will be producing transactions as well as CPU "B", and they too must update all three data bases. The complexity of the resulting picture makes it clear that, to avoid hopeless confusion, networking considerations must somehow be separated from the application proper. We therefore designed a separate program, MAILMAN, to be responsible for all inter-CPU communication.

Figure 4 illustrates this scheme: Only MAILMAN communicates with other CPUs, and therefore only MAILMAN is knowledgeable about the network structure, or even the existence of other CPUs. In the spirit of modularity, only the application program (BATCHMAN) is knowledgeable about the data base structure and the transactions which are processed against it. The only complication to the application program is that it must send transactions to MAILMAN,

for posting to other CPUs (if any), and receive transactions from MAILMAN which originate on other CPUs.

The details of this operation are shown in Figure 5. The five elements shown in this diagram are present on every CPU in the network: The MAILMAN program, the application program, the application data base, and two MPE Message Files, MAILBOX and INTERCOM. The MAILMAN is linked to other CPUs in the network using DSN/DS. In our application, BATCHMAN executes as a son process to MAILMAN.

The application program writes every transaction occurring on this CPU to the MAILBOX file. When the local MAILMAN reads these, it uses Remote File Access to forward them to the MAILBOX files on all CPUs to which it is linked, so that their data bases can be updated accordingly.

Similarly, the MAILMAN programs on other CPUs also use Remote File Access to write their transactions to this MAILBOX file. When MAILMAN reads these, it sends them to the local application program via the INTERCOM file. This allows the local data base to be updated in accordance with transactions occurring on other CPUs.

In addition, MAILMAN forwards transactions received from a remote CPU to the MAILBOX files on other remote CPUs. As will be seen below, this inter-CPU message forwarding is useful in reducing the degree of interconnection, together with its associated overhead.

Note that, by having the local BATCHMAN and remote MAILMAN programs write to the same local MAILBOX file, the MAILMAN has only one input file. If MAILMAN had to read its input from multiple input files, it would significantly complicate the situation.

Logical Network Structure

A typical representation of a computer system network is shown in Figure 6. A simplistic application of the scheme developed so far would have the MAILMAN on each CPU link to each of the other CPUs. For example, CPU "A" in Figure 6 would link to CPUs "B", "C", "D" and "E"; CPU "B" would link to "A", "C", "D" and "E"; and so on.

Under Hewlett-Packard's DS, this means that each MAILMAN would create remote sessions on (in this example) 4 CPUs. Conversely, each CPU would host 4 remote sessions from the remote CPUs, just to support this application. In general, if a network contained "n" CPUs then each CPU would host "n-1" remote sessions. It seems likely that a large network of this sort would require an excessive amount of DS resources.

With this in mind, and somewhat distrustful of this high degree of interconnection anyway, we instead have imposed the logical network structure shown in Figure 7. One CPU (CPU "A" in this figure) is designated the "primary" CPU, and the other CPUs are "secondary". The "cloud" network of Figure 6 is replaced with the "star" of Figure 7. The primary CPU links to each of the secondary CPUs, but the secondary CPUs only link to the primary.

Note: This is only the LOGICAL network structure, not the physical. We have found that it is ordinarily possible to realize such a logical network structure under DSN/DS regardless of the physical structure, and quite simple to do so under newer DS products, such as X.25 and LAN.

The MAILMAN on the primary CPU sends transactions originating on it to all the secondary CPUs directly, since it is linked to each one. The MAILMAN on a secondary CPU, say CPU "E" in Figure 7, only sends its transactions to the primary CPU. However, the primary MAILMAN forwards these transactions to each of the other CPUs, so they receive the transactions in any case, even though they are not linked directly to the originating CPU.

Although the primary CPU still hosts "n-1" remote sessions, the secondary CPUs now host only one remote session each for this application.

When Something Breaks

As mentioned earlier, the presence of the Maestro application and data base on every CPU in the network is an advantage in the event of a component failure. Should some CPU in the network fail, the other CPUs can continue processing. Of course, if some jobs are dependent on jobs which were to run on the failed CPU, there could be a problem. In fact, our scheme of posting updates to all CPUs has the added advantage that, if the job already ran on the failed CPU, all the other CPUs already know it, so a subsequent failure of that CPU need not hold up their processing.

The "star" network design poses a problem if the primary CPU fails, since there is then no communication among the remaining CPUs. This problem is not a severe one, however, since the designation of the primary CPU is essentially arbitrary. If communication links permit, it is a simple matter to designate another primary CPU and restore full communications.

Recovery from Failures

The failure of either a CPU or DS link can disrupt communications between MAILMAN and a remote CPU. We did not wish transactions to be lost due to such an interruption, which may only be temporary. In fact, we felt that it was desirable to allow communications to be stopped and restarted at any time. Perhaps

a single dial-up link would be used to support several remote CPUs, or perhaps a CPU must to be shut down temporarily in order to install a new release of MPE. We devised the following mechanism, illustrated in Figure 8, to preserve the integrity of the application in such events.

Referring to Figure 8, the MAILMAN on CPU "A" normally sends transactions to the MAILBOX file on CPU "B" via Remote File Access. If either the DS link or CPU "B" fails, or the link is shut intentionally, MAILMAN "A" will be unable to write to the MAILBOX on CPU "B". It will then begin writing the transactions to a local file, which we have called a "POBOX". Every secondary CPU contains a POBOX for the primary CPU. On the primary CPU, where MAILMAN may be linked to several remote CPUs, there will be a POBOX file for each each of the secondary CPUs.

When communication is re-established with the remote CPU, MAILMAN first copies the contents of the POBOX to the remote MAILBOX, then resumes writing transactions directly to the remote MAILBOX. Note that this scheme preserves the chronology of the transactions as well as the transactions themselves.

Conclusion

The distributed processing scheme described in this article has been in operation since December, 1984. A number of implementation details have changed since then, as we learn from experience about operating in the DS environment, but this architecture has proven to be adequate for our application.

It has DEFINITELY been proven that the possibility of component failure must be taken into account in the system design. Although the HP3000, MPE and DS are generally reliable systems, Murphy's law holds with a vengeance in a multiple-CPU network. To date, Maestro has proven reasonably robust in the face of MPE System Failures and DS hang-ups. Damage is ordinarily restricted to the CPU or link which failed, while other CPUs continue to process normally. Full recovery is usually accomplished simply by restarting the failed CPU or DS link.

MAESTRO

Data Base:	Jobs running Jobs to be run
Transactions:	Job initiated New job Job complete State change - INTRO, WAIT, etc.
Volume:	Moderate (100 / hour ?)
Response:	Flexible (10 sec ?)

Figure 1.
Application Characteristics

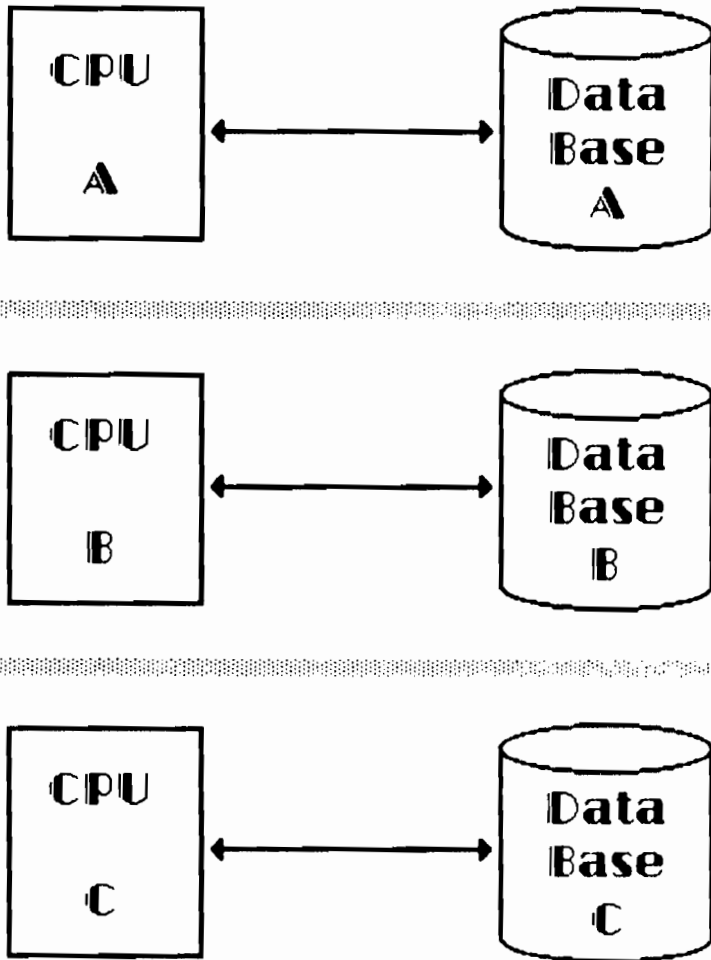


Figure 2.
Distributed Data Base

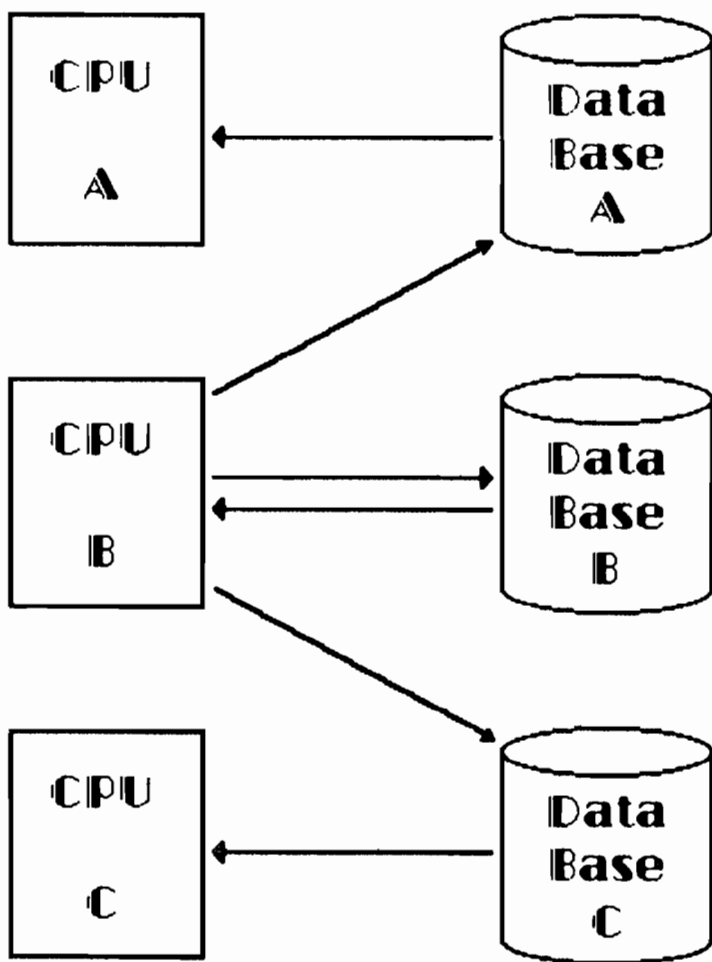


Figure 3.
Distributed Transactions

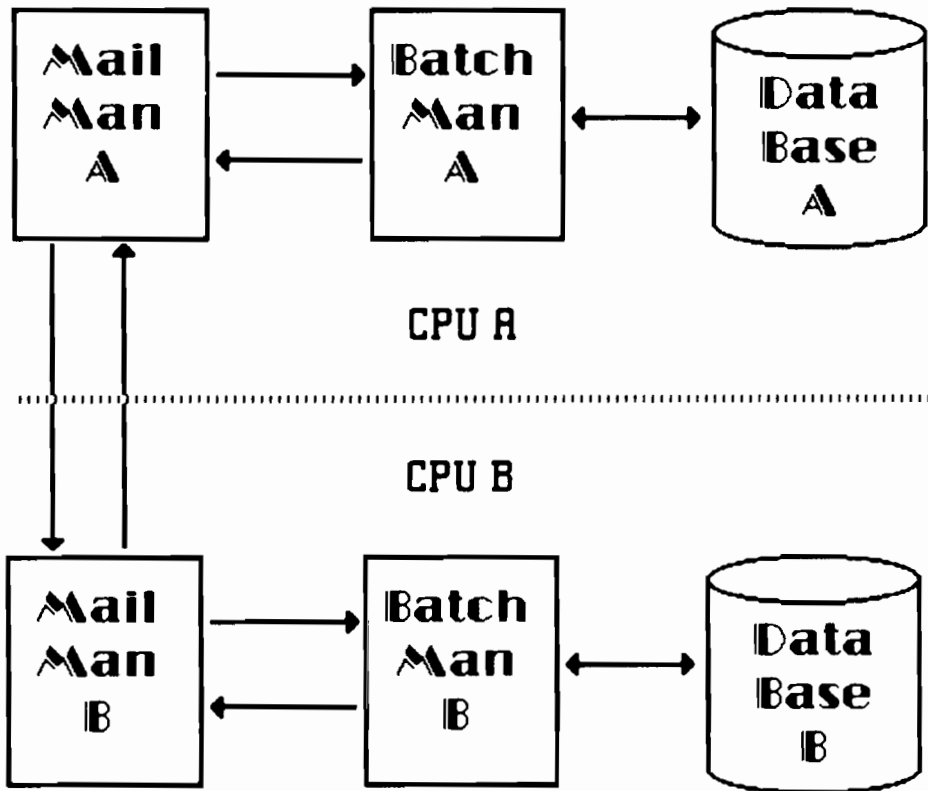


Figure 4.
Transaction Distribution

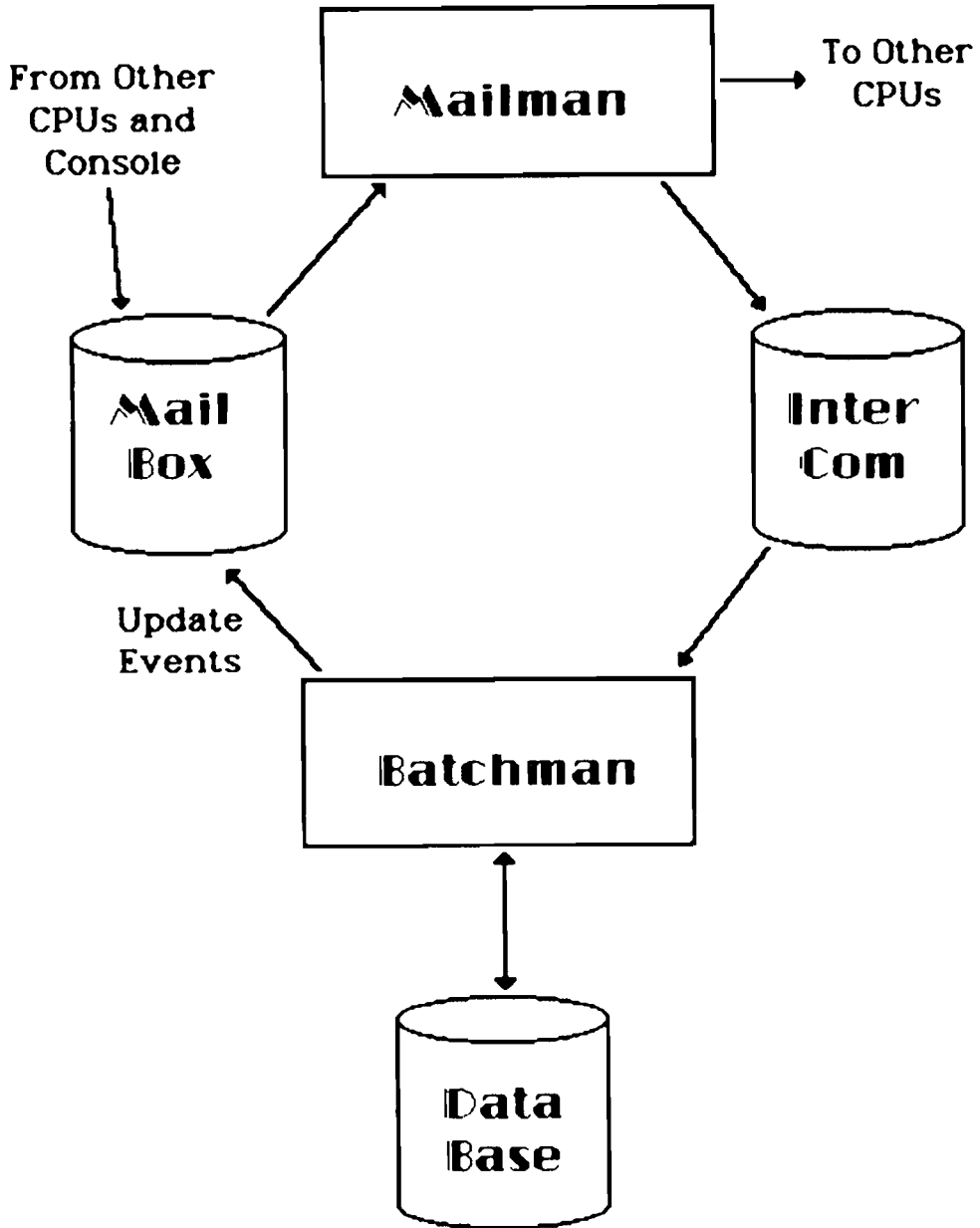


Figure 5.
MAILMAN Operation

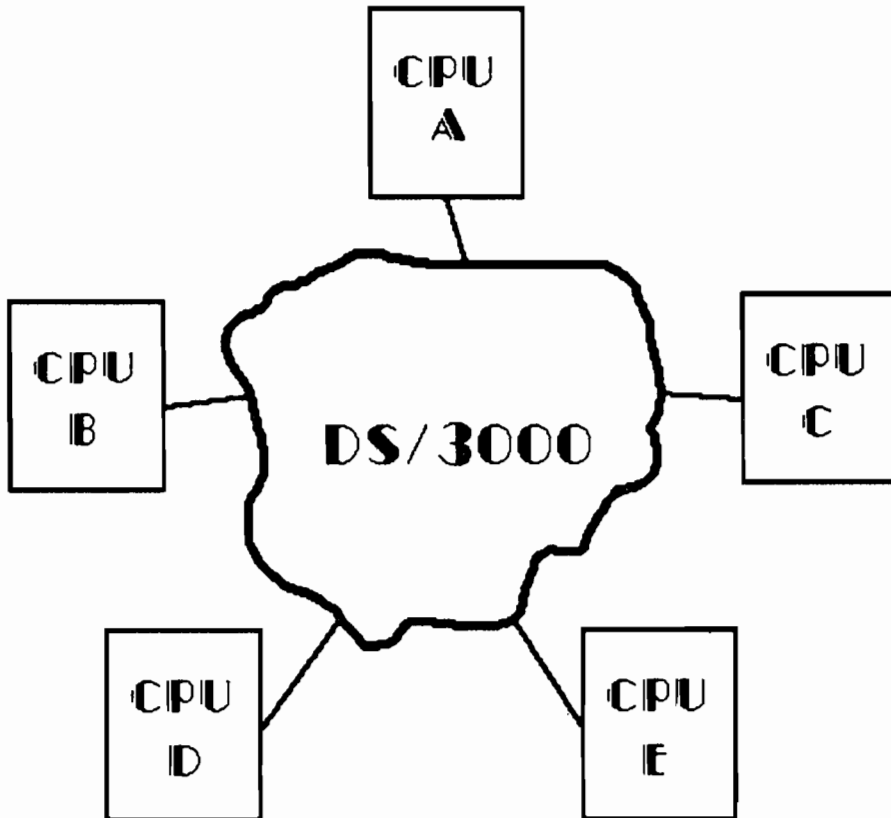


Figure 6.
"Cloud" Network

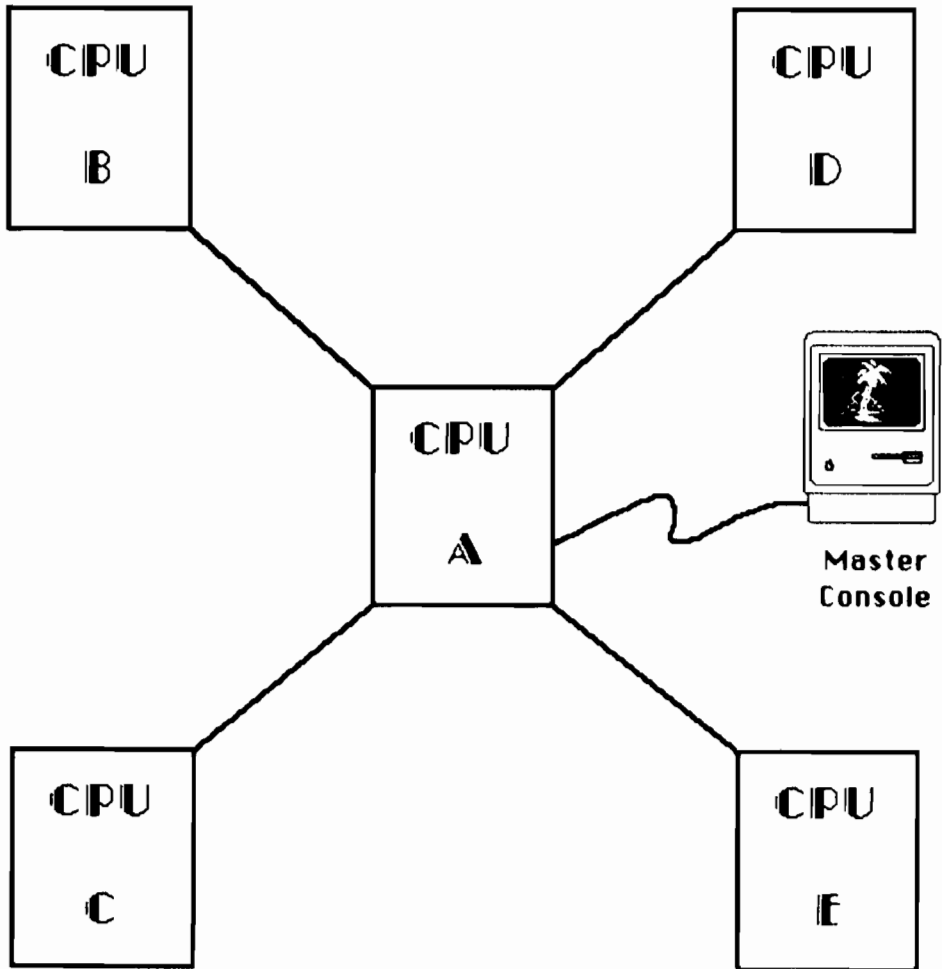
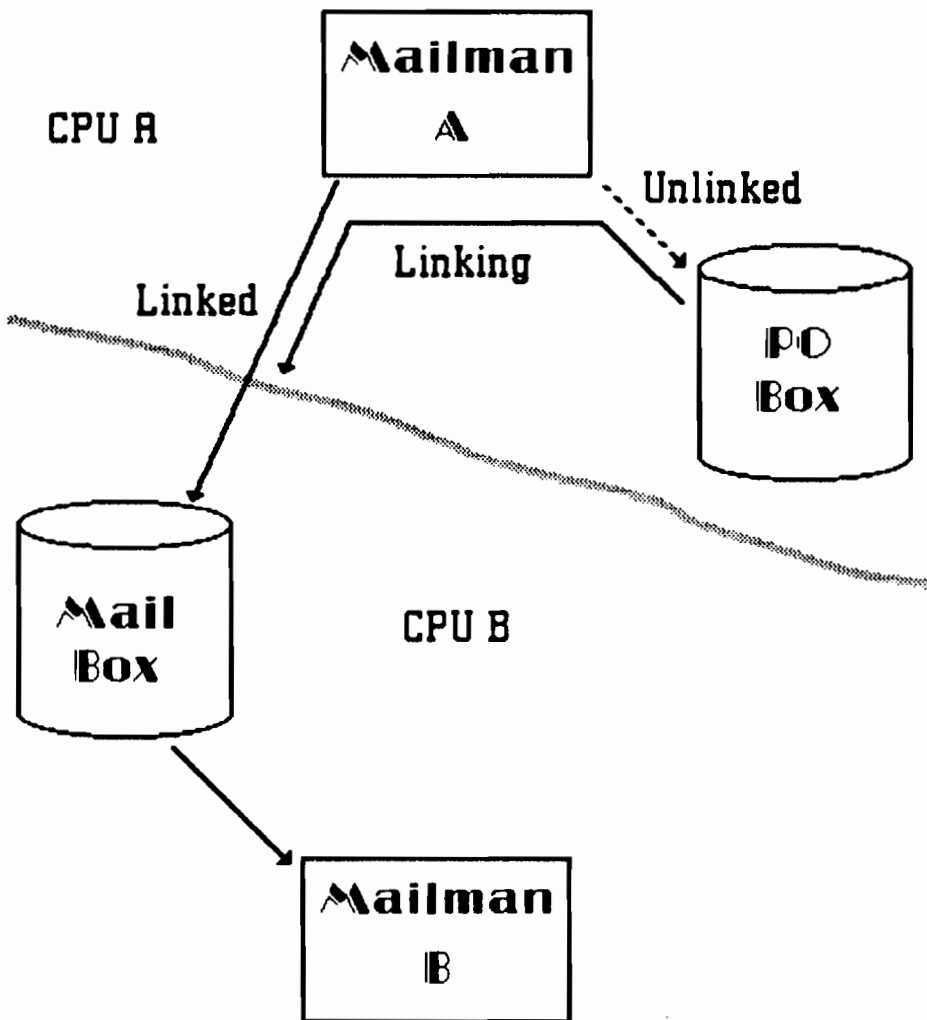


Figure 7.
"Star" Network

Figure 8



**Figure 8.
Communications Error
Recovery**

3030. THE HP3000: A DATA BASE ENGINE?

Charles Sullivan, Jr.
Pacific Coast Building Products, Inc.
3001 I Street
Sacramento, California 95816
(916) 444-9304 x276
Calif: (800) 628-8030
Non-Calif: (800) 824-9046

ABSTRACT OF PAPER

Conventional wisdom states that the HP3000 computer is a great on-line, interactive computer but has grave failings as a batch job machine. This paper will discuss little-used ways to dramatically increase system throughput. Illustrative benchmarks will be used.

For KSAM files, the following techniques will be discussed: reading the data in keyed, sequential order using FREAD; reading the data in chronological order using FREADC; reading the data using no-buff, multi-record I/O; reading just the key file.

For flat MPE files, the effect on performance of disc head movement and blocking factors will be discussed. The following techniques will be covered: reading the file using FREAD; reading the file using no-buff, multi-record I/O.

For IMAGE, benchmarks will be discussed showing that no-buff, multi-record I/O can increase throughput by as much as ten times in applications that serially read a data base.

Sorting performance considerations will be explored. An alternative to HP's sort subsystem will be discussed which uses extra data segments to improve sort performance by 300%.

THE HP3000: A DATA BASE ENGINE?

So-called "back-end" processors have become quite popular in recent years. There are specialized floating-point processors that can make any computer rival the arithmetic powers of a supercomputer. And there are "data base engines" that can assist computers with the task of data base management. The HP3000 computer has an excellent data base system called Image. If the amount of data in the data base is relatively small (under 50,000 records per dataset), reports can usually be generated in a few minutes. But small data bases have a way of becoming big data bases, and then waiting over an hour for a report becomes quite common. Wouldn't it be nice if you could attach a "black box" to the HP3000 and make it fly through even the largest files? Wouldn't it be even nicer if the HP3000 itself could become that "data base engine?"

This paper will explore ways to substantially increase the throughput of the HP3000. It has been my experience that by making a concerted, long-term effort, an improvement in system throughput of 300% is certainly possible. Some applications can be made to run 10 times

faster. The performance of the HP3000 is more than adequate in most cases without adding "back-end" processors.

THE LIMITS OF SYSTEM PERFORMANCE: HARDWARE

In order to get a good idea of the throughput possible on an HP3000, testing was done to determine how much data can be moved between main memory and disc drives.

STAND-ALONE DISC I/O

Processes running on the HP3000 rarely request more than a few sectors of data at a time from the disc. For example, the default blocking factor used by Image is 512 words (4 sectors). [SEE FIGURE 2]

Tests run on a Series 40 with 7933H disc drive

number of sectors transferred	maximum transfers per second	maximum kilowords per second
1	44	5.6
.	44	.
.	44	.
4	44	22.5
.	44	.
.	44	.
42	44	236.5
43	41	225.7
44	22	123.9
.	22	.
.	22	.
96	22	270.3
.	22	.
.	22	.
109	22	306.9
110	20	281.6
111	15	213.1
112	14	200.7
.	14	.
.	14	.
185	14	331.5
186	11	261.9
.	11	.
.	11	.
225	11	316.8

As you can see, transferring just 4 sectors at a time does not make very good use of the I/O bandwidth available on the HP3000. (However, I am NOT advocating that you drastically increase the blocking factors of your Image data bases! There are good reasons to keep Image blocks small.)

On the other hand, transferring 96 sectors at a time (which a system with disc caching enabled can do) is not much more efficient than

transferring 42 sectors. Perhaps setting the maximum cache domain at about 36 sectors would reduce any memory pressure problems that disc caching can introduce.

MULTIPLE-PROCESS DISC I/O

Of course, higher throughput rates can be achieved when there are multiple disc drives available and multiple processes running. On a Series 44 with four drives and two channels, a little more than a doubling of performance is achieved. [SEE FIGURE 4]

Tests run on a Series 44 with 2 GICs and 4 7933H disc drives with 4 processes running.

sectors transfered per I/O per process	total I/O per second	total kilowords transfered per second
1	106	13.6
8	106	108.5
16	92	188.4
24	82	251.9
32	69	282.6
40	61	312.3
48	58	356.3
56	52	372.7
64	51	417.8
72	55	506.9
96	40	491.5
120	34	522.2
144	28	516.1
168	28	602.1
192	28	688.1
216	24	663.6

FILE LOCALITY AND DISC DRIVE PERFORMANCE

You can try to put files which are concurrently accessed on different disc drives so that head movement is minimized. However, since several processes are usually accessing several files on several drives, it is almost impossible to determine where the disc head is. All you can do is experiment; don't be surprised if all your efforts fail to produce a notable difference in performance. It is worth knowing, however, that truly bad head contention can degrade performance by a factor of 2. [SEE FIGURE 5]

Tests done on a Series 40 and 7933H disc:

amount of head movement	physical disc I/O per second
none	44
1 MB	44
2 MB	44
4 MB	40
8 MB	29
16 MB	29
32 MB	29
64 MB	22
128 MB	22

MAIN MEMORY

Because main memory prices have fallen so much in recent years, there are few good reasons for having an HP3000 whose processor spends too much of its time swapping data between main memory and virtual memory. (Even a hundred 30K stacks will fit in 8 megabytes of memory.)

Here are just two common sense tips to help you write programs which make efficient use of main memory:

1. If a program needs a large data stack for only a small length of time, learn to use the ZSIZE intrinsic for shrinking the stack.
2. Don't use the STACK= parameter during program preparation or execution. If your program runs with STACK=20000, then the stack can never be less than 20,000 words. Use the MAXDATA parameter instead and let the operating system dynamically expand your stack as needed.

HARDWARE SUMMARY

During each disc access, transfer as much data as possible. This concept has led to disc caching at the operating system level; much of this paper will be devoted to extending this principle to the applications level where even greater improvement can be realized.

THE LIMITS OF SYSTEM PERFORMANCE: DESIGN

Certainly the HP3000 hardware presents certain limits beyond which performance cannot be improved. But software design has limited performance even further. For instance, the data base and sort subsystems have both been designed to minimize use of main memory--a good idea when 64K of memory was a lot, but not quite so good when a Series 64 can have 8 megabytes. In general, the default way of doing things will save on main memory and disc storage but does not provide the best throughput.

The following discussion will focus on some important things to consider when designing systems and writing programs. This is by no means a comprehensive checklist for designing applications.

MULTIPLE DATA BASES

Image was designed with a bias toward maintaining data integrity at the expense of data base performance (this was a correct choice!). Putting all your data in one data base will probably lead to performance which is inferior to splitting that same data into two or more data bases. (I still prefer putting most of my eggs in one basket. It seems more in the data base spirit to keep all related data in one place.)

TurboImage, which has recently (April, 1985) been announced, should improve the throughput of a single data base by allowing some multi-threading to occur.

FILE SIZE

Always remember that the HP3000 is a minicomputer. Keep your files as small as possible. Try to move superfluous data into history files during off hours so that your daytime processing will go faster.

BLOCKING FACTORS

The easiest course when creating a file is to let the operating system decide how many records should be in each physical block. Rarely will this be even close to the optimum. For example, if you build a file with 128-word records, the file system will create a file with only 1 record per block. Usually you should choose a blocking factor which is much larger. [SEE FIGURE 6]

Test done on a Series 40 with 7933H disc drive.
 Records were 128 words each (1 sector = 128 words).
 Records were serially retrieved via the buffered file system
 (which is the default file access method).

Records per block	Records per second
1	44
2	87
4	168
8	318
16	380
24	394
32	424
40	424
48	423
56	368
64	408

Notice that 32 sectors per block gives the best performance, in this case.

THE DATA BASE ENGINE

Computers impress when they run fast and annoy when they run slow. Although every fast computer eventually becomes a slower one requiring

revitalizing doses of main memory and cpu upgrades, you can certainly delay the aging process by using some of the special features of HP software such as extra data segments, process handling, and no-buff, multi-record I/O.

ADVANCED MPE FILE TECHNIQUES

If you use the normal file access methods, the highest throughput can be achieved by creating files with large blocking factors. But higher throughput is possible using no-buff, multi-record I/O. You must block and unblock your own records from the physical blocks of data that the file system deposits in your stack, but the performance benefit is worth a little extra trouble. [SEE FIGURE 7]

Important note: To achieve the maximum possible speed using no-buff, multi-record I/O, files must have block sizes which are an exact multiple of 128 words.

Tests done on a Series 40 with 7933H disc drive.

Each record was 128 words.

Blocking factor was 32 records per block.

Access method	Records per second
---------------	--------------------

Default file system

(Buffered I/O, 32 records transferred per I/O)	424
No-buff, multi-record (32 records per I/O)	909
No-buff, multi-record (64 records per I/O)	1030
No-buff, multi-record (96 records per I/O)	1333

ADVANCED KSAM TECHNIQUES

The best way to improve keyed, sequential access to KSAM files is to make sure that the records in the data file are in keyed order. This means that you should rebuild the KSAM file as often as possible in primary key sequence. You should, of course, also make sure that the blocking factor of the data file is sufficiently large.

If you primarily use Image but make use of KSAM files as a cross-reference into Image datasets, there is a better way to speed up KSAM access. Whenever a KSAM intrinsic accesses the key file, it also accesses the data file. So if you are reading in keyed order, you will be doing many extra disc accesses to data which you don't need! If you will simply store the pertinent Image information (the key value for master datasets, the record number for detail datasets) with the KSAM key information, you can almost completely eliminate the I/O associated with the KSAM file by opening the KSAM key file as a normal MPE file and reading through the tree structure. (Be sure to read the KSAM manual first, of course!) [SEE FIGURE 8]

Tests were conducted on a Series 40 with 7933H disc.
One 10-byte key, 128-byte records, 20 records per block.

access method	records/second
KSAM FREAD (random data records)	40
KSAM FREAD (sorted data records)	250
Reading key file only (random data records)	630
Reading key file only (sorted data records)	630



If you just need access to the data contained in the data file, you can use the FREADC intrinsic to read the data file serially. (The same speed can be achieved by opening the file as an MPE file and using the normal FREAD.) Unless your data file is already in keyed order, this should be much faster than reading both the key and data files.

Even faster than using FREADC is using no-buff, multi-record I/O on the data file. Open the data file as a normal MPE file and read huge chunks of data directly into your stack area. (Remember that you should build the data file so that its block size is a multiple of 128 words.) [SEE FIGURE 9]

Tests run on a Series 40 with 7933H disc.
KSAM file had one 10-byte key, 128-byte data record
20 records per block.

access method	records per second
KSAM FREAD (opened as a KSAM file, data in random order)	40
KSAM FREAD (opened as a KSAM file, data in sorted order)	250
KSAM FREADC	400
FREAD (data file opened as MPE file)	400
FREAD (data file opened as MPE file with no-buff option)	800
FREAD (data file opened as MPE file with no-buff, multi- record options)	1700

ADVANCED IMAGE TECHNIQUES

Nobody should seriously consider writing programs (except for necessary utility programs such as ADAGER) which bypass the normal intrinsics (DBPUT, DBDELETE, DBUPDATE) that modify Image data bases. In

most cases, however, poor data base performance is not the result of the modifying intrinsics. DBGET is far and away the most used procedure, as the following shows.

The following was collected over a three-week period at our company.

Procedure	Total calls	Relative occurrence
DBPUT and DBDELETE	47,992	1
DBUPDATE	116,182	2.4
DBFIND and DBGETs except mode=2	881,423	18.4
DBGET (mode=2)	20,735,035	432

Even though a serial DBGET (mode=2) is the quickest intrinsic in elapsed time, executing so many DBGETs still significantly degrades performance. Writing a no-buff, multi-record procedure to replace DBGET is probably the best way to improve data base performance. [SEE FIGURE 13]

Tests done on a Series 64/68 with 7933H disc drive.

All datasets had 512-word blocks.

All "MAXDBGET" tests used 10K of the user's stack for buffering.

"MAXDBGET" was run with disc caching off.

Dataset	----- records per second -----		
	DBGET disc caching off	DBGET disc caching on	MAXDBGET no-buff multi-record
"A" (2 records/block)	84	119	869
"B" (3 records/block)	127	166	891
"C" (8 records/block)	173	277	2330
"D" (11 records/block)	239	311	2389

A performance improvement of about 700% is possible using no-buff, multi-record I/O on Image data bases.

ADVANCED SORT TECHNIQUES

Like most other subsystems on the HP3000, the sort subsystem was designed for use on a computer with limited amounts of main memory (the first HP3000 had only 64K). Hence, the sort was written to execute on the user's stack and uses no extra data segments. Unfortunately, if very little of the user's stack is available to the sort, then sort performance can be poor. The stand-alone sort (SORT.PUB.SYS) uses the same procedures (SORTINPUT, SORTOUTPUT, etc.) available to the programmer, thus its performance is also limited because it executes entirely within a single 32K stack. [SEE FIGURE 10]

Just for fun, I wrote a sort procedure which did the following. The sort process launched a write process so that whenever an FWRITE had to be done, the sort process could awaken the writing process and then continue sorting without having to pause for I/O (this gave about a 10% improvement). Next, 8 extra data segments were created so the sort process could do its own input file buffering using no-buff, multi-record I/O (each extra data segment was 31K). An un-optimized Shellsort was also written and all sorting was done in the extra data segments. Comparison of records was done in a 16K area on the user's stack and merging was done into an 8K area of the stack. As the 8K area filled up, it was sent to a 24K extra data segment. When the 24K extra data segment filled up, the write process wrote it to disc. [SEE FIGURE 11]

Tests were done on a Series 44 with four 7933H disc drives and 2 megabytes of main memory.
 There were 245,760 records.
 Each record was 128 bytes.
 The first 100 bytes of each record was the key (but each key was unique within the first 10 characters).

Stand-alone sort (SORT.PUB.SYS).....35.7 minutes

Advanced sort (using extra data segments and process handling for write process).....11.2 minutes

This is a 319% improvement. Please note that about 3/4 of a megabyte of main memory was needed by the advanced sort for maximum speed. I would not recommend this technique for any system which is under memory pressure. (But within 5 years, 3/4 megabyte will seem a laughably small amount of memory to be worried about!)

SUMMARY

So much for facts and figures. What it all means is this: the normal Series 64 is probably processing about 150 Image data base records per second under optimal conditions. If a lot of sorting of the data is necessary or KSAM is being used extensively, 150 is probably on the high side. Converting to no-buff, multi-record I/O can greatly improve performance. Instead of 150 records per second, at least 1000 records per second is achievable. The HP3000 can certainly rival the performance of most data base engines.

STAND-ALONE DISC I/O

DISC I/O 10K WORDS
PER SECOND PER SECOND

—■— - - -◇- - -

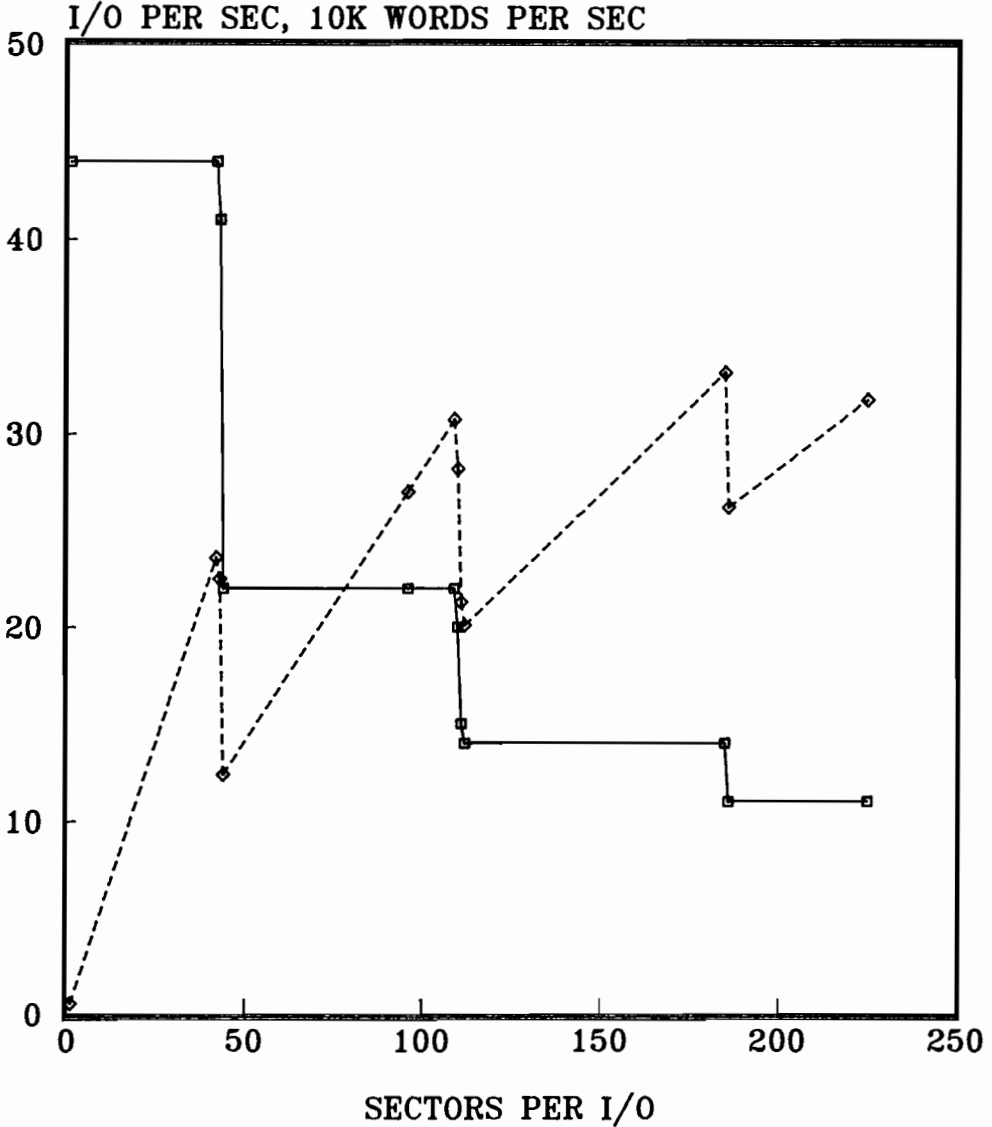


FIGURE 2

MULTIPLE-PROCESS DISC I/O

DISC I/O 10K WORDS
PER SECOND PER SECOND

—■— -○-

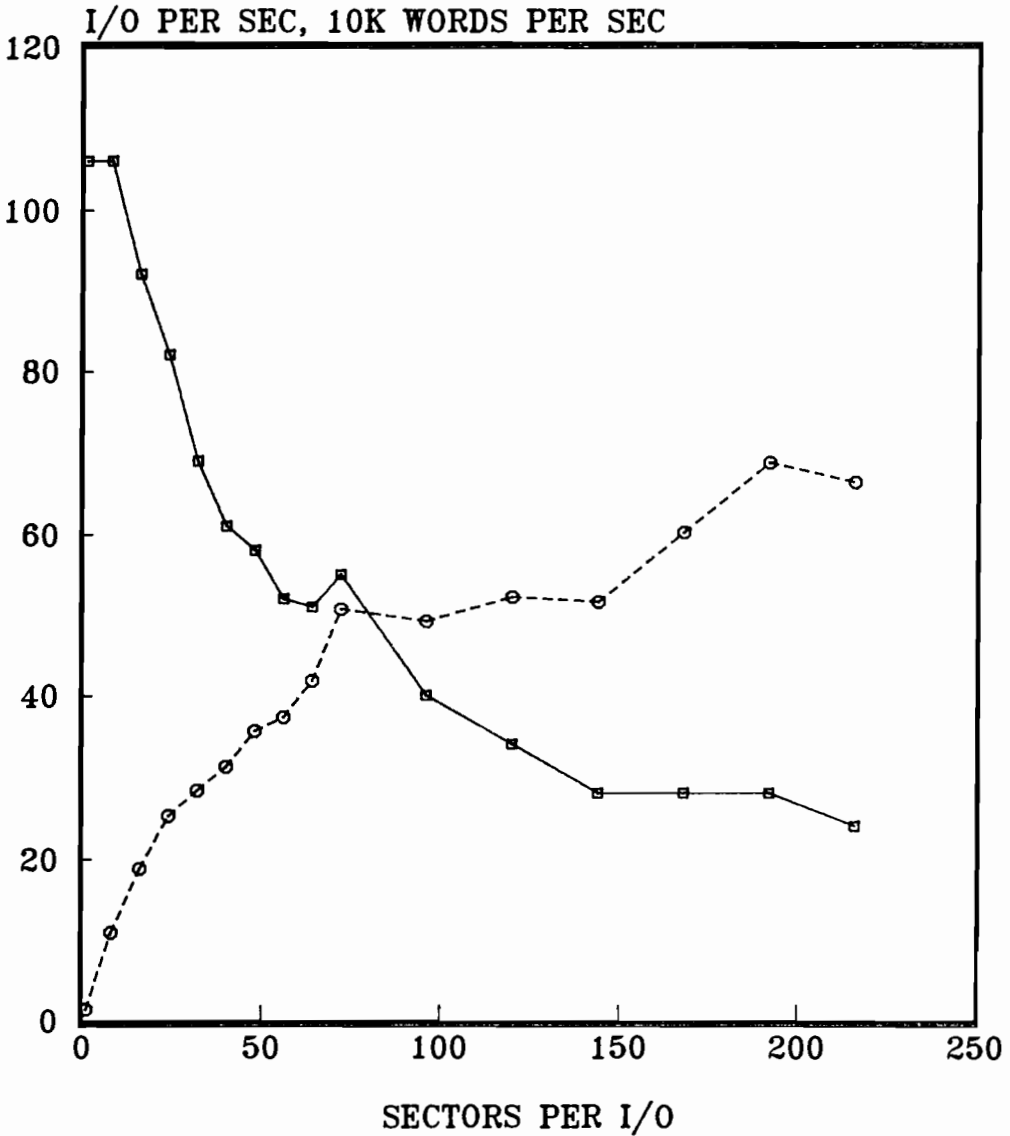


FIGURE 4

DISC HEAD MOVEMENT & DISC I/O

DISC I/O
PER SEC

—□—

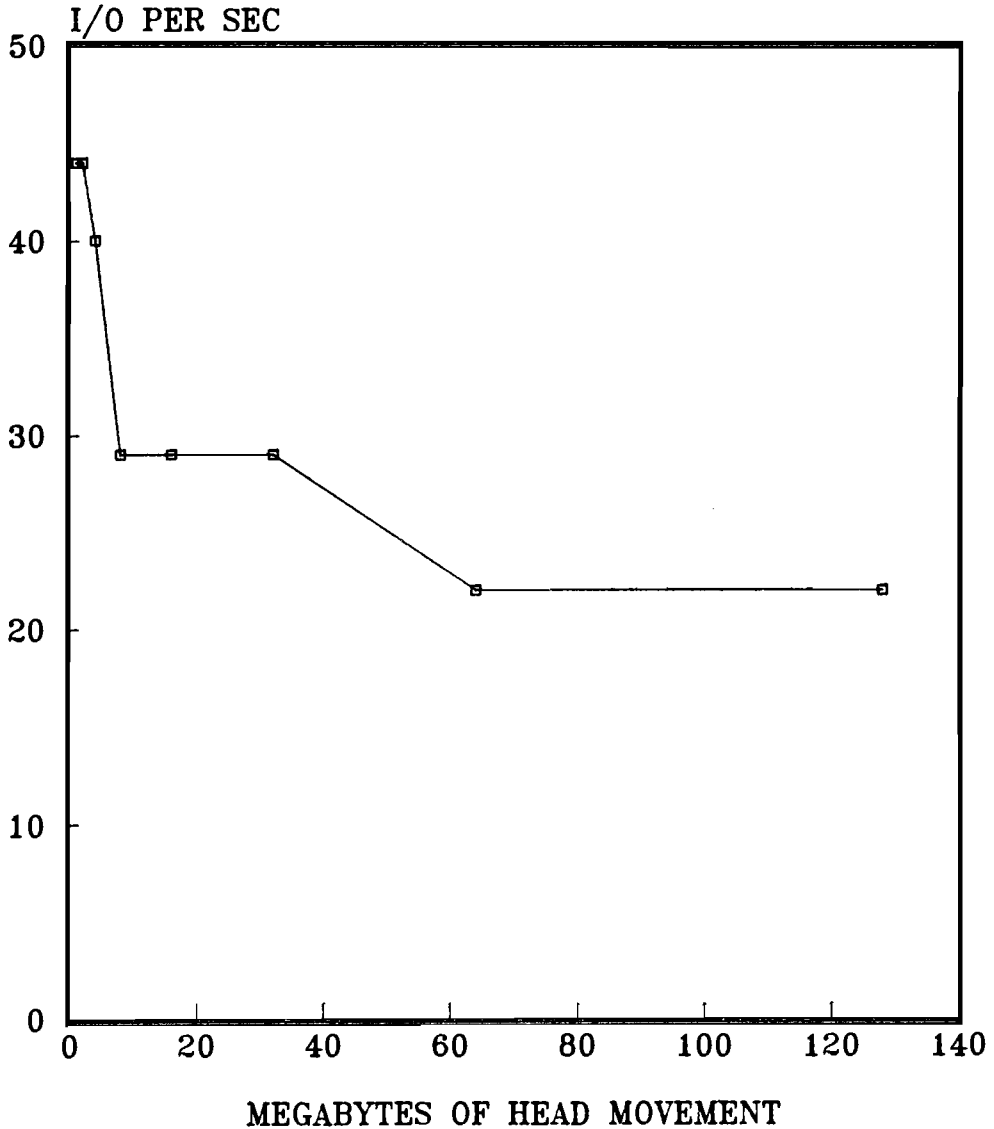


FIGURE 5

BLOCKING FACTORS

RECORDS
PER SECOND

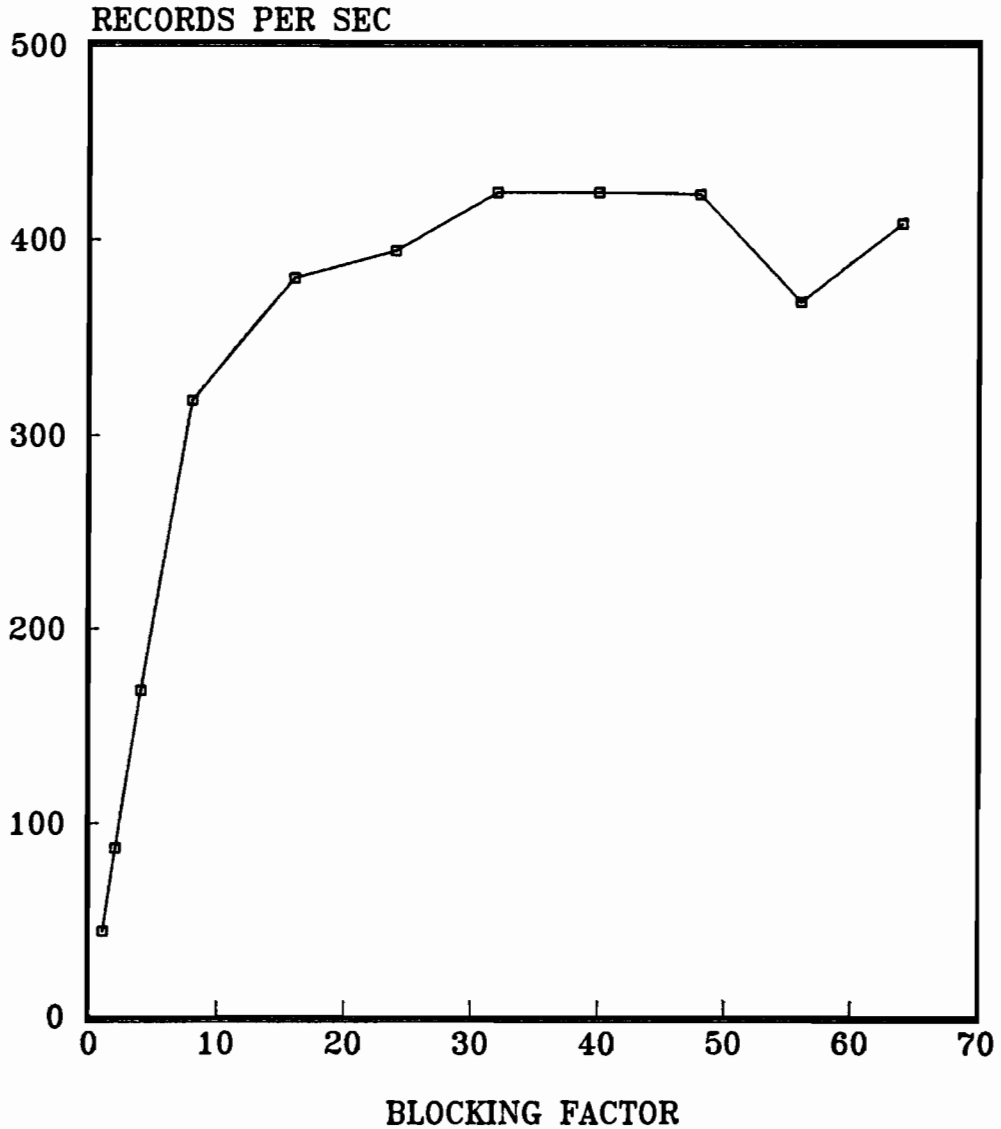


FIGURE 6

DEFAULT vs. NO-BUFF, MULTI-RECORD I/O

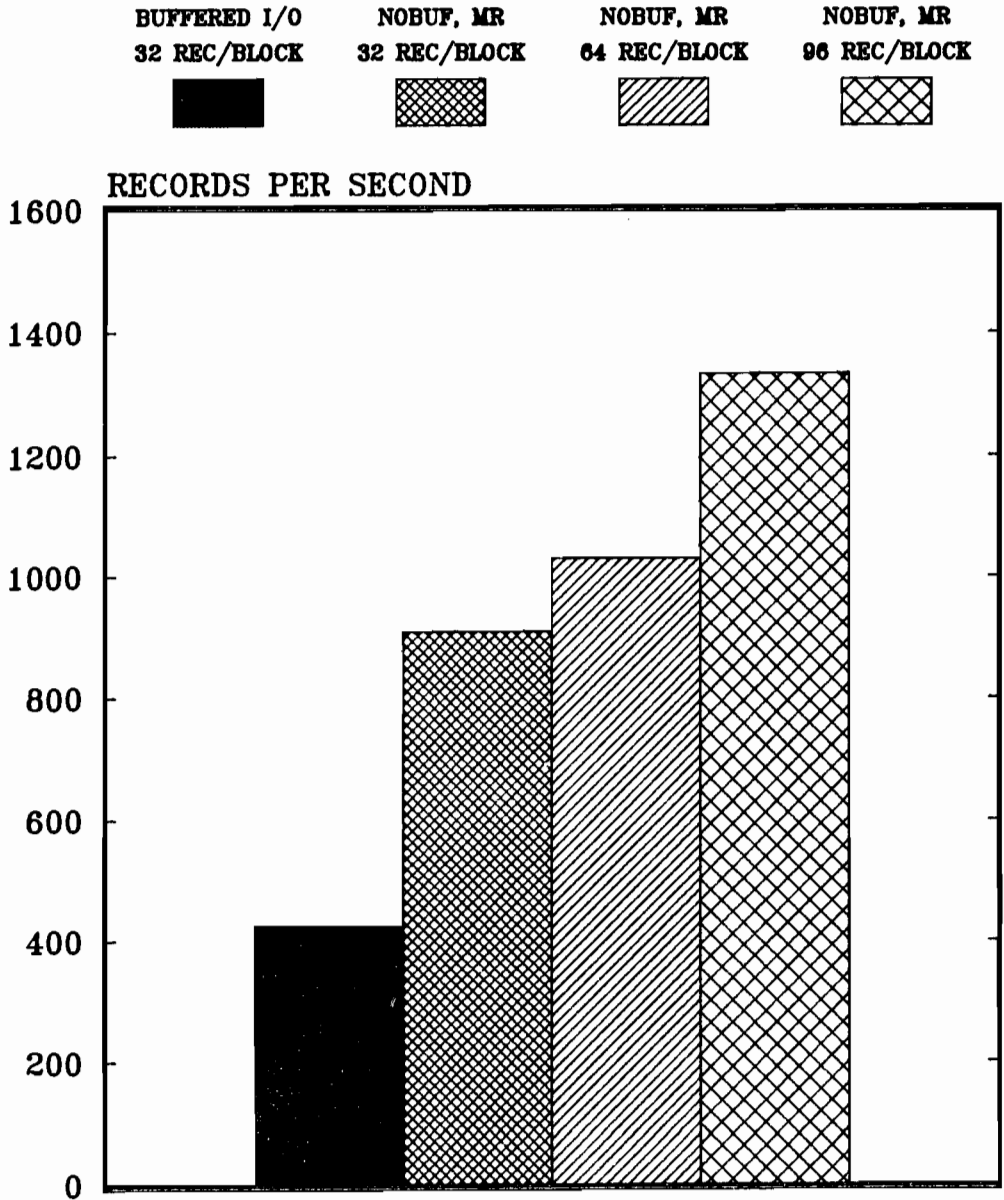


FIGURE 7

KEYED, SEQUENTIAL FILES

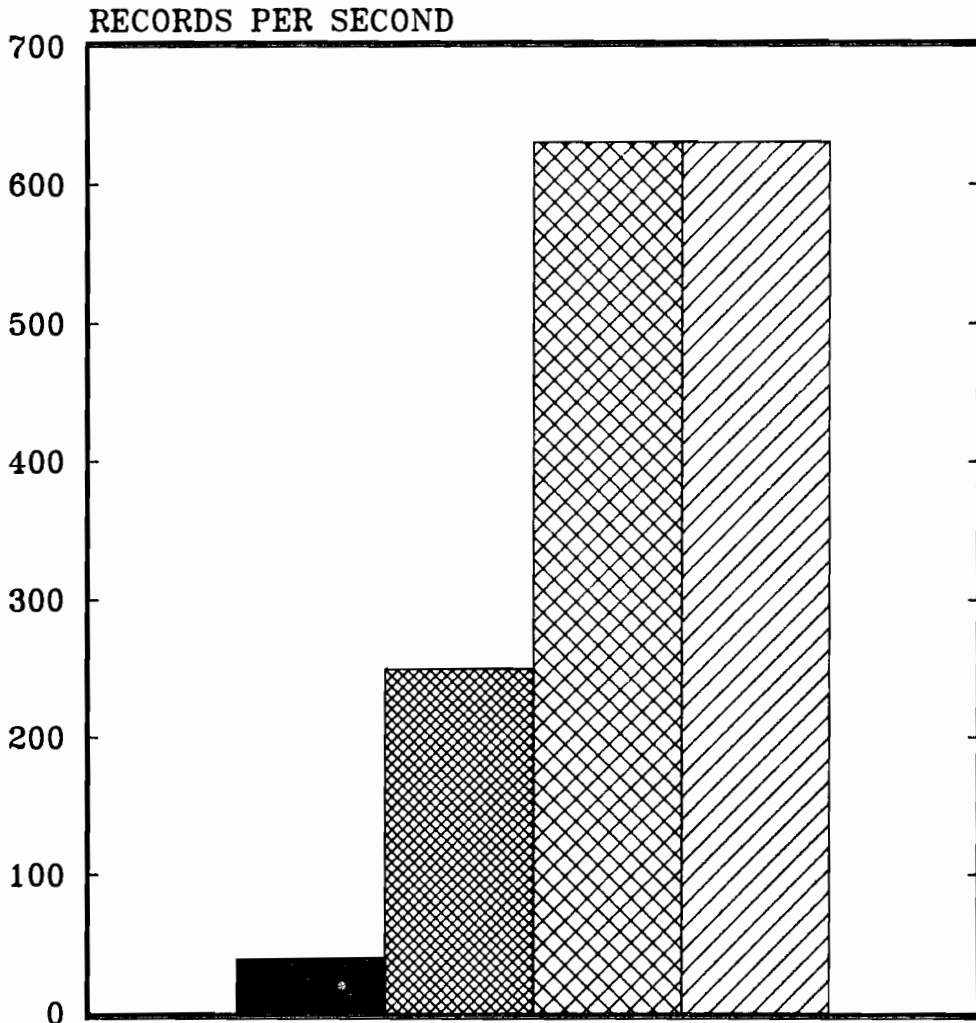
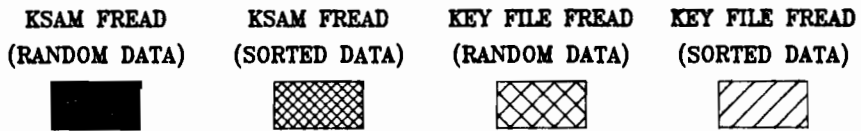


FIGURE 8

KSAM vs. MPE ACCESS OF KSAM FILES

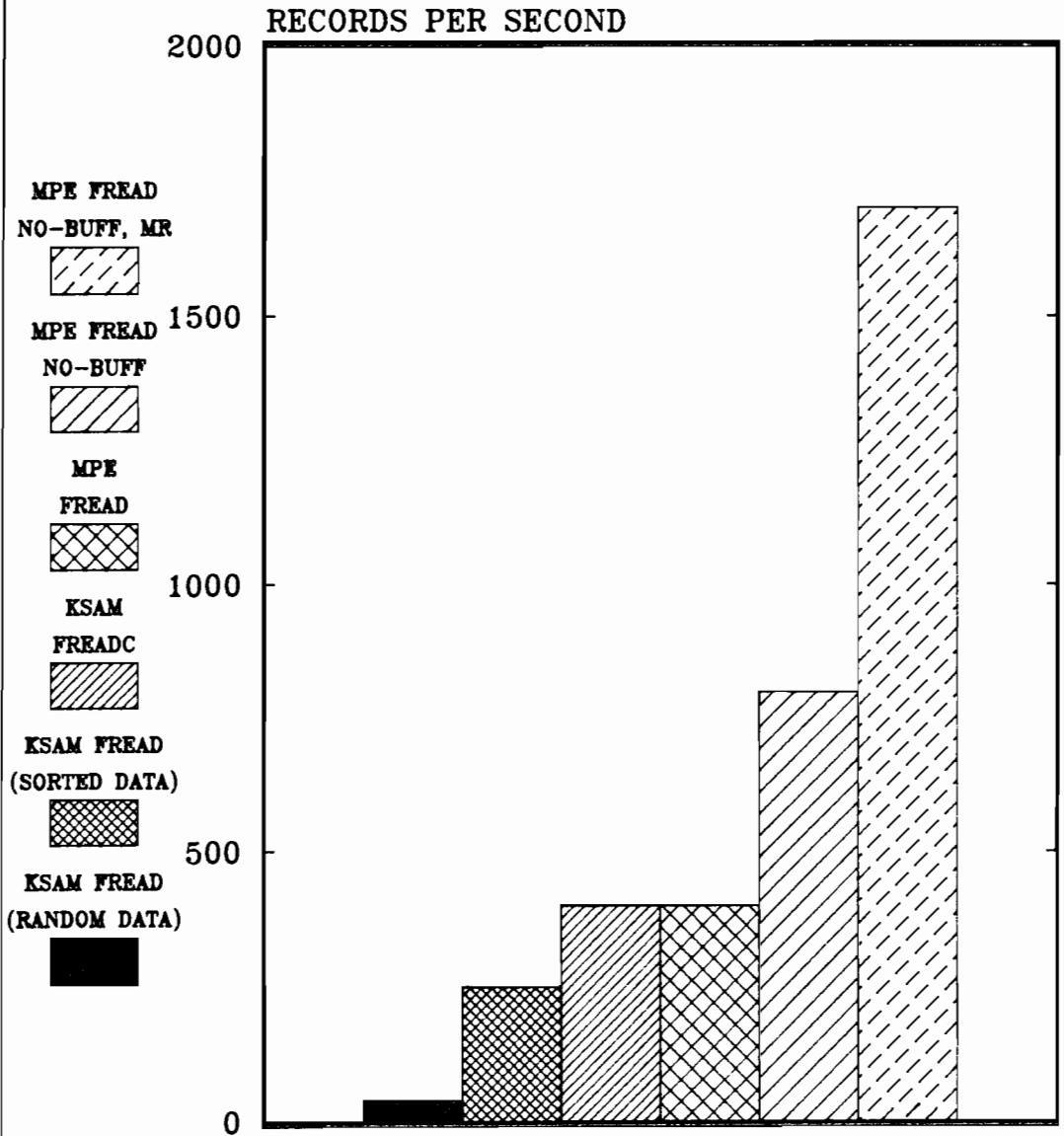
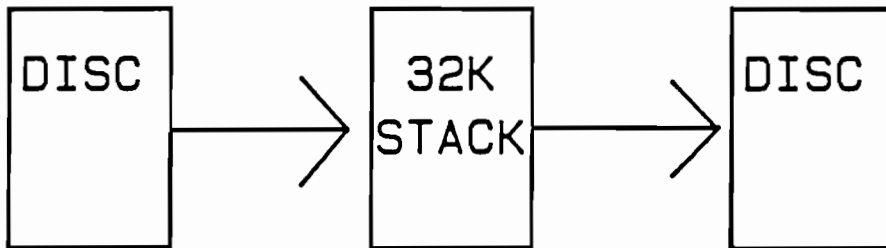


FIGURE 9

Figure 10

SORT.PUB.SYS



Record size = 128 bytes

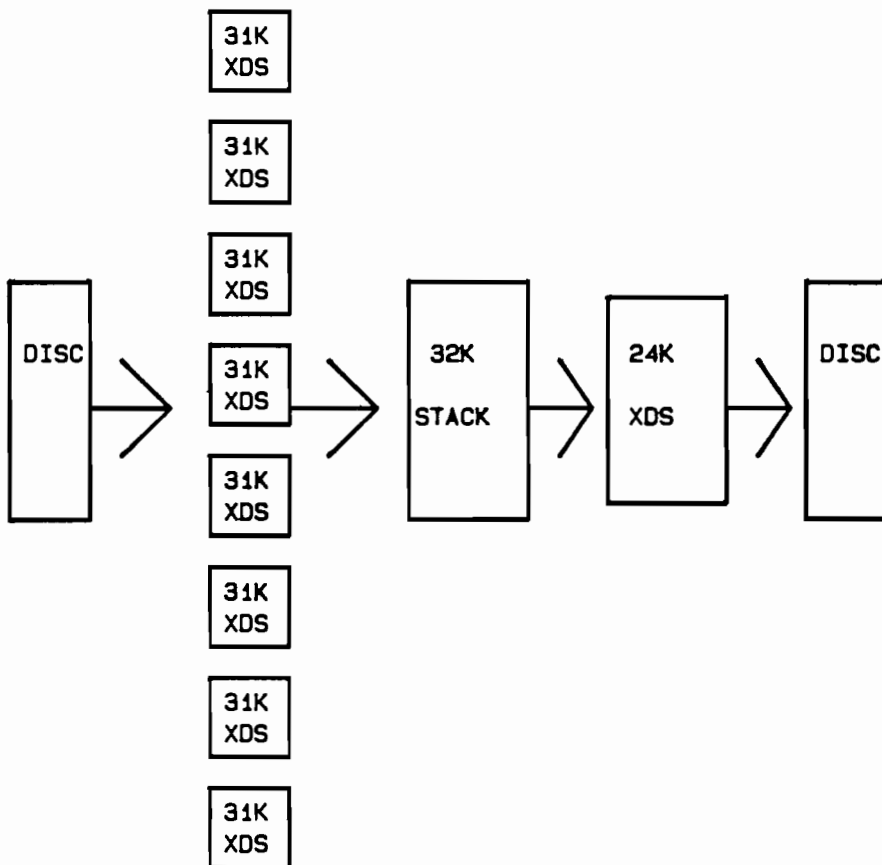
Key size = first 10 bytes

Number of records = 245,760

Elapsed time = 35.7 minutes

FIGURE 10

ADVANCED SORT



Record size = 128 bytes

Key size = first 10 bytes




Number of records = 245,760

Elapsed time = 11.2 minutes

Improvement = 320%

FIGURE 11

SERIAL DBGET vs. NO-BUFF, MULTI-RECORD I/O

DBGET, DISC CACHING OFF	DBGET, DISC CACHING ON	"MAXDBGET" NO-BUFF, MR
		

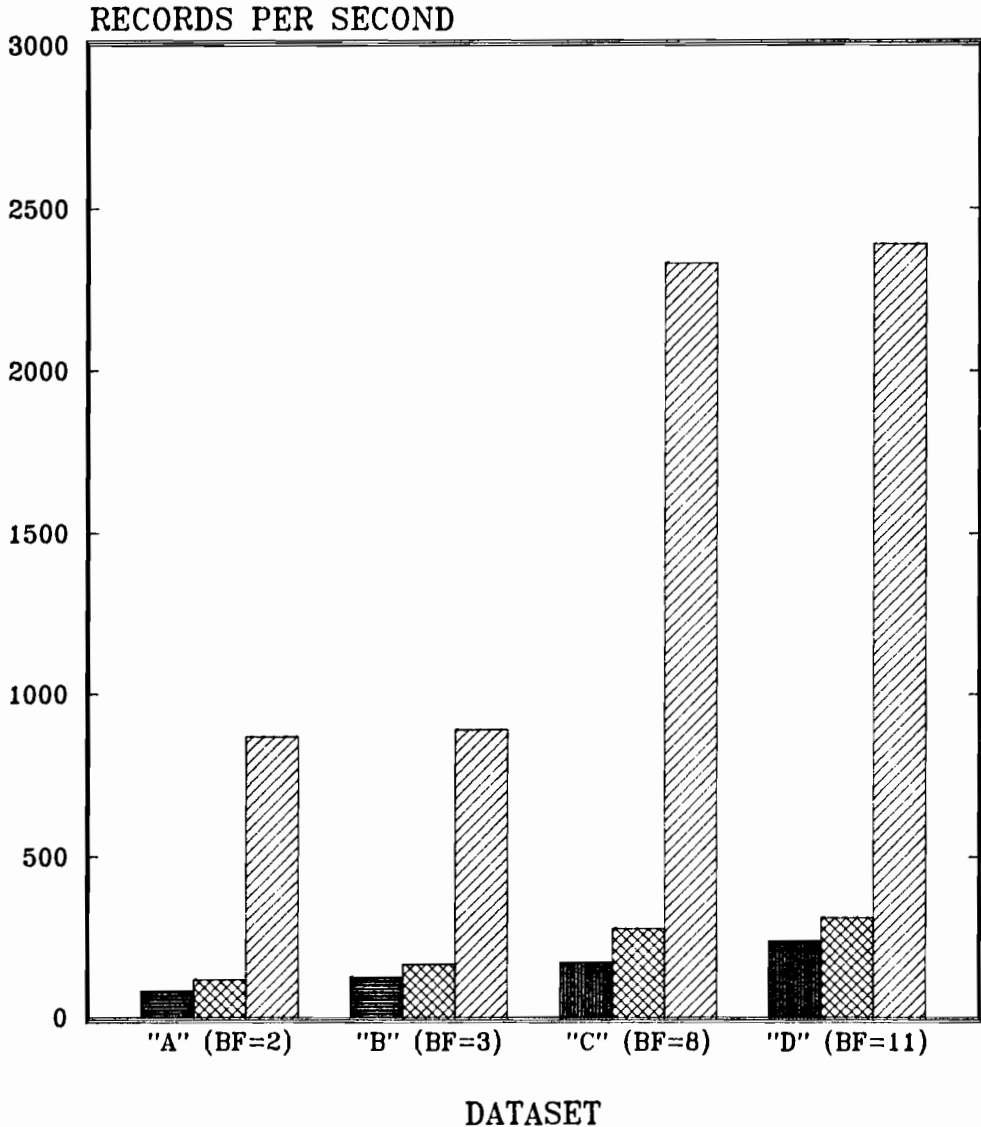


FIGURE 13 (BF=blocking factor)

3031. COMMUNICATIONS IN A MIXED VENDOR ENVIRONMENT:

Wayne E. Holt
Director of Computer Services
Union College
Schenectady, NY 12308

Being a slight treatise on the
foibles, follies, and outright
hazards of listening to folks tell
you how simple it all is.....

PRELUDE

It was a warm, lazy day in the early spring of 1985 - the local INTEREX user group was meeting to discuss the various members' experiences with data communications in a multi-vendor environment. A speaker from Hewlett Packard was present, with the avowed task of sharing "directions" and "market position" with the group.

The speaker had a canned presentation, which had just arrived in the local office. He apologized for not being familiar with the slides, but he felt it important that we hear the latest announcements. It was a typical presentation - mostly gloss without substance. When he came to the part about LAN/3000, a new HP product for 10 Mbps data communications in an IEEE 802.3 environment, a bit of a rustle swept through the audience.

Paraphrasing the script, he intoned "the IEEE 802.3 commitment by Hewlett Packard recognizes that no vendor, including HP, can expect to meet all of a given customer's computing requirements." He paused, and read the passage again, with a pained look on his face

INTRODUCTION

Union College is one of the environments in which it is unlikely that a single vendor could provide all of the computing tools that are required. The setting at Union is quite atypical: it is both a liberal arts college and an engineering college, with a user base that demands a rich diversity of software. The two primary computer vendors on campus are Hewlett Packard and Digital Equipment Corporation, with microcomputers supplied by HP, DEC, IBM, and Apple.

The intent of this paper is to outline the strategy adopted by the College during its Five Year Plan to improve and expand its Computer Services organization. In so doing, a variety of evaluation criteria will be presented, along with a brief overview of the various technologies available today. While I will consistently refer to the VAX as "the other computer", you may substitute something else from Prime, Data General, and so on if you prefer. In most cases, the substitution is appropriate.

This is not an intensively technical paper, such as you might expect from a data communications expert. Rather, from a manager's point of view, it explores the possible alternatives available to those sites that must work with a variety of vendors equipment. Technical issues will be discussed, however, so you may wish to leap to your Datacomm Dictionary from time to time inasmuch as my definitions will be sketchy.

The paper is organized into three sections. The first concerns the issues of basic data communication, particularly connectivity of terminals. The second concerns advanced data communication, with emphasis on computer to computer communication. The last section is a brief recitation of the details of the path chosen by Union College in its quest for universal data connectivity.

BASIC DATA COMMUNICATION

How basic is basic? How about putting a computer at one end of the room, a terminal at the other, and laying a cable between them? This is the most typical method of communicating with a computer. It is most commonly called asynchronous point-to-point, and is generally governed by an ANSI standard known as RS-232-C.

While basically a terminal standard, RS-232-C is often used to communicate between two computers from different vendors. Products such as Kermit, BLAST, and VAXCOM use terminal ports to communicate at fairly slow speeds. VAXCOM, for instance has an effective data throughput of 15 characters per second at 9600 Kbps!

The very concept of a mixed vendor environment implies that you have at least two computers sitting at that far end of the room. You will eventually have to face the fact that at least one, if not all, of your users will want to have access to all of the computers. I will assume that you have chosen a single terminal that will speak well to each kind of computer, such as an HP terminal! If your site is quite small, with modest requirements, you might install a physical switch near each terminal, with a wire running back to each of the computers. This is simple, not too expensive, but very messy and difficult to maintain over time.

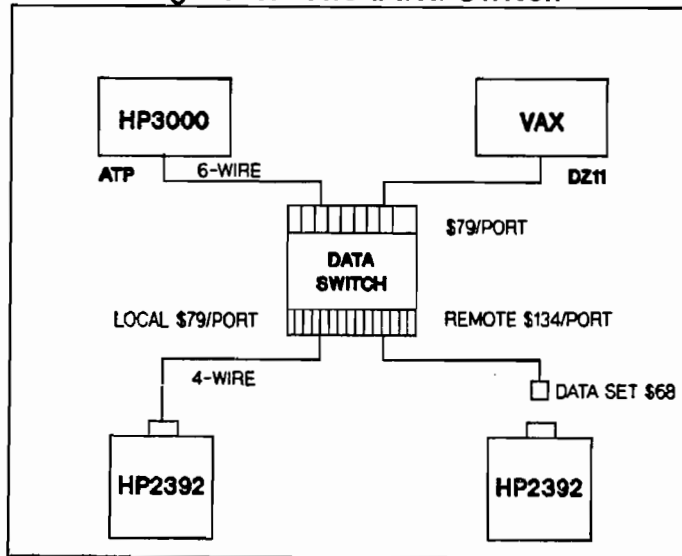
The more elegant solution is to minimize the wiring required and use an electronic data switch to allow the terminal to converse with multiple computers. There are a variety of methods available today, each with different features and associated costs.

The Data Switch. The simple data switch delivers the lowest cost per port when compared to similar approaches. By simple I mean a small centrally located unit that is attached by twisted pair wiring to both terminals and computers. If the distance is great, a short-haul data set may be required to connect the

devices. Micom, Gandalf, Develcon, Infotron, and Equinox are examples of products which use this approach. Most of them also offer more advanced connectivity.

In essence, though, the data switch approach allows the user to talk with a central switching device and choose a target computer. In Figure 1 a typical setup is depicted, with associated costs.

Figure 1: The Data Switch



The example shown is for a Develcon 9000 Dataswitch. Prices may vary significantly from vendor to vendor.

An interesting note at this point is that while the RS-232-C ANSI standard calls for a 25 pin connector, only 3 pins are absolutely needed for connection to the HP3000, with 4 pins preferred. When you are dealing with a data switch, this is just fine for terminals. However, the computer end is another matter. A data switch is actually a multiplexor, and when it drops a terminal it may not drop the computer line --- with the end result of a floating session to which the next user will be assigned, and logged on as you! In order to prevent this from occurring, you must go to a 6 pin cable and allow the HP3000 to operate the port under full modem control. Most switches will use the DTR pin as a signal to drop a connection under this arrangement.

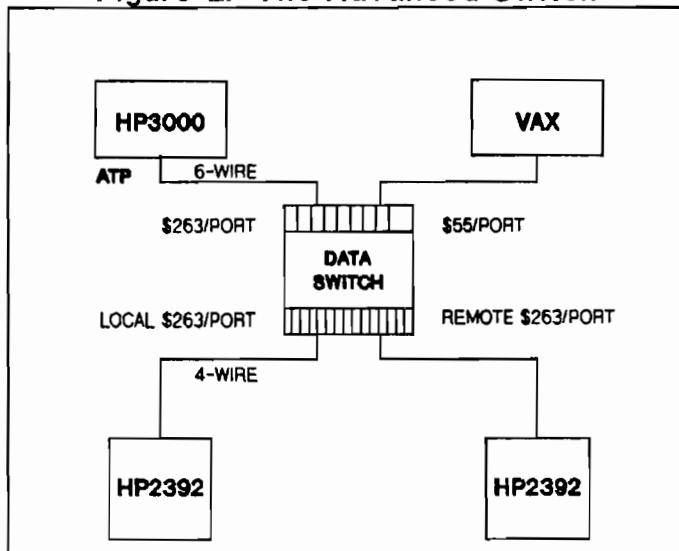
And if you have an ATP, this could pose a major "oops", in that the ATP does not deal well with multiplexed signals. You may have particular difficulty with remote terminals that are spooled on an ATP port, especially those that drop the DTR pin during XON/XOFF exchanges (such as the HP2686 LaserJet). Also, you must now purchase the 25 pin ATP, not the 4 pin version of RS-232-C that HP now offers.

The Advanced Switch. By the definition that I will be using, there is no advanced switch currently available for the HP3000. Please lump the advanced features with which you are familiar into the basic switch, and raise the cost.

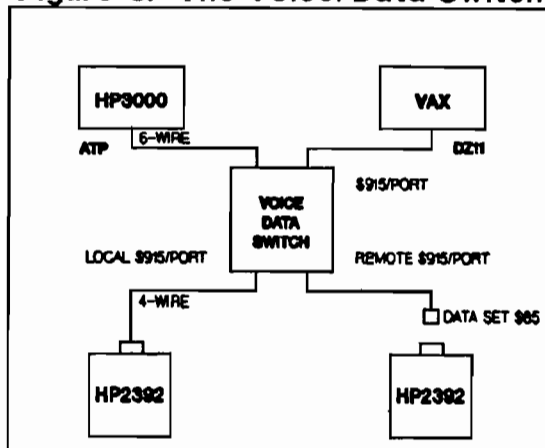
By advanced, I mean a switch device that permits direct memory access on the computer end of the exchange, with the usual features of the data switch on the terminal end. The Xyplex system is an example of such a network, and it works with the VAX family of computers. Direct Memory Access (DMA) is the concept whereby all main computer terminal I/O is eliminated, since the data switch deposits the character string in real memory instead of sending it to the host computer's terminal processors. The advantage of DMA is that the efficiency of the processor is improved, while the benefits of the data switch concept are preserved.

When working with a computer which does not handle DMA, the Xyplex unit operates like a basic data switch. Figure 2 shows a typical setup, with associated costs. Note that the prices are going up.

Figure 2: The Advanced Switch



The example shown is for the Xyplex System. Prices vary significantly from vendor to vendor.

Figure 3: The Voice/Data Switch

The example shown is the Northern Telecom SL-1. Prices vary significantly from vendor to vendor.

The Voice/Data Switch. If you are really serious about ending the clutter of wires at the user's work station, then this approach has merit. Its premise is to integrate the telephone and the terminal into a single data communication link. Intecom, Northern Telecom, AT&T, and Rolm/IBM are examples of these vendors. This approach is definitely expensive, but depending on the size of your organization, it may be the right path for the future.

This technique allows for a single switch (generally known as a PBX) to control both voice and data signals, sometimes on separate wires, sometimes on a common set. The advantage in solving wiring problems for the user is immediately obvious, but the additional benefits for the computer center are not as apparent.

Many computer vendors have joined forces with PBX vendors to form a "standard" for communication links. Like most "standards", there are more than one floating around - but the significance of the idea is that a single cable from the PBX can be connected to the HP3000 (and other computers) to connect as many as 64 terminals.

Figure 3 shows a typical setup, with appropriate cost figures.

So far, I have outlined three different methods for terminal connectivity - and all of them have been generally contained to 19.2 Kbps in an RS-232-C environment.

The next section will deal with faster data communication speed, the kind that is more useful for computer to computer conversations. Quite frankly, you may be asking yourself at this point why I noted at the beginning of the paper that multi-vendor communications was not as simple as advertised. Well, I confess that everything presented thus far is indeed straightforward, and not at all nasty.

But then again, we have only been dealing with your need to get at several computers from a common terminal. Now let's increase the complexity of the situation.

What if you are running a program on a VAX and want to ship some printed output to the Laser Printer on the HP3000? What if you are on the HP3000 using HPDESK and need to send an urgent message to a user on a nearby VAX using All-In-One? What if you want to process a graphics file, or move a data file, or ?

Well, it just got more interesting.

ADVANCED DATA COMMUNICATION

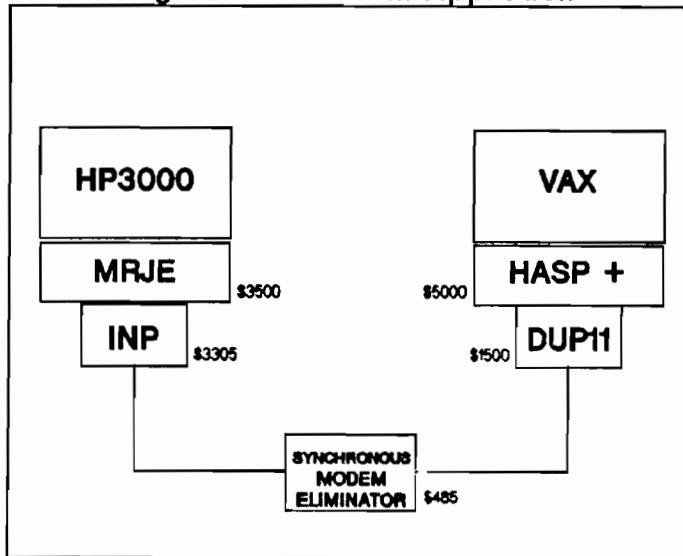
Please recall the HP speaker from the Prelude ... the one who was making the presentation on "directions" and "market positions". When the IEEE 802.3 slide was shown, he noted that the line running off the end of the page was labeled "to other vendors".

Who are the other vendors, and what is IEEE 802.3? First of all, barring IEEE 802.3, the other vendor is IBM. Most of HP's strategy is focused in that direction, and there is a marked presence of tools that enable an HP3000 user to get at the IBM product. So, if you can get your "other vendor" to talk IBM, you can probably communicate.

The IEEE 802.3 is another matter. Ethernet, that universal connect concept being pushed by Xerox, Intel, and DEC, is not IEEE 802.3 at all. When HP talks about bold strides to universal connectivity, it is talking about linking an HP3000 to an HP1000 or an HP9000 ... and maybe to a small handful of vendors that have embraced the standard.

Regardless of what advanced method is used, the ante of what it takes to do anything will go up quite a bit over the basic methods. A new element enters the picture: software. Now that the computers are involved, software is necessary at both ends in order to do something with the data that gets exchanged. The most typical IBM approach, RJE, and the LAN approach will each be discussed separately.

Figure 4: The IBM Approach



The example shown includes HASP+ from Datanex. Prices vary significantly from vendor to vendor.

The IBM Approach. You can't go wrong if you go IBM - just plug it in and run. Oh, if only it were that easy! Should you use RJE, MRJE, or NRJE? Only your Software Engineer knows for sure.

MRJE offers the least complicated approach with the most features. The hardware on the HP3000 is an INP, while the software on a VAX would be a DUP11 or a DMF32. The VAX will also require a software product like HASP+, from Datanex.

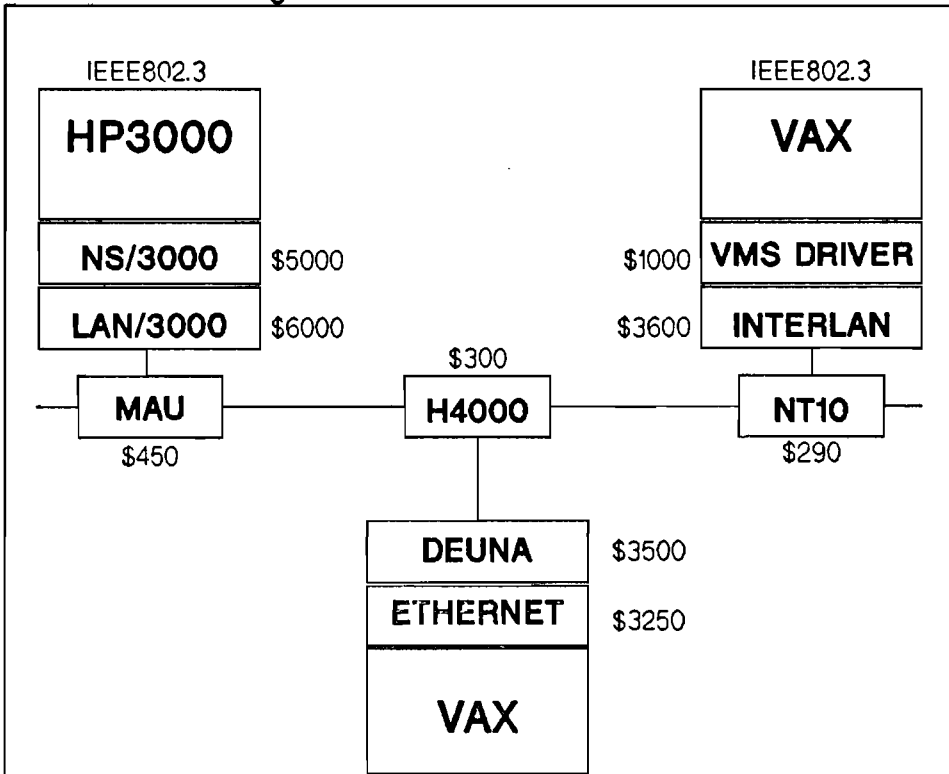
Essentially, a total of 14 channels can be opened up between the two machines, allowing dedicated paths for data exchange. For instance, one channel could be for 1-up Laser formats, another for 2-up, and so on. Data files, electronic mail, and other types of information can be automatically processed. Well, almost automatically.

It is essential that the computers at each end know what to do with the data they are exchanging. MRJE and HASP+ are just part of the exchange path ... they can't do much more than submit files to the spoolers. And on the VAX, even this has problems. The VAX allows embedded carriage control, including TAB

characters - the HP just does not like that at all. Also, each language on the VAX varies in its use of carriage control. The upshot is that you must front-end the HASP+ software with a module that can "standardize" the file for the HP3000 to digest. On the HP3000 side, you must set up different procedures to handle the types of non-print files you receive. Figure 4 is a representation of the data flow between computers.

Aside from these requirements, the connection works very well. The advantage is that the two computers can communicate at a decent rate of speed (9600 Kbps to 56.4 Kbps) in a reliable fashion. The disadvantage is that this is a point-to-point link. That is, only the two computers can share in the conversation. Enter the LAN.

Figure 5: Local Area Network



Local Area Networks. The LAN approach has the benefits of the MRJE technique, with the added benefit of multiple participants in the conversation. That is, a true LAN will allow a large number of computers to share a single data path, and will further allow any member of the group to speak directly to any other member of the group.

However, it is still a bit difficult to get from here to there with an HP3000. There are several problems. First, the situation with a LAN will be similar to that of MRJE in regards to telling the LAN what to do ... the LAN is just the carrier, and when the packet gets from one point to another, someone's software has to tell it what to do and where to go. On the HP side, there is a new product called Network Services that works with LAN/3000 to control the flow. There is nothing as yet on the VAX side. So, how do you tell DECNet what to do with the packet? Nasty business all around.

Furthermore, the HP3000 has yet another problem to overcome when dealing with the VAX: DEC has standardized on Ethernet. It is possible to obtain a Third Party product (Interlan) that will connect a VAX to an IEEE 802.3 network, but to do so will place an even heavier software burden on the VAX. This is because the operating system, VMS, is set up to work with the DEUNA Ethernet card from DEC, not an Interlan card. Interlan will provide a controller for the card, however. Of course, you will have to integrate the two.

For what its worth, the difference between IEEE 802.3 and Ethernet is minor. Both the HP1000 and HP9000 can communicate with either LAN, within the same physical network. That is, it is possible for both the IEEE 802.3 and Ethernet types of nodes to co-exist on the same network, but neither can communicate with the other! Figure 5 shows a configuration that would allow mixed vendor participation in a LAN.

The advantage of "party line" cooperation in a LAN topology cannot be overemphasized. It allows for multiple work stations or computers to freely communicate with each other. An immediate benefit to a site with an HP2680 laser printer is a central print station that could serve all computers with minimum fuss and wiring.

THE UNION APPROACH

In 1982, Union College embarked upon a Five Year Plan to improve computing services. Included in this plan was a comprehensive data communications network, considered essential to the overall success of the Plan itself. It was recognized early that the data communications market place was rapidly changing, and that early decisions might be obsolete before the plan was completed. This proved to be the case.

The decisions made at the beginning of the plan involved a relatively small monetary investment in exchange for a significant improvement. A simple data switch was purchased in order to provide access to the three computers then on campus: an HP3000/64, a VAX 11/780, and a Burroughs B6805 mainframe. For the first time, all users could access all computers. Because of the relatively small size of the campus, it proved to be very economical to pull twisted pair cables to the primary academic

buildings and the administrative areas. This was not the case with dormitories and remote facilities.

By the third year of the plan, over 300 terminal locations were served by this RS-232-C network. Faculty offices, public terminal clusters, and administrative areas were included. At this time, the College sold the B6805 and purchased three VAX 785s, two VAX 11/750s, and an HP3000/48.

An Ethernet was installed in order to connect the VAX computers and a variety of engineering workstations. An MRJE link was established between a VAX 11/785 and the HP3000/48, in order to use the two HP2680 Laser Print Stations, and to promote file exchange. The same cable that contained the Ethernet also was used to establish an IEEE 802.3 LAN between the HP3000 computers.

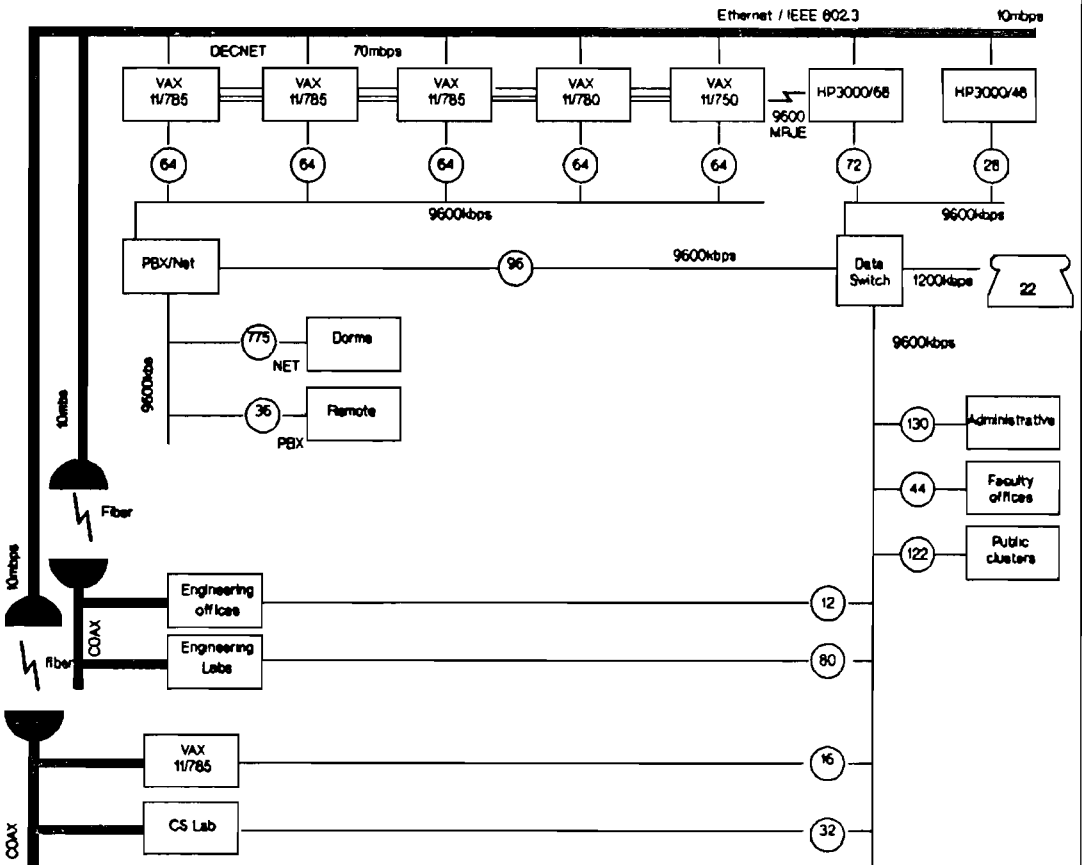
Now in the fourth year of the plan, the College has expanded the number of terminal stations to over 400, and is in the process of purchasing a voice/data PBX. The data portion is intended for remote data access. At the same time, a parallel data network is being planned for the dormitories, using the same wiring plant as the telephone system. This will add about 775 new terminal outlets to the network.

The network at Union is known as GARNet/UC, for the General Access Resource Network. Figure 6 is a schematic diagram of the network showing the points where the various parts of the network intersect.

Why so many methods? Each is cost effective for the targeted purpose for which it was designed:

- o RS-232-C very inexpensive for a large number of close users, with a proven technology that will be cost effective for many years...
- o Ethernet an effective way of very high speed data transfer between certain computers, it also represents the best future for computer to computer data communications throughout the campus ...
- o MRJE a reluctant choice, but the only effective technology for an HP3000 in a mixed vendor environment ...
- o PBX an expensive choice if this were the only data path, but for the purpose of serving only very remote parts of campus it is actually very cost effective.
- o Common Wire less expensive than voice/data, it uses the same wiring plant as the telephone, but still provides the features of the RS-232-C method.

Figure 6: GarNet/UC



There is no question that a certain commitment must be made to having staff that can deal with the data communications issues. The problems of having multiple methodologies are less than you might imagine, however. These methods are quite reliable in general, and once a certain amount of initial difficulty is overcome in each area, the techniques required to maintain the network are manageable. If the hardware fails, a service contract will repair the damage. If the software fails, the original vendor can usually provide help in diagnosing the problem. Proper training of the end-user has proven to be the most difficult, and important, part of maintaining the usability of the network.

Any time a site decides to purchase major computing equipment from a variety of vendors, the caution flag should be displayed. No matter what the original intent, the time will come when someone will order the combatants to actually speak to each other. And talking just isn't what it's made out to be ... at least, not yet!

3034. Why Software Projects Don't Quite Succeed

Robert R. Mattson
WIDCO
9545 Delphi Road
Olympia, Washington 98502

INTRODUCTION

When we start a software system project our goal is success. If one has been around the business or read the common press we know, that too many times for comfort, we don't quite succeed. At the same time many of us may have been associated with the project that we "know" was successful but was not viewed that way by other important people such as management. What is the scenario that takes place in most software projects and what are the reasons why they don't quite succeed as we had planned.

THE GENERAL SCENARIO:

- Management has by some process decided to do a software project.
- Sometime before or after the decision to proceed there has been the establishment of deliverables and the time and cost "plans/budgets".
- Management believes that software projects are like any other project and you should be able to manage them as such.
- Following managements lead and the "principles" of project management you prepare a plan detailing deliverables, tasks, responsibilities and time.
- Then you dutifully go to producing the product.
- What happens next....?
 - . the time budget do starts to fall apart and/or
 - . the cost (usually labor dollars) go over budget and/or
 - . tasks pop up that were not "planned" for and/or
 - . people get assigned other places and/or
 - . deliverables, specifications change (usually means added to) and/or
 - . etc, etc
- Then we start explaining to management about the "realities" of a software project.
- Since management knows that software projects are like any other project the later "realities" sound like excuses.
- We redouble our efforts since...it's probably our fault we just don't know how to manage well enough.
- We cut quality while doubling our efforts to meet the ORIGINAL time and cost budget.
- Finally the project is over and we hum a sigh of relief as we lick our wounds for not doing a better job...and go onto the next project.

Does this sound all too familiar. It doesn't sound all that appealing and yet I fear this scenario is played out over and over again. The net result of this scenario is that all too often the project is judged as not successful by one or more key people. So what went wrong? Something obviously did since we didn't succeed by managements measure. I believe that in a sense we are the "victims" of the gods of projects and their diving "principles". These principles turn out to be "myths" when applied

to software projects. We are caught between the rock and the hard place. So what are the reasons that we have got ourselves into this situation and don't succeed as often as we would hope? I believe the reasons can be grouped into three areas:

- 1) Technical Reasons
- 2) Definitional Reasons
- 3) Project Management Reasons

As with most things, it is not usually the case that there is only one of these reasons involved when projects have problems. What I'd like to address are the latter two reasons. I'll leave the pure technical reasons (ie the system just doesn't do what is expected) to other people to discuss and solve. I want to discuss the two non-technical reasons because I believe they are the least understood and maybe the most common.

DEFINITIONAL REASONS OR"SUCCESS"...ITS MANY MEANINGS

I want to focus on what it means to say that a software project was a success or failure. This issue turns out to be an interesting one because there is no one definition. In other words project success is measured differently by different people. Let us explore these different views of success since this is many times a major contributor to why projects don't quite succeed. Factors in Measuring Success: The following are the most common categories used in the measurement of a projects success:

- a) TIME - addresses whether the project and/or major phases was completed on or before the date in some "plan".
- b) BUDGET - addresses whether the project or phase was done at or below the cost planned.
- c) EXTERNAL QUALITY - addresses whether the software does what it is suppose to and the number of "problems" with it.
- d) INTERNAL QUALITY - addresses some measure of how well the system is built and its anticipated maintenance costs and flexibility over time.
- e) STYLE - addresses the appearance of how the project is managed and whether things are under control.

The following chart outlines the different way that different groups "weight" these different measures in "toting" up a project success "score".

----- Percentage Weighting on Success Meaures -----

	On Time	Under Cost	External Quality	Style	Internal Quality
Mgmt	40	40	5	15	0
Buyer Mgmt	30	50	15	5	0
Users	10	0	80	10	0
Systems	20	20	30	0	30

As with all such generalizations, there are exceptions and deviations.
But

I believe this is close enough to the reality for making the following points.

- + Management measures project success by time and cost and style.
- + The users management measures project success by time and cost then external quality.
- + Users measure success by external quality.
- + Systems people measure success by all of the above plus internal quality.
- + Style is mostly the extent that a project manager expresses and radiates confidence and gives the sense of being all knowledgeable and in control. To a large extent it means not trying to explain to management, etc why there belief in "myths" are not correct. This latter activity is viewed as bad style.
- + When people differ on their scoring measures they will access the final score (success or failure) differently.
- + Systems projects don't quite succeed in part because the different participants use different scoring methods. And managements scorecard ultimately counts more than anybody elses....always has, probably always will.
- + The biggest conflict is because management views time, cost and style as being 95% of the measurement of project success while system people view these as only worth 40%.
- + This difference in management and systems scoring means that system people can succeed by their scorecard and fail according to managements. In otherwords we could put in a very good system which will function well and have low maintenance and ongoing cost but be viewed as not successful because we exceeded "time and budget or our style wasn't right".
- + One reason management uses the measures of success they do is that there are clear "right and wrong". For example you either made the time for delivery or not. You either were under budget or not.
- + Management should place more weight on the quality aspects of projects. The reason is that in the long run it is the external and internal quality that gives one the true cost of a system. But management would have to understand far more about the projects to do this and human nature wins out most times. Additionally, we don't keep track of system life cycle costs so the added cost of a poor system over time is not documented. Finally, it is generally the case that american management is very near term oriented. This orientation works against long term system life cycle evaluation of success.

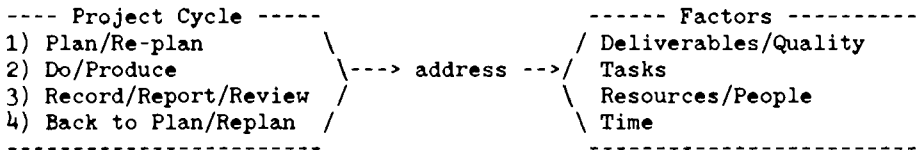
- + Until management gets better educated, systems project managers need to understand how they are being scored. Then we can adjust our behavior to the extend we each feel is justified.
- + Understanding many ways the game is scored is essential to success.
- + Most of the issues related to success or failure in systems project relate to the areas of cost, time and style...or managements measures.

The bottom line of achieving success is that we must understand how we and the project are being measured; then make our choices regarding how we play the game.

Now let's address how project managements reasons affect success.

THE PROJECT MANAGEMENT REASONS OR ..."PRINCIPLES" AND "MYTHS"

A project is non-routine undertaking to accomplish some goal within some timeframe and within limitations on resources used. A system project has these key factors....a deliverable, quality, tasks, time and resource/people (dollar) constraints As part of the management of a project there is a cycle that is followed involving planning/replanning, doing/producing, recording/reporting and reviewing. During each cycle the factors outlined above need to be addressed. This process is illustrated below.



It is as we address these factors in the project cyle that we come up against the principles, myths and realities of software project management. It is in our inability to succeed in applying these "principles" that we have problems meeting managements' success measures. I would like to address some specific principles and myths that are common to the area of software project management. In almost all cases the myths have a sense to them. This sense is that if they were true then achieving success in managing the projects would be enhanced. They are almost universally false due to the nature of a software project. To SOME extent the myths are false because we haven't the techniques and tools that allow us to do w.at the myth says. To the extent it is a matter of tools and techniques we can make improvements. To the extent that the myths are false due to the nature of a software project itself then we have to learn how to work around the realities. The myths that follow are not all inclusive but hopefully give some feel for the challenges and pitfalls of software system project management.

+++++

Myth 1: "One begins a project by defining all the deliverables."

Reality: It is not possible to define all the deliverables at the beginning of a software project and in fact our understanding of how we define a deliverable is very limited.

Facts:

- + Deliverables are extremely hard to define at the beginning of a project. Many times this is due in large part to only a partial understanding of what is going to be produced.
- + The larger the project or the more complex the system the harder it gets to define all deliverables up front.
- + Systems are made up of a multitude of small inter-related deliverables.
- + The definition of a deliverable should include quality but usually doesn't.
- + The quality of the deliverable many times affects the time to do and the quality of another dependent deliverable.
- + When we don't define quality then we tend to focus our task management on meeting time and budget constraints to the detriment of quality.
- + If we do not define every deliverable we will not be able to define a task and subsequently time/resources to produce the deliverable.
- + The missing of deliverables is one of the prime reasons that system projects are not successful by time and cost measures.

Suggested Approach:

- + Attempt to define as many deliverables as possible as early as possible.
- + Continue to define new deliverables as they become known. This is a must if one is to keep on top of task planning and time requirements.
- + Make deliverables the key to your task planning. (ie. the producing of a deliverable requires a "task" and vice versa.)
- + Keep track of what deliverables you actually produced on a project for use in planning later projects.
- + Strive to define the quality of all deliverables.

+++++

Myth 2: After defining the deliverables we can define all the tasks.

Reality: It is impossible at the start of a project to define all the tasks to be done in a project at a level required for accurate time and resource planning.

Facts:

- + Task definition should flow from the list of deliverables but as we've seen...the list of deliverables is never complete and most times very incomplete therefore task plans are never complete.
- + So we are faced with the reality that not all tasks will be defined at the beginning of a project..therefore time and resources will be understated.
- + Additionally most tasks plans are far too "coarse" to accurately allow for time and resource planning. A prime example is time allowed for a "program".
- + Defining tasks at a coarse level only gives an illusion of planning.

Suggested approach:

- + For task plans to be of the most benefit in defining time and cost they must be as complete as possible and as "detailed" as possible.
- + Define tasks in a way that accounts for all significant steps and intermediate deliverables.
- + Develop gross tasks plans , but revised within the next two-four week window for purposes of people planning and doing.

+++++

Myth 3: "Then allocate the resources (people) to the tasks"

Reality : A person is not a person, a task is not a task, etc

Facts:

- + How long a task is going to take depends on who is going to do it.
- + The concept that people are interchangeable is absurd but we ignore this fact in the "experts" approach to project planning.
- + Therefore when planning the time and resources to do a task we must know who will do it inorder to have ANY HOPE of accurately arriving at reasonable figures.
- + Tasks and deliverables also have a great deal of variability. This means that one program is not just like another program. When planning we need to be aware that tasks and deliverables are not all alike also.

Suggested Approaches:

- + Develop people into their own estimators. Given the right history they are the ones that can do it best.
- + Realize that if you have to estimate a task's duration prior to knowing who will do it or exactly the nature of THIS task that there is a range of probable time and plan accordingly.

+++++

Myth 4: "We then define how long a task is going to take."

Reality: Our ability to define how long a task is dependent on factors we do not most times have the knowledge to accurately assess and inter-relate. Further we confuse the meaning of time.

Facts:

- + The time to do a task is dependent on the person(s) doing the task. Therefore, without defining exactly who will be doing a task it is very likely that the estimate will be wrong.
- + There is at least three types of time related to projects.
- + There is man-hours.
- + There is elapse time.
- + There is man-days.
- + All three will yield different values most of the time.
- + Using any one will probably lead to failure by the time standard of success.
- + Time is not recorded accurately.
- + Almost universally time is recorded to form the basis for reimbursing the doer and to "prove" the project plan was right.
- + Time reported is a juggled for all types of reasons which distorts it and deminishes its value for later project planning use.
- + People score there timecard depending on the way the reward system is setup.

Suggested Approaches:

- + Learn the different meaning of time. Then plan and record time accordingly.
- + Keep a meaningful history and use it in your planning.
- + Make the keeping of time something that people see as helping them rather something that has negative consequences. Part of this is done by realizing that people need a place to allocate time not charged to "project" tasks.

+++++

Myth 5: We can monitor people to achieve our time, cost and quality.

Reality: People cannot be monitored in a system project to assure a quality project is achieved within time and cost budget.

Facts:

- + When people are monitored by other people they don't monitor themselves.
- + Looking over someones shoulder will only make them do something until the person goes away.
- + Getting people to be their own monitor of quality and time is the only way to achieve consistent quality and productivity.
- + To achieve the goal of self monitoring we need to give the doer tools that they can use to do their own task management.
- + The Japanese have shown that quality is the result of the attitude of the people doing the work plus understanding of the numerical measurement of quality.
- + Quality can not be reviewed into a product by supervisor or project manager.
- + Quality is achieved by developing people who value and understand the multi-faceted nature of quality.
- + Quality is achieved by reinforcing people for producing quality in addition to making milestones and budgets.

Suggested Approaches:

- + So how do we get people to be their own monitors and quality controllers? I don't have all the answers but I believe a lot has to do with the way we score our peoples performance. We need to be sure our value system reinforces the people along the self monitoring lines and not place that "burden" on the project managers shoulders.
- + Additionally, we must understand that although all people can be "changed" that some people have more of this self responsibility. A "easy" way to approach this problem is JUST hire this more responsible group.

+++++

Myth 6: "Using the "....." method will assure system project success."

Reality: No single cookbook recipe will lead to success in a system project.

Facts:

- + Every system project is a unique blend of system, human, administrative and other factors. One must learn to fashion a successful project which is appropriate to the circumstances and measures of success.
- + The belief that there is one "methodology" that will lead to successful systems is false. The proponents of these systems would have a difficult time applying their methodologies in every case. The mere fact that there are so many ONLY answers should tell us something.

Suggested Approaches:

- + Learn the various approaches to project management and then as with the systems themselves mix and synthesize the appropriate technique.

Myth 7: "Time, cost and deliverables for a systems project are all accurately definable at the beginning of a project."

Reality: The nature of the deliverables, tasks, resources and time make their accurate definition at the beginning of a project impossible.

Facts:

- + As seen above, management measures success by how well we conform to the budgeted cost and time. Because we cannot accurately estimate these at the time they are generally set (the beginning)...we stand a good chance of failure unless allowed to set wildly large estimates.
- + As seen above, however, most of the factors having to do with scoring well by managements measures are extremely difficult to "manage" in a software project.

Suggested Approach:

- + Strive to use an incremental budgeting approach. This means that at the end of each "phase" there is a new budget prepared. Get management to buy off on this approach. It will be viewed as poor from a style standpoint by management. However your still ahead because management weights the meeting of time and cost budget more than style.



+++++

Myth 8: "System projects are like any other project and should be manageable by the techniques that work for those other projects."

Reality: System projects are of a nature that is fundamentally different from projects to which they are compared.

Facts:

- + The toughest consequence of this myth is that management, users and buyers apply the same expectations to system projects as other projects.
- + System projects produce "synergistic" products. This means that the people interact with the products in such a way that alters the acceptability of the product even if it is delivered to original specification.
- + The users and buyers in general have a poor knowledge level when dealing with system analysis, design and development issues.
- + The users and buyers, for whatever reason, do not spend adequate time in the review of specifications and almost always "sign-off" without truly understanding the product to be delivered.
- + The expectations of management, buyers and users do not bear a reasonable relationship to the realities of developing "synergistic" products.
- + The building process required to develop a system is not understood by most none system people.
- + The process involved in developing a system is hidden from view and so everything appears to be magic. Magic isn't suppose to take time and cost money and have uncertainties.
- + In addition, as outline in earlier myths and realities, the means to "manage" in the manner expected for "projects" is just not available.

Suggested Approaches:

- + The most obvious "solution" is to educate users and management to all that has been outline here. The realities are probably that this is not a very likely event. When we do try to educate we usually lose style points for obviously trying to make "excuses".
- + The approach with more chance of success is to work on achieving a good understanding of all the myths-realities and techniques for dealing with these in software projects. By balancing the realities of what management expects with our professional sense of values on a project we can achieve the best compromise.

CONCLUSIONS:

This paper dealt with the myths and realities of project management for software projects. Much of what has been written and is taught about project management is of dubious value. The standard "truths" and methods are counter-productive many times by making project managers abandon all project "management". This abandonment of project management usually occurs when the manager sees they can't do what the "experts" say they should. It was the goal of this paper to express that the "emperor has not clothes". If you've secretly harbored doubts about your abilities to manage software projects because you could not do what the "experts" say then hopefully you have gain some insight and sense that you're not alone. Software development projects are fascinating challenges. The challenges are not so much technical (the computer doesn't have bad days) as they are "social" and "political". We work on machines of unbelievable speed and "power" but which carry few "myths" in their memory....for myths we need humans. Finally, may all your projects be successful!

3036. The Sorted File Access Method

Wanyen Chang
Longs Drug Stores, Inc.
141 North Civic Drive
Walnut Creek, California 94596

I. Introduction.

Hewlett Packard's KSAM is a powerful file access facility that allows an application do both random and sequential I/O to a data file. The random I/O operations are supported by using a key file organized in a structure called B-tree. It has been shown [1] that for a completely random access pattern, B-tree offers the optimal performance in terms of the number of I/O required to locate the key. Thus, if an applica- tion has no idea about the distribution of key values, and the percent- age of output operations is significant as compared to that of input, KSAM would be the ideal tool to use.

The cost to support this generality, however, is also significant. For one thing, the disc space for the entire key file has to be allo- cated at create time, even though no key exists yet. For many appli- cations, the key file never reaches 80 percent of the total capacity. For another, the number of accesses required to locate a key uniform- ly increases as the key file grows. This fact coupled with the amount of memory used for buffering the key record blocks often re- sult in a slow response time. This is especially apparent in a multi- user, on-line environment. In this case, the application is forced to engage in a lock/unlock scheme for any type of access, including reads. This, in turn, forces KSAM to empty the buffers and perform file label update, which means even more I/O's.

In circumstances where the application knows that writes constitute a small percentage of total I/O, and disc space is a scarce resource, an alternative that can make use of these known conditions would be very attractive. The project that brought about the Sorted File Access Method (SFAM) was motivated by such circumstances.

Briefly, the design criteria for SFAM were as follows:

1. Quick read access to exsiting data.
2. Low percentage of updates. (Less than 10 per cent.)
3. Small and fixed-size index.
4. One primary key with possible duplicate key values.
5. Generic search on partial key value.
6. Multiple record types in one data file.

The purpose of the last criterion is to save memory space for data buffers.

II. Design.

The SFAM package provides users a facility to create/build SFAM files, and a set of run-time procedures to access SFAM files.

The Concept of An SFAM File.

An SFAM file consists of an index, a primary area, and a overflow area. Records loaded at build time (detailed below) are placed in primary area in order of ascending key values. Records added to the file thereafter are placed in the overflow area and chained together in order of ascending key values. Every linked list in the overflow area is pointed to by exactly one primary record. Since writes after build are assumed to be infrequent, a link is a word pointer. This allows a maximum of 1024 overflow blocks (the leftmost 10 bits of the link word) and a maximum of 60 records per block (the rightmost 6 bits of the link word). A facility is available to reorganize an SFAM file whereby the records in overflow area are merged into primary area.

Each index entry consists of a part or the whole value of the low key in a data block and a double-word record pointer. No two entries have identical key values.

Overflow area, as well as primary area, are blocked. Blocks in overflow area are created on demand. Thus, any overflow block may contain more than one chains and a chain may span several blocks. In the worst case, a chain of n records may span n blocks, which increases the number of I/O in walking through the list. To improve the performance in accessing overflow area, the length of every chain in every overflow block is monitored. Before a chain expands to a new block, the chain length is checked. If the length is below a preset value, the entire chain in the block is moved to a new block. The preset value is called Minimal Capsule. This device keeps a chain segregated and disc I/O more efficient. The size of the minimal capsule is configurable, and should be between 1 and the blocking factor of the file. Default is one fifth of the blocking factor. The space freed by a chain is reusable only when other chains in that block grow. So there is a trade off between space and time.

Actual disc I/O is done in units of block, using SFAM internal buffers. The minimum (default) number of data buffers is 2, and maximum is 4. Number of buffers is specified through the use of JCW.

Creating And Building An SFAM file.

Since each SFAM file can have its own unique characteristics, such as key length, index key length, blocking factor, and min. capsule size, a 'spec' file is built first to hold the file information. This is done through the 'create' facility. Once a spec file exists, it may be used repeatedly by the build facility to actually build the SFAM file.

In term of MPE terminology, the build facility is a combination of build and load. Data are drawn from up to 10 flat MPE files sorted in ascending key sequence.

Run-time Procedures.

The run-time procedures are SFOPEN, SFCLOSE, SFSTART, SFREAD, SFREAD- BYKEY, SFLOCK, SFUNLOCK, SFWRITE, SFUPDATE, SFDELETE, SFERROR, and SFINFO.

SFINFO is used to provide usage statistics and capacity information.

To ease the conversion from KSAM to SFAM, call formats are identical to those of COBOL KSAM calls except for functions not available in KSAM.

III. Implementation.

Memory Index.

It was decided that the index, being moderate in size, would be loaded into memory at once and shared by all applications accessing the file. To accomplish this, the index is built into a USL file together with a binary search routine. The index (a code segment, to be exact) is then added to an SL so that it can be dynamically loaded and shared.

The memory index has the following advantages:

1. It is shared.
2. No disc I/O is necessary to access the index.

The disadvantages are:

1. In single user environment, a memory index would take more memory than if it were read into memory a part at a time.
2. The size of the index is limited by the max. code segment size of 16384 words. Subtracting 315 words for the binary search routine, the net index size is 16069 words.

SFAM File Label.

This area contains global file characteristics specific to SFAM, and one OFCB (overflow control block) per record type. Each OFCB contains information specific to a record type, pointer to the current over- flow block, and usage statistics such as number of writes, deletes, and maximum chain length, etc.

Extra Data Segment.

At SFOPEN time, an extra data segment is created. The layout of the XDS is as follows:

1. SFAM call parameters
2. control info (e.g. pointers, flags, EOF, etc.)
3. work area
4. usage statistics
5. message area
6. buffer pool control area
7. SFAM label
8. space for 2 records
9. buffer pool.

The extra data segment is freed by SFCLOSE.

IV. Miscellaneous.

As stated in section I, KSAM applications must lock the file for read in a multi-user environment. This is because insertions and deletions of data records also modify the B-tree. Thus, the key block obtained by the read process may be invalid due to this change.

SFAM employs a fixed read-only index, which is not affected by update operations. Thus read without locking the file is generally valid, except that the current record being inserted or deleted may not be 'seen' by the read process.

To avoid the above situation, SFLOCK accepts a parameter which indicates the purpose of the lock request: for read or for update. When SFUNLOCK is called, it would not update the file label if it was a lock-for-read request. This feature ensures that no other cooperating processes may update the file, but at the same time, no unnecessary disc I/O (for updating the label) is performed.

The size limitation on the index may be lessened by using a shorter key length for the index than that of the actual full key. As long as the partial keys uniquely identify every primary block, there would be no degradation in performance. Using an index key length of 4 bytes, for instance, the maximum number of index entries is 4014. This means at least 4014 primary blocks, or 240,840 records using the maximum blocking factor of 60.

V. Installation of SFAM at Longs.

We will conclude the paper with a brief review of how SFAM has been used at Longs for the last two years.

At the time of this writing, Longs is running a sizable pharmacy prescription system in more than 90 stores. The computer systems used are HP 3000 model 40, 39, and 37, all with one mega-bytes of main memory. Disc drives used are 7912's (66 MB) and 7914's (132MB). The software itself consists of a main program (over 860 KB) and many support utilities. The size of the database ranges from 26.4 MB to 57.6 MB. In addition to the database, it was necessary to maintain a KSAM file for generic search and other

quick info for on-line real-time uses. The average size of the KSAM data file was about 5000 sectors. The size of the key file was over 11200 sectors, due to duplicate keys.

After conversion to SFAM, the average data file size is 8000 sectors, and the index size is about 100 sectors. So the disc space saving is estimated at 50 percent.

The average disc I/O to access a KSAM file with 3-level B-tree is 4. The number of disc I/O's to access a SFAM primary record is 1. Since the length of an overflow chain is directly proportional to the number of SFAM writes AND the key distribution of these writes, the actual length can vary significantly percentage-wise. Assuming an average chain length of 9, the number of disc I/O's to access an overflow record is less than 3 (one primary block and two overflow blocks). The reduction in disc I/O in accessing the file is estimated at 70 per cent, (again, assuming a 80-20 read/write distribution). If taking into consideration the savings in label updates (mentioned in Section IV), the percentage could be even higher.

There is a price for all this gain - memory usage is up. Eliminating KSAM also eliminates the buffer space for key blocks. However, the entire index is now in memory, which causes an average increase of 10 KW of memory consumption.

The conversion of the code was done by adding an interface in the group SL, which translates all KSAM calls to SFAM calls. Since the syntax of both types of calls are almost identical, the translation is simple and straight forward.

The SFAM run-time package contains privileged intrinsic references. Storing them in a group SL would require all user applications to be prep'ed with PM capability. Longs chose to store them in the system SL. This way the applications are not required to have PM, and SFAM is made available system-wide.

VI. Acknowledgement.

The SFAM project was conceived by Bill Gates, manager of the Information Systems department at Longs, during the course of the pharmacy system development. Without his support and motivating force, SFAM would never exist. The author also would like to express his appreciation for the help given to him by Stephen Porterfield, Rick Gerlach, Debra Jensen-Dykes, and Sid Simpson. Finally, he wants to thank the programming staff of the pharmacy group for the cooperation and feedback they gave during the first year of SFAM conversion.

VII. References.

1. Bayer and McCreight, "Organization And Maintenance of Large Ordered Indexes", Acta Informatica, Springer Verlag, 1972, pp 173-189.

3035. YOU SAID YOU HAVE A BUNCH OF MICRO'S LINKED TO YOUR HP3000
GREAT!! NOW WHAT??

Eric S. Fisher
Vice President, Director of Systems Development
Wellington Management Company
Boston, Massachusetts, USA

ABSTRACT:

If you have already embarked on the great adventure of linking microcomputers to your H-P 3000, you have already encountered this problem. If this adventure is still in your future, beware!! Once you succeed in establishing the physical link, you have solved ten percent of the problem: the EASY ten percent! The major problem remaining is the content of the data you wish to share. Where does it come from? Where is it going to? What format is required? What security problems are created? In this paper, I will deal with some of these problems as they have been encountered, and solved, at Wellington Management Company, a Boston financial management company with an H-P 3000 and 60 IBM PC's.

INTRODUCTION:

This paper addresses information management issues. Throughout my eighteen-year career in computer programming and data systems development, I have found a consistent bias toward the technological side of our profession. I must confess that I share this bias; I react to the latest and greatest equipment like a kid in a candy store. I just love to work with state-of-the-art equipment, designing innovative solutions to previously insoluble problems. Who doesn't? This type of activity tends to be the source of our inner satisfaction and is often the way we achieve fame and fortune among our peers.

However, at some point in our careers we run into some hard, cruel facts: Our users are not our peers. Our bosses are not our peers. Our users judge us not on our technological elegance but on the ease with which they can use our creations. Our bosses judge us not on our technological elegance but on the relevance of our work to the ongoing functioning and profitability of our company.

Nowhere is this dichotomy between our interests and those of our bosses and users more apparent than in the area of microcomputers and networking. It is here that many of our users first encountered programs that were truly easy to use. It is here that many of our bosses first began to use the fruits of our labors directly instead of indirectly, and began to question the adequacy and cost of our large system designs.

Paradoxically, it is this same area of microcomputers and networking that can provide us with unprecedented opportunities

to redeem ourselves in the eyes of those same users and bosses. With some consideration in advance, we can provide them with useful, easy-to-use means of obtaining data from our large systems as well as feeding data to those systems. We can improve the utility and responsiveness of our large systems at the same time by removing some of the burden of supporting ad-hoc reporting, word processing, graphics and financial modelling from them.

The key phrase in the above paragraph is "some consideration in advance." In this paper, I will address ways we can consider in advance how our large and small systems will function together.

STATEMENT OF THE PROBLEM

In order to create a viable multi-system environment, the following problem areas must be successfully addressed:

1. A data environment must be created on the host system that fosters convenient access to system data;
2. A communications environment must be created between the host and microcomputer systems that allows convenient interchange of data;
3. An applications environment must be created on the microcomputer systems that allows host data to be conveniently used;
4. A security environment must be created on the network that protects the interests of the company without imposing undue hardship on the network users.

I. THE HOST DATA ENVIRONMENT

All data structures in the host environment are divided into two parts: those which were designed before PC networks became available, and those which were designed afterward.

Host data environments can be hostile to micro access for a number of reasons. Sometimes they are actively designed to be that way. If this is the case, it is usually because of some security considerations. I will discuss this issue further in the section of the paper on security.

In most cases, however, data access is difficult because the host environment was designed before micro networking became a consideration. Such issues as key relationships, file organization, data placement and data representation take on new meanings when placed in the context of micro access. Contributing to the general difficulty is the fact that the advent of micro access is also the advent of a new class of user whose classic request sounds something like, "I want to be able

to load the contents of this sales analysis report into my 1-2-3 worksheet."

Little does this user know, or care, that the sales analysis report is produced by three COBOL programs with two intermediate SUPRTOOL runs, takes three hours to execute, and would take five weeks to modify. Even if the modifications are made, the next thing we hear is, "This is great! Now could you move this column to the end, and add a percent of total on this other column right next to it?"

I could go on and on. Obviously, the presence of micro users adds to an already complicated situation and can create a potentially disastrous increase in programming backlog. Although it is not the intent of this paper to explore fourth-generation languages and data extraction tools, I nevertheless believe that the only efficient way to support micro-based data access is by means of such tools. We need to encourage the micro user to perform his or her own data extraction with as little help from the professional DP staff as possible.

While fourth-generation tools will obviously make data extraction and micro interfacing easier, they must be used in combination with the data access and structure issues indicated above. These are key relationships, file and data base organization, data placement and data representation.

KEY RELATIONSHIPS

Keyed file access usually reduces the time required to obtain the desired subset of records from a data structure. Multiply-keyed files, on the other hand, usually take longer to update. These considerations are applicable in general to data structure design when making trade-offs between on-line and batch optimization. Micro access adds another dimension to the on-line side of the equation. Where programmatic on-line access can sometimes make do with some combination of keyed access, internal selection, and an on-line internal sort, it may be necessary to add keys and possibly sort items to a file in order to make direct access by micro users easier and more intuitive.

FILE AND DATA BASE ORGANIZATION

IMAGE is the data base manager of choice for most on-line transaction processing applications. It is optimized for rapid retrieval of specific records, and safely shares file access among multiple users. Its limitations do not interfere with most types of programs. When access scenarios are being designed for micro users, however, some of KSAM's capabilities become very nice to have. All keys are inherently sorted. Generic key access is supported. The biggest problem with KSAM file organization is the fact that while retrieval can begin at the starting point of a particular key value, it does not

automatically stop when the key value changes; rather, it only stops at the end of file. Many fourth-generation tools solve this problem internally, allowing users to define an ad-hoc "chain" during KSAM file access.

If the application environment is sufficiently complex to require multiple IMAGE data bases and/or a mixture of IMAGE and KSAM files, a fourth-generation tool with an on-line dictionary becomes a virtual requirement. With a good dictionary, logical data bases understandable by the micro user can be defined with less regard for actual file organization.

One caveat about fourth-generation techniques: Sometimes a key must be added to a file that will uniquely define a single record in order for a fourth-generation tool to work efficiently. QUIZ, for example, has no construct to allow linkages to be created for exactly one record in a chain.

DATA PLACEMENT

Data placement issues cover both the location of the files and/or data bases and the location of the data within the files and/or data sets. The three-level directory structure (account/group/file) of the HP 3000 coupled with the home group designation of the logon user provide the system designer with flexibility to simplify or complicate data access at will. I have found it rather easy to communicate the file/group/account concept by talking in terms of first, middle and last names. Users tend to have little trouble with the concept of attaching a "middle name" to a file. The only requirement is that the groups be sensibly named. With multiple groups, careful use of access capabilities and a home group logon, access security can be easily implemented. If desired, files and data bases that are part of the permanent data environment (as opposed to working files created by the user for his or her own use) can be identified by file equations placed in a logon UDC.

More important than the placement of files and data bases is the placement of the data within the files. Once again, we run into the classic conflict between on-line and batch, between ease of use and machine efficiency. When access to data was entirely under program control, we could optimize our data bases considering only machine performance, disc space and response time issues. With the advent of micro access and ad-hoc data extraction, we must add consideration for user understanding. We need to reduce complex linkages and place more of the data in single files. There is a more urgent requirement for physical grouping of data to resemble its logical grouping. Dictionaries can cover some of the discrepancies between logical and physical grouping, but they and the tools they drive are much easier to manage and more efficient in their operation if they don't have to work too hard to cover a fragmented design.

DATA REPRESENTATION

The way we see numeric data is almost never the way it is stored internally. With modern binary file transfer techniques (discussed in the following section), it is easy to move data from one machine to another regardless of its internal representation. The one remaining problem is the greatest one, and will not be solved: one machine's internal numeric representation scheme is often radically different from another's. Thus, where machines differ in internal representation, numbers will always have to be converted into a common external representation before they can be usefully exchanged.

Fortunately there are commonly accepted standards for external data representation. Unfortunately, there are two standards instead of one in the United States. The first, Extended Binary Coded Decimal Interchange Code (EBCDIC), is used with IBM equipment except for the IBM PC. The other, American Standard Code for Information Interchange (ASCII), is used everywhere else, and is sometimes even available as an option in IBM environments. As we are mostly in non-IBM environments, we can safely assume that we are all going to be using the ASCII standard.

In the ASCII standard, all information is converted into a text-like representation. Thus, any report writer capable of redirecting output from a printer to a disc file can be used to prepare data for downloading. The only other consideration for the data preparation stage is the format in which the micro application expects to receive the data. Many micro applications, including in-house programs written in BASIC, will accept data formatted with quotation marks around text string data and commas between fields. This is an easy format to specify with a report writer. The only other thing to watch out for is if your dictionary has default numeric pictures that include commas, these default pictures will have to be overridden in the report body; otherwise your sales figure of 1,555,448.60 will become three fields: 1, 555, and 448.60.

OTHER CONSIDERATIONS

At Wellington, the size of our application required that the actual transaction processing portion of the system be "lean and mean" -- optimized for batch and on-line programmatic processing. To provide our end users better access to the data they are interested in, we have created a whole separate data environment designed for on-line inquiry and user reporting. This environment consists of data bases and KSAM files organized to provide user data with a minimum of complexity. There is a nightly batch stream that takes data from the transaction processing environment and moves it to the reporting environment. At the same time, many complex calculations are performed so that our users can retrieve pre-computed statistics. These "added

value" functions are greatly appreciated. At the same time they reduce the amount of CPU resources consumed by calculation routines in the daytime, allowing them to be concentrated on information retrieval and graphics.

Our data administrator has also created a very useful utility program for POWERHOUSE subfiles. It reads the "minidictionary" stored in the file labels of the subfile and, using the information contained therein, directly translates the subfile into a LOTUS download file. At the time this paper was written, we intended to contribute this program on the Washington swap tape. If it is not there, feel free to send a tape and a letter and we will place a copy of this program on it.

II. THE HOST/MICRO COMMUNICATIONS ENVIRONMENT

This extremely complex topic can be discussed extensively. Baud rates, error correction, Xon/Xoff, Enq/Ack, async/bisync/SDLC, all of these topics potentially merit a paper on their own. For this reason, I will give what I consider to be the proper solution to the problem without going into the technical details: Buy a package. Buy a terminal emulator package for your micro's that includes protocol-controlled error-correcting file transfer (a mouthful, but always mentioned on the cover of the software package if present). In the IBM PC/HP 150/8088 compatible world, your choices include PC2622 by Walker Richer & Quinn and HP AdvanceLink, both very competent packages. With other micro systems, your choices are much more limited, to the extent that I would suggest that if you are serious about micro-to-HP networking you should settle on one of the above PC's.

With a good emulator package, the only other considerations revolve around the nature of the physical communications link. For direct connection, a straight-through three-wire cable connecting pins 2, 3, and 7 of your PC's RS 232 asynchronous port to the same pins of the HP 3000 RS 232 port is all that is necessary. Modem connections require vendor-supplied modem cables between the various ports and modems. I would suggest that if you are intending to do a lot of interactive work or file transfer across the long distance network, you should consider the new 2400 baud dialup modems with internal error correction. I live within forty miles of Boston, but even across those distances the error rate at 1200 baud can be so frustrating that I sometimes log off and redial in an attempt to improve my chances at a good line. The modems should conform to the new CCITT V.22 bis standard. If you also choose error correction, it is too soon to tell what the final standard(s) will be, but it currently seems that the MNP system is a good bet.

III. THE MICROCOMPUTER APPLICATIONS ENVIRONMENT

The most frustrating and dangerous thing about the micro side of this network is that you as the HP 3000 manager are not going to have much control over it. Unless you seize the initiative,

assigning staff and resources to the micro support task (see my paper on the subject in the Amsterdam proceedings), the chances are that you will be presented with a micro and an application as a fait accompli by someone you may never have met.

Application development and package selection can be greatly simplified if a standard data interchange format can be agreed upon. Notice that I spelled out those words and did not mean to imply DIF. I can think of no less easy format than DIF, unless perhaps one wished to attempt to transfer numeric data in internal format and translate it by bit twiddling on the recipient machine. Truly, DIF is a hacker's dream and a user's nightmare.

At Wellington, we have standardized on two formats. Both of them are BASIC compatible; thus we might call them BASIC Interchange Format 1 and 2 (BIF1/BIF2) or BASIC Input Format and BASIC Output Format (BIF and BOF). I prefer BIF and BOF because they imply the direction in which the data moves.

BASIC Input Format is directly readable by BASIC programs. It is also known as LOTUS Import Format, because the File Import Numbers function of LOTUS 1-2-3 properly places data from such a file into a worksheet. This is the familiar method of enclosing text strings in quotes, separating fields by commas, and letting each record of the file represent one row of data. LOTUS and BASIC are not the only application environments that easily accept this format. Many other micro-based applications will either accept it for input, produce it as output, or both. Additionally, HP 3000 BASIC accepts it as input, as does DSG/3000 graphics. Our utility program to convert POWERHOUSE subfiles into this format enables us not only to extract data for micro downloading applications, but also to provide a simple interface into our graphics facility.

BASIC Output Format, similarly, is directly writable by BASIC programs. This is the format produced by the LOTUS Print to File function. While BIF is directly readable by database, spreadsheet, and graphics types of applications, BOF is more amenable to word processing and other more text-oriented systems. In some cases, like LOTUS, it is the only format available for application output that can be used by other applications. It is ironic that LOTUS is unable to import files that it has output in this format. In a way, BOF is just a fancy term for an ordinary printed report that has been redirected from a hardcopy device to a disc file or communications line. Sometimes, as in BASIC, this redirection capability is built into the language. In other cases, it is a part of the operating system, as in MPE. Applications or languages that have substring capability and ASCII to internal numeric format conversion facilities can access data from a BOF file and use it for updating or extracting information. At Wellington, we use such files to allow users familiar with LOTUS to prepare data in bulk for entry into our data bases. It is much more efficient for them to record an

review this data at their convenience and in a format that they can understand. It also conserves data processing resources because we do not have to design special-purpose applications or screens to allow them to enter data. Once the files they have entered are uploaded, we can create a special one-shot scenario to load the data. We also drive certain data base retrieval functions from files in this format.

One application that is in widespread use involves the entry of stock symbols into a LOTUS worksheet along with other information about the particular issues. Using the Print to File function, the users extract a list of symbols which is then uploaded to the HP 3000. There, a canned program uses this list of symbols to extract information from our user data bases about the specific issues indicated, in the order desired. This information is formatted into a BIF file which is then downloaded back to the micro and read back into the worksheet via the File Import Numbers function. Although this sounds a bit complicated, it is really very simple to learn, and the elapsed time after a little bit of practice is less than five minutes for the entire operation. Our users love it because it does precisely what they wanted it to do: it allows them to update their worksheets from the contents of our data bases without rekeying the information by hand.

IV. SECURITY CONSIDERATIONS

Many articles have been written about data and system security in the HP environment in particular as well as the information processing industry in general. The advent of microcomputer networking has exacerbated the security problem in several ways. The dichotomy between making data access easy for unsophisticated users while protecting the security and privacy of that same data has become more acute. The opportunity for unauthorized access to programs and data has increased. Concerns for data integrity and critical file backup can now extend to hundreds of locations beyond the central site. Each of these subjects is potentially worthy of a paper in itself. In many cases, such papers have already been written. Therefore, this paper contains only a brief overview of each area.

EASE OF ACCESS VS. SECURITY AND PRIVACY

I have spent the majority of the previous space describing ways to make it easier for your users to access central data. Obviously, making it easy for your users potentially also makes it easy for others as well. If your computer has dial access, others can include hackers and possibly industrial spies, if you keep proprietary information on your computer net that could benefit competitors. The degree of security you can achieve is directly proportional to the amount of money you wish to spend. Perhaps the most effective economical solution to dialup access security is provided by the new dialback modems. Formerly only possible with expensive equipment, it is now easily affordable to

install a modem that first requires a password to complete the computer connection, then requires the user to hang up the telephone while the modem calls back. This system requires the user not only to know the password, but also to be at an authorized telephone number.

More and more people are connecting their computers into public packet networks like Telenet, Tymnet and CompuServ. If security is an important consideration, and it is still desirable to connect into these networks, password schemes can be implemented with the cooperation of these networks to restrict access at the connection level even before beginning the logon process.

Security can be a problem even if the network is completely hard-wired. Often there is company data that is confidential even to most employees. In this case, the security provisions must be aimed at preventing employees from gaining access to unauthorized accounts and/or data bases. Depending again on the amount of money you wish to spend, solutions can range from the traditional changing passwords often, to data encryption, to the placement of sensitive data on a separate machine connected only to authorized terminals. With the advent of the Series 37, the last option has become much more feasible for smaller shops.

The choice of security protection methods depends on a cost/benefit analysis. Data needs to be classified according to its sensitivity and the cost to the company needs to be computed of the value of the data and the value lost by unauthorized access. Then the cost of the protection can be compared to the value of the protected data. Risks addressed by the various protection schemes can also be assessed. Finally, an informed decision can be made similar to the ones made regarding insurance coverage.

ACCESS OPPORTUNITIES

The stories that are told, including the movie "War Games", reveal the increase in opportunity for unauthorized access provided by the microcomputer. Microcomputers can be programmed to dial successive telephone numbers, keeping a tally of those that respond with a modem tone. They can be programmed to keep dialing a number and trying different passwords until one succeeds. Computers identify themselves as running under a particular operating system and being of a certain type as they request logon. How many computers, for instance, respond to a carriage return with a colon (:) prompt? For the experienced hacker, this is a sure clue to an HP 3000 possibility. With over twenty thousand HP 3000 computers in the field, all running MPE, how much of a secret do you think it really is that once you have an HP 3000 on the line, all you have to do is find the password to `MANAGER.SYS`?

All of the above refers only to the casual hacker trying to get into your system just for kicks. Far more severe dangers are

present within your own organization. File transfer is an unprecedented opportunity to steal data and even programs. Even the HP Response Centers use the binary file transfer capability of AdvanceLink to download software patches and to take copies of your HP 3000 programs (with your permission of course) in their attempts to respond more quickly to your problems and to duplicate them on their own HP 3000's. It does not matter in the least that the data and/or programs transferred from the HP 3000 to the IBM PC or the HP 150 are unusable on those machines. In this case, the micro is simply an intermediary, a repository, a medium for the exchange of information from one HP 3000 to another. The important thing is that HP 3000 programs and data can be recorded on IBM PC or HP 150 diskettes.

DATA INTEGRITY AND CRITICAL FILE BACKUP

The data integrity issue expands with network file access. With the possibility of creating data files on micros that will be used to update data on the mainframe comes the increased opportunity for bad data to get into the mainframe data bases. At the same time, the use of downloaded data in micro applications raises the question of the timeliness of the data used, because the timing of the download is usually under the control of the micro users. It is easy to imagine the chagrin of a product manager when it becomes obvious that the figures used as the basis for a whole series of LOTUS worksheet-based projections and scenarios are two weeks out of date because the new download was forgotten.

At the same time, micro users are notoriously lax in their file backup discipline. If critical data or applications are used at the micro level, some method should be used to make sure that backups exist. Responsibility for this operation should be in the hands of the data processing department, not with the individual users. One scenario is to allocate a certain amount of HP 3000 file space for micro archives. As it does not matter whether the micro can use data recorded from the mainframe in the context of data and software piracy, so it also does not matter whether the mainframe can use the data and programs uploaded from the micros. If this is done, the micro files will be backed up along with the mainframe data.

V. CONCLUSIONS

The purpose of this paper is not to discourage mainframe/micro networking. I believe that the benefits of this far outweigh the dangers and costs. At Wellington Management, we actively support the concept and are always seeking ways to improve our user support and expand our applications base.

The power of the microcomputer is only beginning to be used. I think that by welcoming the networking and intelligently distributing the processing resources, we will reap great rewards in improved office productivity. The benefits are many, and the

risks relatively small. Due consideration of the risks involved can minimize them and greatly accentuate the benefits.

3037. The Effectiveness and Shortcomings
of
Using Programming Tools

Tad L Olson
Marketing Engineer
Hewlett-Packard
Computer Systems Division

A. The problem.

The software business enjoys all the standard challenges of other businesses: the high cost of producing the product, the rising need to be more productive, the long turn-around time from idea to release. While major changes are taking place in the technology of building computers, software development has not come close to keeping pace with advances in hardware. Improvements in the software development process have been slow in coming.

Because hardware has advanced so rapidly in recent years the amount of new software needed is increasing. With this improvement in hardware, applications that were not feasible in the past are now within reach. The end result has been a steady increase in the backlog of incomplete applications and an increase in dissatisfaction among users because of the inability of EDP organizations to do its job.

As machine costs continue to drop and the number of machines increase, the demand for new applications will greatly exceed the capacity of the DP industry to create these applications if conventional development is not replaced with more productive methods.

B. The solution.

There still seems to be no magical or revolutionary solutions on the horizon.

What is required to achieve gains in software productivity is an integrated long range program and a sustained effort. A piecemeal and sporadic attack on problems may only compound the problem.

1. The Role of Management

The prospects for improving productivity begins with improving management practices. Managers must have a clear view of the project from inception through test and implementation. Having a good schedule and budget control, use of good development methodology, the hiring of good people and keeping it simple is part of the answer. But another answer is the use of programming productivity aids or software tools.

Top DP management often is not familiar with the remarkable productivity gains which can be achieved with recent productivity tools. Although many managers have attempted to stay current in the application of new technology, the actual use of new techniques has resulted in only a marginal improvement in productivity. The end result has been lengthening backlog and rising end-user frustration.

From top management's perspective, however, there is an enormous economic incentive to use these new techniques in order to reduce the cost and increase the quality of application development.

In his book, *Software Engineering Economics* (1), Barry Boehm argues that most companies could achieve productivity gains of 2 to 1 in a period of 3 to 4 years and up to 5 to 1 in a time span of 6 to 8 years. Gains could be even greater in some cases, ranging up to threefold in four years and as much as 8 times in 9 years.

With increase pressure from the end-user for more applications and corporate pressure to increase productivity the DP executive is caught in a very difficult position.

2. The Role of Tools

There is a whole spectrum of tools designed to improve programmer productivity, many of which use mutually incompatible approaches. The most sophisticated and most complex sort of tools are applications development systems or program generators. These tools, which usually contain fourth-generation languages, attempt to eliminate programming as a handcraft by automatically creating code in response to non-procedural queries (in the case of fourth-generation languages) or by following specifications set down by systems analysts (in the case of program generators).

At the other end of the spectrum are tools that speed up the programming process. They include utilities like test data generators and flexible screen editors that fit into the work routines of programmers. The aim of these tools is to make programming more efficient rather than to revolutionize it. In between are the products ranging from code generators that only produce part of an application and systems that support libraries of reusable software modules or code, to products that help to streamline, structure, document, or test programs produced by traditional methods.

These tools need to be introduced one step at a time in an evolutionary manner rather than as a great leap forward. Experienced programmers, who are the skilled craftspersons of this profession and work primarily on a custom basis, may be reluctant to employ them. "Automatically creating code" rings harshly against the delicate balance of skill and experience programmers need to produce quality products.

3. Tools For the Systems Design Stage

This is the stage in the program development cycle where most errors can be avoided. Unfortunately, this is also the area which has the fewest tools.

o System Design Methodologies (SDM)

There is a class of tools generally referred to as "System Design Methodologies" which attempt to automate the design process. "Data Dictionaries" are useful in this stage because they assist in describing the data elements involved in an application and in establishing the interrelationships among various data elements. Another class of packages, "Documentation Aids," provides help in producing hard-copy documentation of design data.

o Fourth Generation Languages

Analysts can also take advantage of powerful new automated fourth-generation languages. Using these new languages the analyst can learn how to build complete applications faster than they could write the specifications by hand. In addition, the analyst can learn how to act as a consultant to an end user and how to become highly effective in an Information Center environment.

o Prototyping Techniques

The use of prototyping techniques can be used in the system design stage. Prototyping is common in engineering systems, where several concepts for a system are often tested before committing to a final design. Complex data-processing also requires prototyping in order to resolve design questions and to ensure that the final system fully meets end-user requirements. If the analyst can demonstrate a prototype to the end user, much of the misunderstanding and miscommunication between the analyst and the end user is eliminated.

o Documentation Aids

Documentation aids provide automated production of documentation such as flow charts, formatted program listings, Data Dictionaries and JCL. Systems people consider documentation one of the most distasteful tasks, yet necessary for future maintenance activities. These aids generally help to reduce the amount of programmer time needed to produce documentation and circumvent the reluctance to document.

4. Tools for the Program Development Stage

Application development is the area which has the richest set of tools. The tools generally fall into two broad categories; those which simplify the programming process using conventional languages, and those which substitute for conventional languages.

The primary objective of these products is to make writing programs easier and faster. These tools do this by providing methods that make programming more efficient.

o Programming Environments

This classification of software tools comprises those systems which provide an integrated set of tools, or an environment for the programmer. Typically, the environment includes on-line, conversational capabilities which permit the programmer to create new programs, correct and manipulate existing programs, submit programs for compilation and execution, review test outputs, specify hard copy printouts. Usually, a command language capability is provided to simplify the programmer's interface to the system, and use of a library may also be included as part of normal on-line access. Programming environments greatly streamline the activities of the programmer, eliminating much routine clerical work and reducing the chance of error.

Key Features To Look For:

- o Product should be an integrated, interactive set of tools
- o Increases Programmer productivity
- o Contains: Environment file manager
Full-Screen Editor
Program Translation Feature
Source level or "Symbolic" Debugger
On-Line Help
Source code generator

- o File Managers

One of the ways in which program development can be made more productive is to use tools that keeps track of files used in the development of a program. There is a collection of files that go into the development of a single program, including the source files, program file, USL file, and \$Include files. These tools should remove the "Bookkeeping" tasks from the programmer.

What to Look for in a File Manager:

Several versions of the same source file should be able to be kept in one program development.environment Each source file in the environment may consists of one or more versions which may differ from each other. A version is a logical copy or "snapshot" of the file. As changes are made to the file in the form of additions, deletions, updates etc. they should be kept in versions of the same file. The user, through simple commands should be able to control when the "snapshot" of the file is taken to create the logical copy or version.

- o Source Level or "SYMBOLIC" Debug

Overview

The Symbolic Debug feature allows the user to interactively debug a program without any knowledge of memory locations or code address. You can inspect and modify data structures on-line using names defined in your source program. This allows even the novice programmer to quickly produce error-free application programs without having to work at the primitive level of memory locations.

What to Look For in a Source Level Debugger:

Symbolic Debug should allow the programmer to access the following functions when monitoring the execution of your program.

- o Set or clear breakpoints symbolically by (qualified) paragraph names, section names, level 1 procedure names, qualified level 2 procedure names, or compiler generated listing line numbers.
- o Allow qualification of symbolic locations by main or sub program name.
- o Allow the user to specify a list of commands which will be automatically executed at a specified breakpoint.
- o Display paragraph or procedure names as they are executed
- o Displays the last n paragraphs of procedures executed
- o Monitor data items
- o Display or change variable values

o Source code generation

The most sophisticated and complex sort of tools are applications development systems or program generators. These tools, which usually contain fourth-generation languages, attempt to eliminate programming as a handicraft by automatically creating code in response to non-procedural queries (in the case of fourth-generation languages) or by following specifications set down by systems analysts (in the case of program generators).

According to the MARTIN REPORT (2), the largest gains in application-development productivity have been achieved with automated fourth-generation language tools such as data-base query languages, report generators, graphics languages, decision support systems, and application generators. Fourth-generation-language tools are capable of 10 to 1 relative to hand coding using FORTRAN, COBOL, or Ada. These tools include both procedural and non-procedural languages, languages for DP professionals and languages for end users.

5. The Results

There are numerous ways in which productivity can be increased in the traditional development environment. Admittedly, tools by themselves are only a partial answer to the kinds of productivity gains needed over the long term, but they should receive serious attention by management.

"We'er going to start using software tools - when we get the time." It's the old story; we've got so much on board that that we can't sit back and think how to solve the problem. Other concerns are that people fear that they will become guinea-pigs for products that are not yet fully developed.

There's an exponential trade-off between time taken to solve a software problem and cost. A 10% increase in the time may mean a 25% decrease in the cost.

The results of a recent Productivity Survey (3) show that the use of productivity tools, or software packages to aid in the development process. The results are shown in Figure 1. There is a poor showing of many classes of tools which are known to provide productivity advantages. The survey suggests that a large part of the problem is simply the unwillingness of programmers and analysts to adopt new methods, and the failure of management to provide the readership and support.

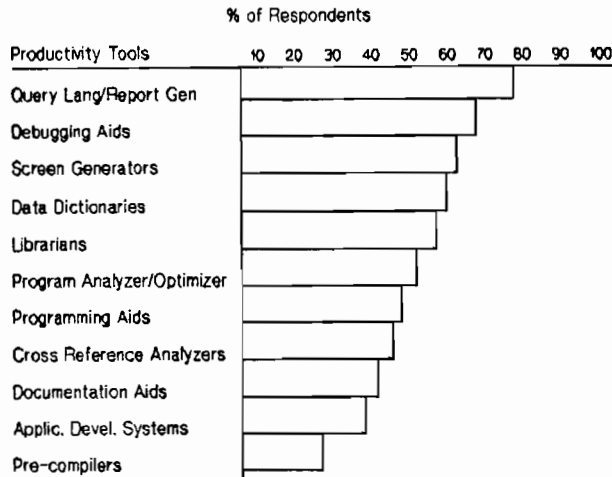


Figure 1. Use of Productivity Tools

6. Conclusions

Are users making an effort to improve productivity? Some are, but it couldn't really be called a major trend. This does not necessarily mean that productivity isn't improving. What it does mean is that tools are not being widely applied as they might be.

The required increase in productivity can be achieved using existing language tools and fourth-generation languages. If there were no increase in productivity of application development, we would have to increase the number of programmers from 300,000 to 12 million in the next ten years (Martin, 1984). Clearly this is impossible. The only other alternative is to achieve a major increase in average programmer productivity through the user of these tools.

How do you decide which tools or productivity aids to consider? Because there are so many classifications of tools, and because they apply to different stages of the program development cycle, it is usually best to take stock of where your organization stands right now and attack those areas which are the biggest problems. Analyse your own situation carefully and find out where your major weaknesses are. Then look to productivity aids as one means of overcoming those weaknesses.

References

1. Barry W. Boehm. Software Engineering Economics, 1981. 767 pp. Prentice-Hall
2. James Martin. The Martin Report on High-Productivity Languages 1984. Section 1.0/13. Technology Insight
3. "Productivity Survey," System Development, July 1984, pp. 1-4

3038. Emulating A Real-Time, Multi-Tasking Application System On The HP-3000

Jerry Fochtman
Systems Analyst
Exxon Chemical Americas

Introduction

Until recently, real-time processing could only be achieved by sacrificing versatility and/or user friendliness. With the HP-3000 computer system it is possible to emulate real-time processing without losing these or any of the other benefits available on the HP system.

The HP-3000 architecture offers several mechanisms and strategies that can be exploited to imitate a true, real-time processing computer. It is even possible to improve overall performance and data throughput of an application by utilizing the resource sharing features of the HP-3000. Large storage capacity, communication capability and tools for data analysis make the HP-3000 an excellent alternative to some types of real-time systems.

These characteristics have been demonstrated by successfully implementing this form of data processing on an HP-3000 in a real-time production environment. In addition, the particular system being presented achieved an 85% improvement over the existing process control computer.

Project Background

The project that will be described in this presentation was developed by the Production Department of Exxon Company U.S.A.. This department is responsible for the monitoring and collection of data associated with producing oil and gas wells. The existing system collects production data, alarms and status information from various metering devices using an IBM Series-1 computer (Figure 1). The data is then transmitted to an IBM-1800 process control computer system for further processing.

In addition to reading the various meters, the IBM Series-1 computer also receives user inquiries and manually entered data from terminals located throughout a division. These requests are also passed to the IBM-1800. After being processed on the IBM-1800, the results are transmitted back to the Series-1 to be routed to the appropriate terminal.

Once a month, the production information that has been collected and stored on the IBM-1800 is transmitted to the

headquarters computer center. There the information is used to produce a variety of reports for Exxon, the government and the lease holders of the producing properties.

During an average day, this system collects and processes approximately 11 million bytes of data. When the communication protocol and data synchronization information is added, the transmission volume increases to almost 40 million bytes a day.

This system has been in operation for since the late sixties. It is currently being used in four divisions within the Production Department.

The Problem - Hardware Obsolescence

IBM announced that hardware support for the 1800 computer system would be discontinued in September, 1985. Furthermore, IBM does not have a compatible replacement computer that can handle the 1,500+ programs and subroutines that constitute the existing system.

Therefore it became necessary for the staff in Exxon's Production Department to examine the properties of other computer systems for a suitable replacement. One of the major considerations was to locate a computer that could replace this system with a minimum amount of software conversion. The investigation culminated with the selection of an HP-3000 system to replace the existing IBM-1800s. However, this type of project had never before been attempted on an HP-3000. Ahead lay the difficulty of adapting the HP-3000 to this form of computing.

Project Definition

The next task was to identify the steps necessary to convert the existing system from the IBM-1800 to the HP-3000. This involved dividing the project into two distinct phases. The first phase involved the development of a system on the HP-3000 that would communicate with the IBM Series-1 system and also interface with the application programs. The second phase required the conversion of the existing application programs to execute on the HP-3000.

This presentation will concentrate on the first of these two phases, the development of the communication and application interface system. This phase also demonstrates a method of emulating real-time processing on an HP-3000.

Project Requirements

There were many stringent requirements that had to be met by this particular system to make the project a success. The following is a list of the more critical requirements:

- * Bi-Directional Communication With An IBM Series-1
- * Identify/Schedule Application Programs
- * Permit Application Programs To Schedule Each Other
- * Priority Scheduling
- * Terminate/Suspend Application Programs
- * Common Memory Storage Area Available To All Application Programs
- * Extensive Error Handling
- * Internal Messages Used To Notify Operating Staff
- * Provide Performance Information
- * Include I/O And Event Tracing For Problem Investigation
- * Control Access To Shared Data
- * Minimize Overhead
- * Event Driven
- * Multi-Threading Of Application Programs
- * Continuous Operation
- * Synchronization Of Computer Clocks

HP Special Capabilities And Features Used By The Interface System

Many of the special capabilities and features of the HP-3000 are utilized in this system. The majority of the features have been documented at one time or another by Hewlett-Packard. The following is a list of those that are used in the system:

- * Special Capabilities
 - Process Handling
 - Extra Data Segments
 - Multiple RINs
 - Privilege Mode
- * File Systems
 - Message Files
 - Circular Files
 - KSAM Files
 - Message Catalog Files
- * Split-Stack Operations
- * Communication Subsystem Procedures (CS)
- * CS Trace Facility
- * No-Wait I/O
- * DL-DB Stack Area
- * Extensive Error Traps
- * Segmented Libraries

Overview Of The New System

The software system that was developed to meet the project requirements was divided into five functional areas. Each area performs a necessary function of the system but is dependent upon the other areas (see Figure 2). In addition, this design would provide the highest level of performance and data throughput. The following are the five functional areas contained in the interface system:

- 1 - HP-3000/Series-1 Communication
- 2 - Data Identification And Program Scheduling
- 3 - Application Program Control
- 4 - Error Message System
- 5 - Application Program Interface Procedures

The programs and procedures were developed in SPL programming language in order to take full advantage of various HP features.

Area 1 - HP-3000/Series-1 Communication

The primary purpose of this area of the system is to communicate with the Series-1 computer. It also serves as the father process to the interface structure and as such, establishes several components that will be used by other aspects of the system. In addition, this portion of the interface system controls the synchronization of the clocks between the HP-3000 and the Series-1. The INFC program, three message files and three extra data segments comprise this area of the interface (see Figure 3).

Data Communication

The hardware link to the Series-1 computer utilizes an Intelligent Network Processor (INP) board. This is connected to a modem eliminator using a V.35 cable. In turn, the modem eliminator is connected to the Series-1 computer. It was necessary to have the modem eliminator to provide a high-speed clock signal to the Series-1 computer. This allowed the two computers to communicate at speeds up to 56,000 Bits/Second (56K baud). The logical device number of the INP board was configured as a standard Remote Job Entry line.

The bisynchronous communication is performed using the MPE internal Communication Subsystem (CS) procedures. These privileged procedures were documented in the third edition of HP's Communication Handbook. The options used to open and access the communication channel were similar to those used by the RJE Subsystem.

Program Initiation

Upon initiation, INFC acquires three extra data segments. One is identified as the Control Data Segment (CDS). It contains various information on the interface and application programs such as their condition and performance (see Figure 4). The other two data segments are used by the Data Identification and Program Scheduling portion of the interface. One is used to maintain a list of available and used extra data segments employed as data storage buffers. The other extra data segment contains the scheduling queue used to control the execution of application programs. (These structures will be outlined in more detail during the discussion on the Data Identification and Program Scheduling portion of the interface.) The Data Segment Table (DST) numbers that identify these three data segments are placed in three open words contained in entry 0 of the system's Process Control Block (PCB) table. This provides programs outside the interface system access to these structures.

In addition to establishing the various extra data segments, INFC also initiates the Error Message portion of the interface system. (This area of the interface is outlined in more detail later.) Once the error handling facility has started, INFC waits for a date/time buffer from the Series-1 computer.

Disc Files

Two MPE message files are used as a staging area for data being received from, or transmitted to, the Series-1 computer. A third message file serves as a method of passing various commands to the interface system via INFC.

The two input message files along with the communication channel are accessed in NO-WAIT I/O mode. This HP feature allows this portion of the system to be driven by an I/O event or interrupt. This permits INFC to wait indefinitely for activity from either of the three channels without impacting the performance of other interface areas.

Clock Synchronization

The IBM Series-1 computer contains a battery supported clock. The date and time indicated by this clock serves two purposes. First, the date and time is used to insure the computers are properly communicating. Every two minutes the Series-1 sends a buffer to the HP containing the date and time. If the transmission fails, the Series-1 notifies the staff that a problem exists with the HP. If the HP fails to receive the date/time buffer from the Series-1, the interface sends out a message indicating there may be a problem with the Series-1 computer.



The second use of the date and time is to schedule certain application programs for execution. It is crucial to the application that these programs execute at the proper time. Because the IBM Series-1 clock determines when application programs will execute, the Series-1 is used to set the clock on the HP-3000. This is accomplished by updating the appropriate storage words in MPE's Timer Request List table (see Figure 5).

The setting of the clock on the HP is only done when the first date/time buffer is received from the Series-1. Each subsequent date/time buffer is checked against the HP's clock to insure synchronization. Should the computer's clocks be out of phase more than 90 seconds, several flags are set to prevent execution of the critical programs. In addition, messages are sent to the operations staff indicating the clocks are out of phase. After evaluating the situation, it is possible to re-synchronize the clocks by sending a command to the INFC program via the command message file.

In addition to setting the HP's clock, the first date/time buffer is also used as an indication that the communication link is functioning. This causes INFC to initiate the Data Identification and Program Scheduling aspect of the interface system.

Area 2 - Data Identification And Program Scheduling

This portion of the interface system identifies the data received from the IBM Series-1 computer and schedules the appropriate application program to process the information. This operation is performed by the LOADBUF program and is configured to use up to 40 extra data segments for temporary data storage (see Figure 6).

Program Initialization

When LOADBUF is initiated by the INFC program, it starts the Application Control portion of the interface system (discussed later in more detail). Once that aspect of the system is functioning, LOADBUF commences processing the information placed in the input message file by INFC.

Processing Input Data

Upon initiation, LOADBUF posts a read to the input message file. When the INFC program receives data from the Series-1 and writes it to the input message file, LOADBUF obtains the buffer. The first buffer contains information to identify the application program responsible for processing the data and the amount of data in the transmission. From this information, LOADBUF

calculates the number of records to be read from the file to complete the input buffer.

Intermediate Data Storage

At this point LOADBUF examines the Buffer List structure for a free extra data segment to store the Series-1 data. The Buffer List was one of the data segments created by INFC. Its purpose is to maintain a list of extra data segments used for temporary storage of input buffers received from the Series-1.

The Buffer List contains two linked lists; one identifies available data segments, and the other identifies the data segments already in use (see Figure 7). The structure itself is accessed by a set of procedures which execute in split-stack mode. Through a series of Q relative pointers, the GETBUFFER procedure is able to quickly interrogate the list for available segments. Should all the existing data segments be in use, the procedures will acquire a new extra data segment and add it to the list. Once obtained, the entry (Figure 8) for the available data segment is transferred to the in-use list and the DST number of the data segment is returned to LOADBUF.

The total number of data segments that can be maintained in the structure is established during creation. Once this limit is reached and LOADBUF requests a buffer when none are available, the GETBUFFER procedure used to access the structure places the program in a JUNK wait state. (This is a process wait state reserved for MPE and can only be accessed by calling the privileged procedure WAIT.) The following is the code used for this operation:

```

LOCKLOCIN(e'dstlist'rin, l'true);
GetDSTBuffer(i'prog, i'dst, i'dstid, i'err);
IF i'err = e'list'empty THEN
  BEGIN
    GETPRIORITY(0, e'bs'priority);
    UNLOCKLOCIN(e'dstlist'rin);
    l'true := TRUE;
    LOCKLOCIN(e'dstwait'rin, l'true);
    WAIT(e'junk'wait, 0);
    GetDSTBuffer(i'prog, i'dst, i'dstid, i'err);
    UNLOCKLOCIN(e'dstwait'rin);
    GETPRIORITY(0, e'cs'priority);
  END
ELSE
  UNLOCKLOCIN(e'dstlist'rin);

```

The first step performed by this series of code is to lock the local RIN used to control access to the Buffer List. Once the RIN

is locked, the program attempts to obtain an empty buffer. If no buffers are available, the program's execution priority is elevated to the BS Linear queue. This insures the program has priority over any other interface program attempting to obtain a buffer. Next, it releases the local RIN used to control access to the Buffer List and locks a second RIN. This second RIN serves two purposes. First, should multiple programs be attempting to obtain a DST buffer, this will stack them in a First-In/First-Out manner. The second purpose of this RIN is to identify the program next in line for a buffer.

Once the program obtains the second RIN it is at the top of the list of programs waiting for buffers. The next instruction places the program in a JUNK wait state. The program will remain in this state until another program releases a buffer using the FREEBUFFER procedure. Once it is re-activated, the program obtains the available DST buffer. Next it releases the RIN used to stack waiting programs and returns to normal execution priority.

The FREEBUFFER procedure for is used for releasing a DST Buffer. When called, it places the DST Buffer on the list of available entries. It then checks for any programs waiting for buffers by examining the second RIN. If the RIN is owned by a program, that program is first in line for a buffer and is in a JUNK wait state. FREEBUFFER simply awakens the program from its JUNK wait so it can obtain the buffer just placed on the list:

```

LOCKLOCIN(e'dstlist'rin, 1'true);
FreeDSTBuffer(i'prog, i'dst, i'dstid, i'err);
i'pin := LOCINOWNER(e'dstwait'rin);
IF i'pin <> 0 THEN
    AWAKE(i'pin*e'pcb'entrysize, e'junk'wait);
UNLOCKLOCIN(e'dstlist'rin);

```

Transferring Data To The Buffer

Once LOADBUF has obtained a data segment to store the input data, it changes its DB register to point to the base of the data segment (split-stack mode) and initializes the area. After initialization, the first record read from the input message file is moved into the data segment. This is accomplished by using the firmware instruction MVLB while still executing in split-stack mode. This instruction moves data relative to the DL register to a location relative to the DB register. Because the DL register still points to LOADBUF's stack and the DB register is pointing to the extra data segment, this makes it possible to move data directly from the stack to the extra data segment while in split-stack mode.

The reason for selecting this method of transferring data from the stack to the data segment was for performance. LOADBUF

can now complete the loading of data into the data segment by performing the FREADs while in split-stack mode. The target array specified in the intrinsic call is a pointer contained in the segment identifying the next data location. This pointer is incremented after every read until the input record count, determined by the first record, is exhausted.

After completing the record count, LOADBUF returns to normal stack operation and attempts to read another record from the input message file. If a record exists and it's for the same program, LOADBUF determines if the current data segment has enough room for the next series of records. If the data segment has sufficient space, LOADBUF enters split-stack mode again and starts the process of placing the input in the data segment. If there is no more input, or if the data is for another application program, LOADBUF places an entry on the Scheduling Queue for the application program which is needed to process the information placed in the data segment.

Application Program Scheduling

Like the Buffer List, the Scheduling Queue was created by the INFC program when the interface system was initiated. The queuing structure is a linked-list of application programs to be executed along with the number of the data segment containing the information to be processed (see Figures 9). The list is maintained in a sequence based upon priority. If two programs are scheduled at the same priority, the second one scheduled is inserted in the list behind the first one.

The Scheduling Queue is accessed through another set of procedures that operate in split-stack mode. The structure also contains two linked-lists. One list identifies the programs to be executed. The other contains free queuing entries. Like the Buffer List procedures, the Scheduling Queue procedures will place a program in a JUNK wait state if there are no available entries in the list. But instead of using local RINs to control access to the list and identify waiting processes, global RINs are employed. This allows programs outside of the interface system to schedule application programs for execution. It was necessary to develop a procedure similar to LOCRINOWNER to identify the owner of a global RIN.

The global RINs used in the interface system are programmatically acquired by INFC and stored in the Control Data Segment for reference. When the interface is shutdown, INFC releases the global RINs. This technique eliminates the need to use specific RIN numbers in the code.

Once an entry has been placed in the Scheduling Queue, LOADBUF returns to the input message file to either process the next set of records or wait for more input from the Series-1 computer.

Area 3 - Application Program Control Area

The Application Program Control portion of the interface system is responsible for executing the scheduled application programs. This function is performed by the QUEUE program (see Figure 10).

Program Initialization

Once started, QUEUE examines the Scheduling Queue for any application programs awaiting execution. If the list is empty, the QUEUE program goes into a JUNK wait state.

Scheduling Queue Entry Exists

When the procedure PUTQUEUE places an entry in the Scheduling Queue it examines the status of the QUEUE program. If the QUEUE is in a JUNK wait, the PUTQUEUE procedure will activate it. Upon activation, QUEUE obtains the first entry in the Scheduling Queue by using the procedure GETQUEUE. This procedure will always return the highest priority entry that exists in the Scheduling Queue at that moment.

Locating The Application Program

Once QUEUE has been given a program to execute, it examines the Program Information Area (PIA) of the CDS for the specified program. Each application program has an entry in this area that contains a variety of information. For instance, if a program has not been executed since the system started, the corresponding area of the CDS will be empty. If an application program has been executed and has terminated, the PIA will have the program's name and scheduling information. Additionally, if the program was configured to suspend after execution, the Program Information Area will include the PIN number of the suspended program. The next action taken by QUEUE depends on the flags of for the selected program.

If the area was empty, QUEUE obtains the application program information from a KSAM file called PROGINFO. Every application program has an entry in this file. The information consists of the fully-qualified name of the program, a flag indicating whether the program is to suspend or terminate when finished processing the data and also an indication of what mode of execution is assigned to the program (this will be explained in more detail later). From this information QUEUE constructs and updates the program's information in the Control Data Segment.

If the CDS entry already exists and contains a PIN number, QUEUE examines the characteristics of that PIN to insure that it is the same program and is a SON process of QUEUE. If the program identified by the PIN is not the same, QUEUE generates a warning message and clears the PIN in the Program Information Area. If the PIN does belong to the desired program, QUEUE next determines the status of the program. If it is currently executing, QUEUE sets a flag in the CDS indicating that the program should suspend when finished and then QUEUE waits for the program to suspend. Once the program has completed execution, QUEUE begins setting it up to process the next block of data.

Before the application program can be executed, QUEUE places the DST number of the data segment containing the input for the program in the Control Data Segment. (Remember, this information was loaded into the data segment by LOADBUF and the DST number was placed in the Scheduling Queue along with the program ID code.) Now that everything is ready, QUEUE examines the execution mode assigned to the application program.

There are four types of application program execution modes available in the interface system (see Figure 11). The following is a description of the modes and how they relate to each other:

Execution Mode	Scheduling Criteria	Description
FOREVER	Program may run at any time.	Special purpose programs that run continuously such as scheduling programs (i.e. SLEEPER).
EXCLUSIVE	Can be the only executing program except any number of FOREVER mode may run at the same time.	User mainly for critical file modification programs.
SHARE	Program may run at any time as long as an EXCLUSIVE program is not executing.	Most user inquiry programs use this mode of execution.
SEMI-EXCLUSIVE	Must be the only program running except for any number of SHARE and FOREVER programs.	File updates with inquiry

Based on the mode of the program, QUEUE may have to wait for currently executing program(s) to finish before the waiting program can be initiated.

Empty Scheduling Queue

After processing all available entries from the Scheduling Queue, QUEUE makes a pass through all the application program entries contained in the Control Data Segment. This is done to insure the information in the CDS reflects the current status of the corresponding programs. Once this verification is complete, QUEUE places itself back in a JUNK wait until another entry is placed on the Scheduling Queue.

Area 4 - Error Message System

The Error Message System was started during the initiation phase of the INFC program. This portion of the interface consists of two components: a procedure to actually generate the error message and the message printing program (see Figure 12).

Error Message Generation

A standard procedure was developed to generate all of the messages needed by the interface system. This was done to provide a consistent format so all messages contain a date/time stamp and the location of the error within the program. The procedure, CAPSMMSG, utilizes the same type of message catalog file used by the MPE message system and functions the same way as the MPE intrinsic GENMESSAGE. However, CAPSMMSG provides several additional features not available with GENMESSAGE. First, it does not limit the number of variables that can be inserted within the message. The GENMESSAGE intrinsic has a limit of five variables per output message. CAPSMMSG receives an array containing the addresses of the variables and another array identifying the variable type. The variable list is terminated by a -1 address. This design allows CAPSMMSG to incorporate any number variables in a message.

A second feature of CAPSMMSG is the ability to output variables in character, octal, real, double integer, extended real and hexadecimal formats. This allows the message system to format the error information in a manner appropriate to the condition encountered.

An additional feature of CAPSMMSG is the ability to output numeric values in a column format. This improves the readability of messages that contain many numbers.

Error Message File

CAPSMMSG can be called by application programs, interface system programs or procedures. It opens the message catalog file (if not already open) and merges the variables

passed by the calling program/procedure into the specified message. This formatted text is then written to an MPE message file.

Message Printing

The message printing program MSGPRINT was originally initiated by the INFC program. The program simply reads the text placed in the error message file and writes the records to the dedicated printer. In addition to the printer, the messages are also written to a circular file on disc. This allows the error message history to be reviewed on-line by remote support personnel.

The MSGPRINT program continues to perform the read/write function until it receives a record containing a command to terminate. This method is used by INFC to terminate MSGPRINT during the shutdown of the interface system.

Area 5 - Application Program Interface Procedures

As indicated earlier, several routines were developed to provide the application programs access to various aspects of the interface system in addition to improving performance. Four of the more critical areas that were involved include program setup and termination cleanup, receiving/transmitting Series-1 data, HP disc I/O and program error traps.

Procedures SETSTART And EXIT

The two procedures, SETSTART and EXIT, are called by every application program. SETSTART initializes several areas of the program's stack in preparation for other interface procedures. It also arms the various trap procedures and saves the program's current stack marker in the DL-DB area. This stack marker is used by the EXIT procedure to restart the program at the beginning when re-activated after suspending.

The EXIT procedure does the housekeeping activities that are necessary before a program can finish. It then either terminates or suspends the program depending upon the flags for the program contained in the Program Information Area. By suspending frequently executed programs, the interface system reduces the overhead needed to CREATE the process each time. Before EXIT terminates or suspends the calling program, it calculates the programs cpu and elapsed execution time and stores the information in the PIA. This allows the support staff to review the performance of the program by examining the PIA area using a utility program.

When a suspended program is re-activated, EXIT constructs a stack marker based upon the marker stored in the DL-DB area and

returns to the program. This new stack marker causes the program to begin execution at the point immediately after the call to the SETSTART procedure.

Receiving/Transmitting Series-1 Data

Data received from the Series-1 computer is placed in an extra data segment by the LOADBUF program. This made it necessary to design a method to obtain that data for the executing program. A procedure was developed to extract the DST number of the segment containing the data for the program from the program's area in the Control Data Segment. This procedure transfers the amount of data requested to an array in the program's stack (see Figure 13). After each transfer, the procedure updates a pointer in the extra data segment which marks the next available data. When the information in the data segment is exhausted, an end-of-data condition is returned to the calling program.

Sending data to the Series-1 computer also involves a special procedure. The routine would first lock a local RIN that is used to prevent concurrent access to the output message file. When the lock has been obtained, the procedure simply writes the data contained in the program's array to the message file then unlocks the local RIN. The act of placing data in the message file causes INFC to return from its IOWAIT call. INFC then constructs a buffer of data from the records in the message file and transmits the information to the Series-1 computer (see Figure 14).

HP Disc I/O

The various application programs perform large amounts of disc I/O. It was recognized that the best method of improving the I/O performance was to use HP's Disc Caching facility. But another aspect of disc I/O posed an even greater impact on performance and execution of the system.

The existing IBM-1800 application programs performed all of their disc I/O through a set of subroutines. This dictated a similar set of procedures be developed on the HP to minimize the conversion process. It was decided to save the FNUM assigned to the each file to speed access and prevent multiple FOPENs and FCLOSEs.

The standard I/O procedures developed for the HP-3000 expand and utilize the executing program's DL-DB stack area to maintain the numbers assigned to the opened disc files (see Figure 15). In addition to the file numbers, several flags are also maintained in this area for each opened file. One flag might contain a local RIN number that is to be locked to

control access contention. Another flag was used to indicate the file had been modified.

The information for the various files is obtained by the I/O procedures from the File Definition Table file FDNT. Anytime a file is opened, the procedures uses the number that identifies the file to access the file's information in FDNT. The application programs utilize a FORTRAN PARAMETER statement to assign numbers identifying the various files.

When an application program finishes executing it calls the EXIT procedure. This routine insures all local file RINs are released and if any file has been modified, it would force MPE to post the file's buffers to disc. Once this is completed, the EXIT routine would either terminate or suspend the program.

The DL-DB area is also used for storing several other file numbers. As indicated in the illustration, the file numbers for the FDNT file, message catalog file and also the Series-1 output file are also maintained in this area. This is also done to eliminate multiple FOPENS when these files are accessed.

Error Traps

One of the requirements of the system is continuous operation. Should any error occur, appropriate messages must be sent to the support staff and the system should continue to function. When programs are executed in a process tree structure while in batch and they encounter an error, the MPE error message would probably go unnoticed. This made it necessary to trap the error and provide sufficient information about the problem so it could be corrected.

To accomplish this requirement, the SETSTART procedure arms various traps conditions so special interface procedures can be used to handle any errors that might occur. These trap handling procedures are set using the XARITRAP, XLIBTRAP and XSYSTRAP intrinsics.

It is also important to catch any of the other errors that aren't covered by these procedures. These other errors include stack underflows, STT violations, illegal capability, etc. This made it necessary to use the MPE internal procedure XCODETRAP. This allows the interface system to interpret all application program errors except stack overflows. Depending on the type of error, the trap handling procedure issues an appropriate error message and then returns to the program or gracefully terminates.

The only remaining error that needed to be trapped was the stack overflow. The vary nature of a stack overflow indicates insufficient stack space for a user trap handling procedure.

This is why MPE doesn't allow the user access to this type of error. However, it is possible to design a method to catch this error. The solution involves programmatically setting a breakpoint in the MPE internal procedure ABORT. The interface procedure SETSTART, called by all application programs, determines the proper address and sets the breakpoint. However, instead of calling DEBUG when the breakpoint was encountered, it calls the procedure ABORTTRAP which was placed in the system SL.

When a trap occurs and this procedure is invoked, it examines the arguments to the ABORT procedure. If the parameters indicate a stack overflow occurred, the procedure resets the Q and S registers to initial Q, generates the appropriate error message then terminates the program. If the parameters to ABORT indicate anything other than a stack overflow, the procedure would simply exit back into ABORT.

Interface System Performance

Several benchmarks have been conducted to assess the performance of the system in an attempt to improve data throughput. These tests identified several areas in the interface system that required additional efforts to improve performance. The final results of the benchmarks showed an overall increase in data throughput by the new HP system verses the IBM-1800 system.

For example, one test involved executing the same program on the IBM-1800, an HP-3000 Series 48 and also an HP-3000 Series 68. The application program required 25.7 minutes to execute on the IBM-1800. When executed on the Series 48, the program took 8.7 minutes. However, when the program executed on the Series 68, is finished executing just under 4 minutes, an 85% improvement!

Additional tests on the Series 68 showed that the only limiting factor of the system is the communication speed. The 56K baud line between the HP-3000 and the Series-1 is simply too slow! Different memory configurations had only a minor impact on performance.

DISCLAIMER

Many of these same techniques can be employed to adapt the HP-3000 to other similar environments. However, be prepared to spend many hours analyzing memory dumps while you attempt to achieve the proper marriage between your system and the MPE operating system.

ACKNOWLEDGMENTS

I'd like to take this opportunity to acknowledge the following individuals who were involved with this special project:

Peter Howell	for allowing me the time to work on this unique project
Jack Bailie	for giving me a free-hand in designing and programming the system
Terry Tipton	for designing and programming CAPSMSG procedure, the application program I/O routines and the trap-handling procedures.

Also, a special thanks goes to my wife Donna and Terry's wife Mindy. They both had to put up with alot of phone calls and dinner discussions on the intricate details of this project.

REFERENCES

MPE Intrinsic Manual, Hewlett-Packard Company, January, 1985

MPE Systems Table Manual, Hewlett-Packard Company, Updated October, 1981

Communications Handbook, Hewlett-Packard Company, Third Edition, April, 1981

BIOGRAPHY

Jerry Fochtman has been working on computers for over 15 years. His first contact with Hewlett-Packard computers occurred in 1974. At that time HP's Model CX computer was the only machine on the market with simulated wood grain front panels. Since that first meeting, Jerry has been working on HP-3000s in both the educational and manufacturing environments.

As a Systems Analyst for Exxon Chemical Americas, Jerry has been involved in a wide range of special projects ranging from large data entry systems to performance analysis. Besides the HP/IBM link presented in this paper, Jerry has also developed a system to communicate with a Honeywell 4500 process control system.

Exxon Production

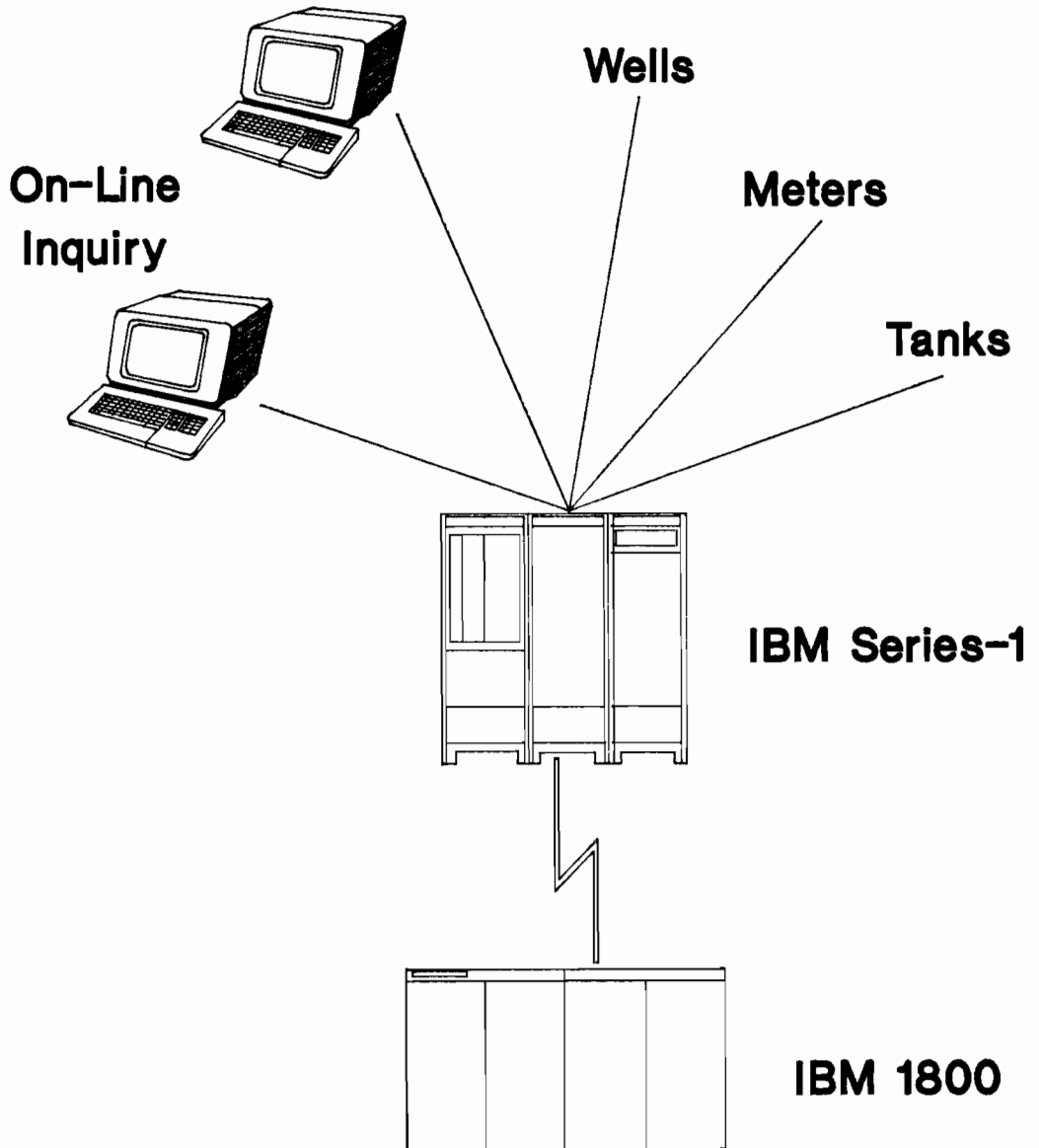


Figure 1

System Overview

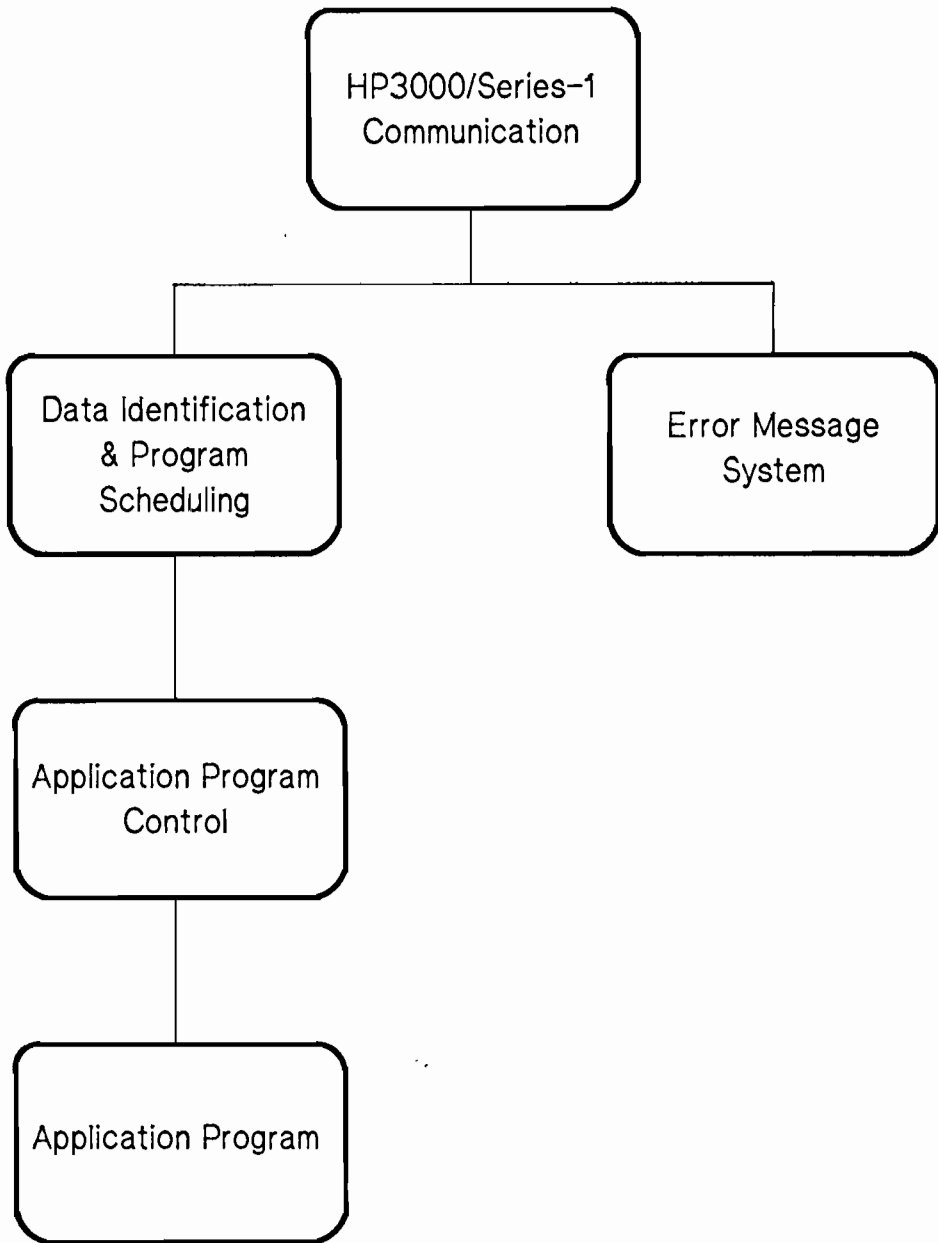


Figure 2

HP-3000/Series-1 Communication

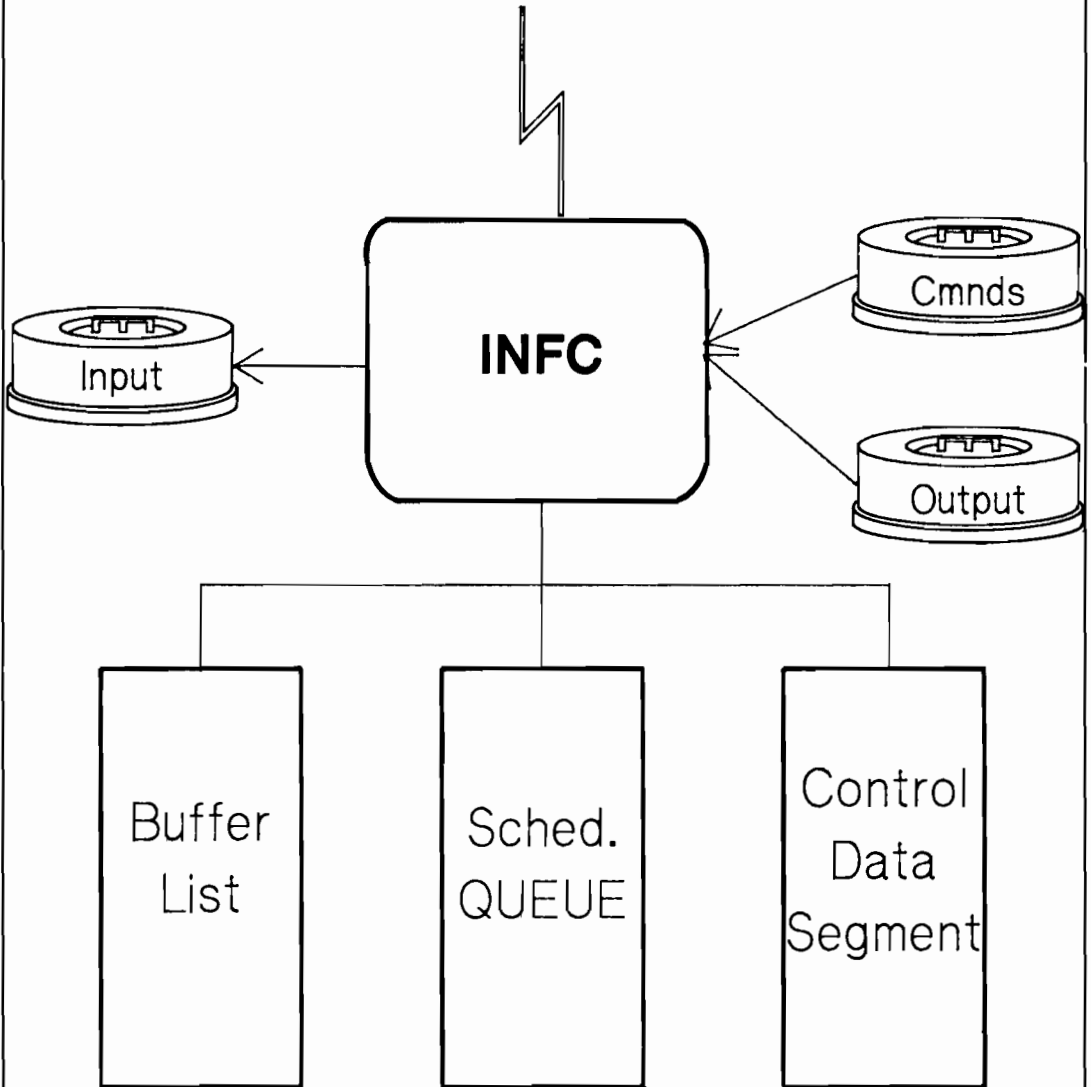


Figure 3

Control Data Segment

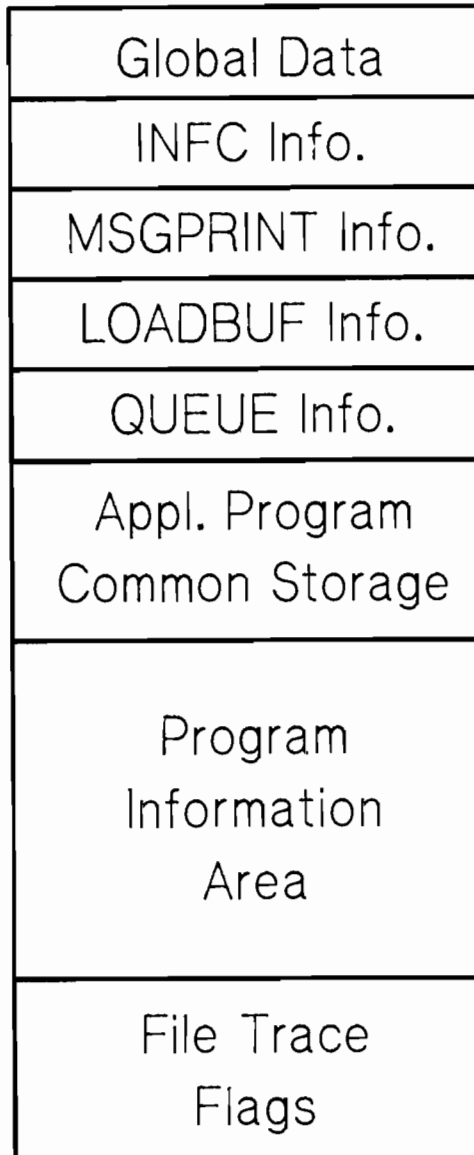


Figure 4

Timer Request List

DST %23

0	
1	
2	
3	# Days Since Start
4	
5	Time Of Day In Milliseconds
6	
7	Date Started
10	
11	
	.
	.

Figure 5

Data Identification and Program Scheduling

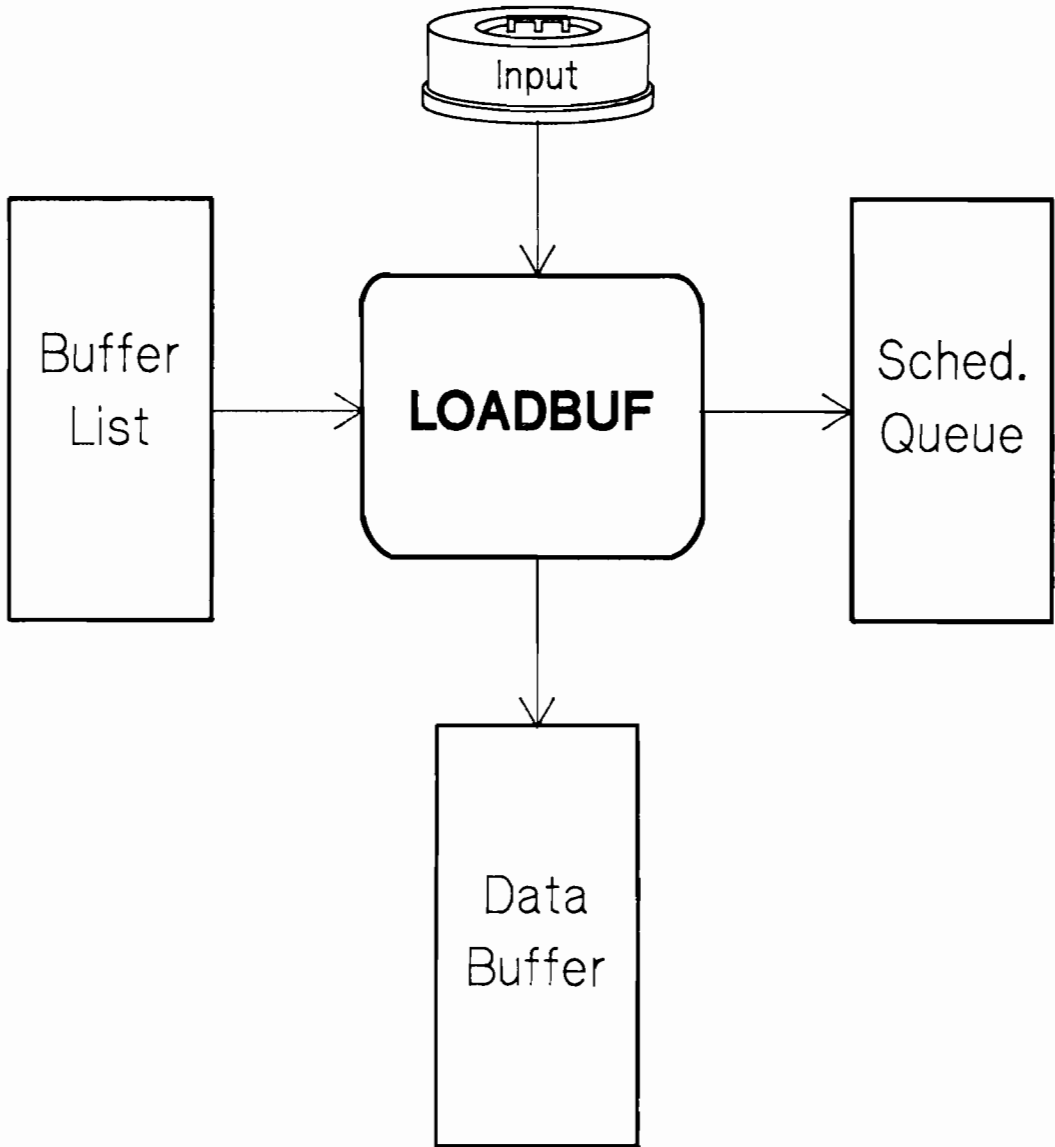


Figure 6

Buffer & Queue Lists

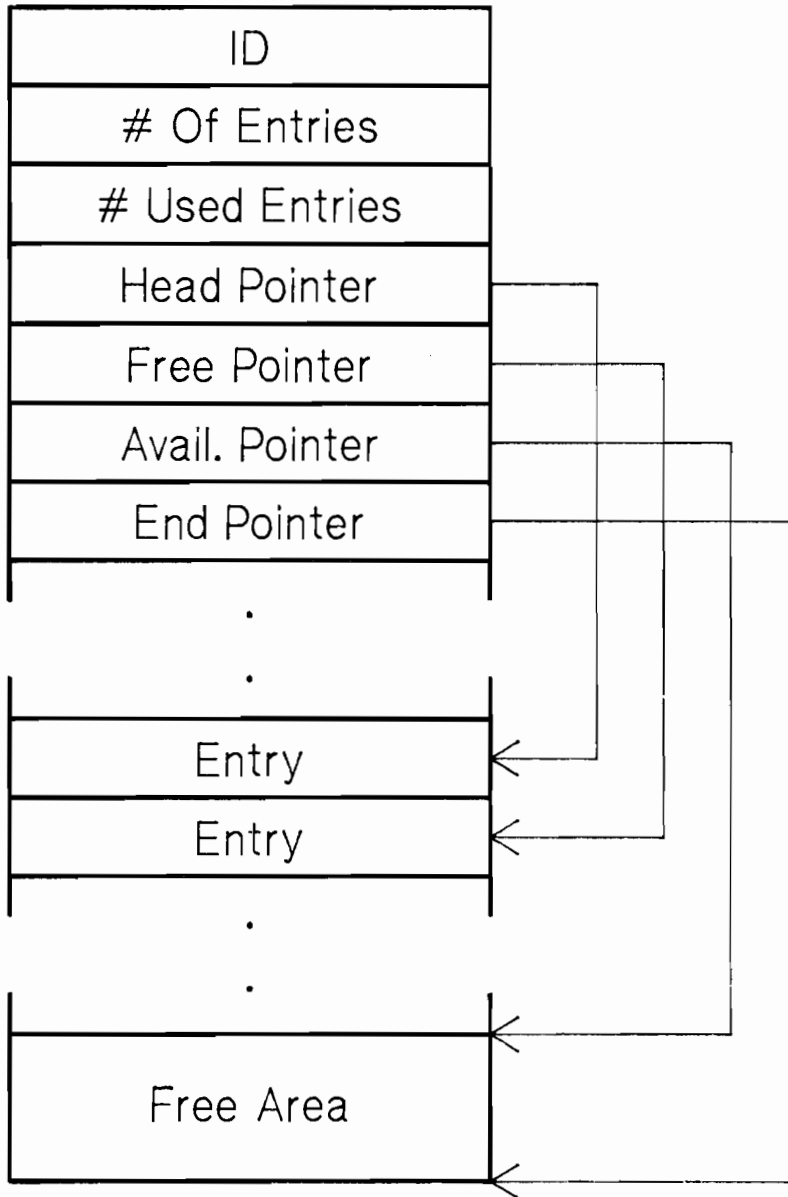


Figure 7

Buffer List Entries

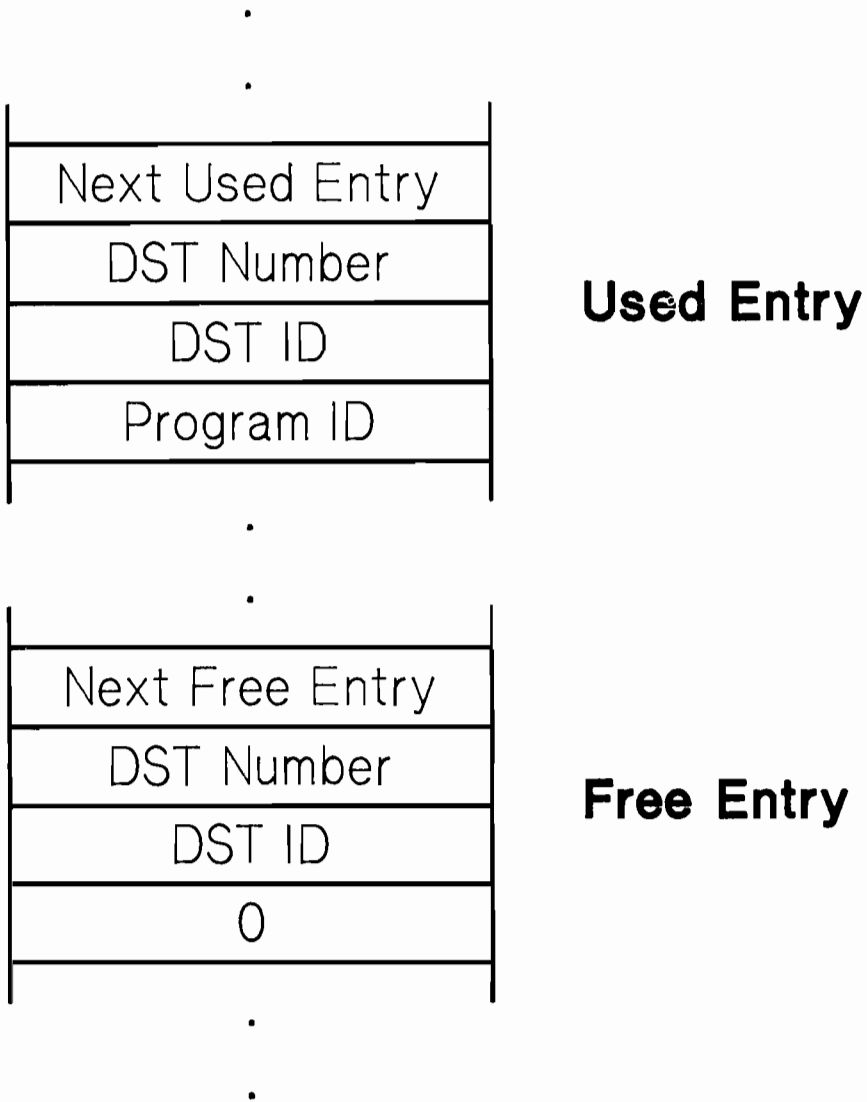


Figure 8

Queue List Entries

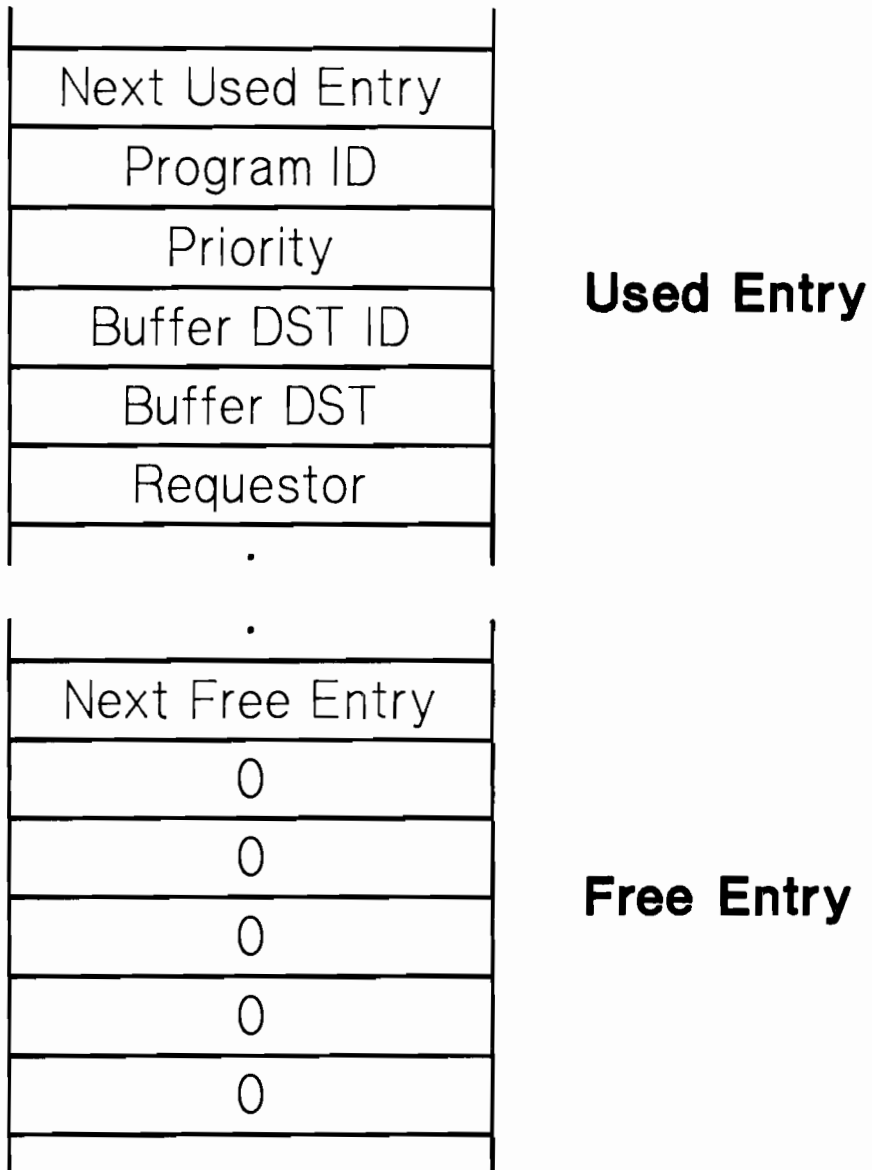


Figure 9

Application Program Control

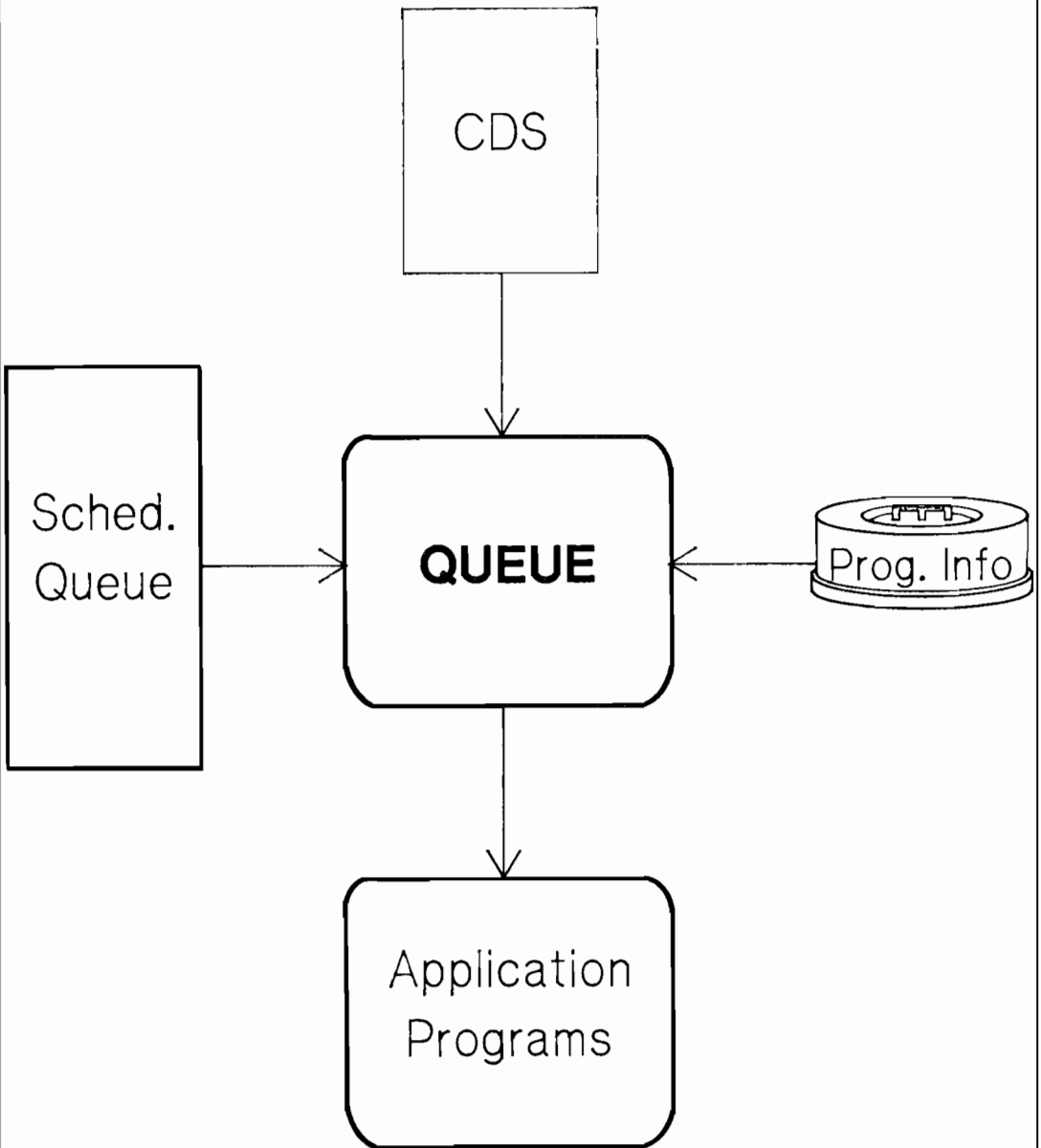


Figure 10

Application Program Execution Modes

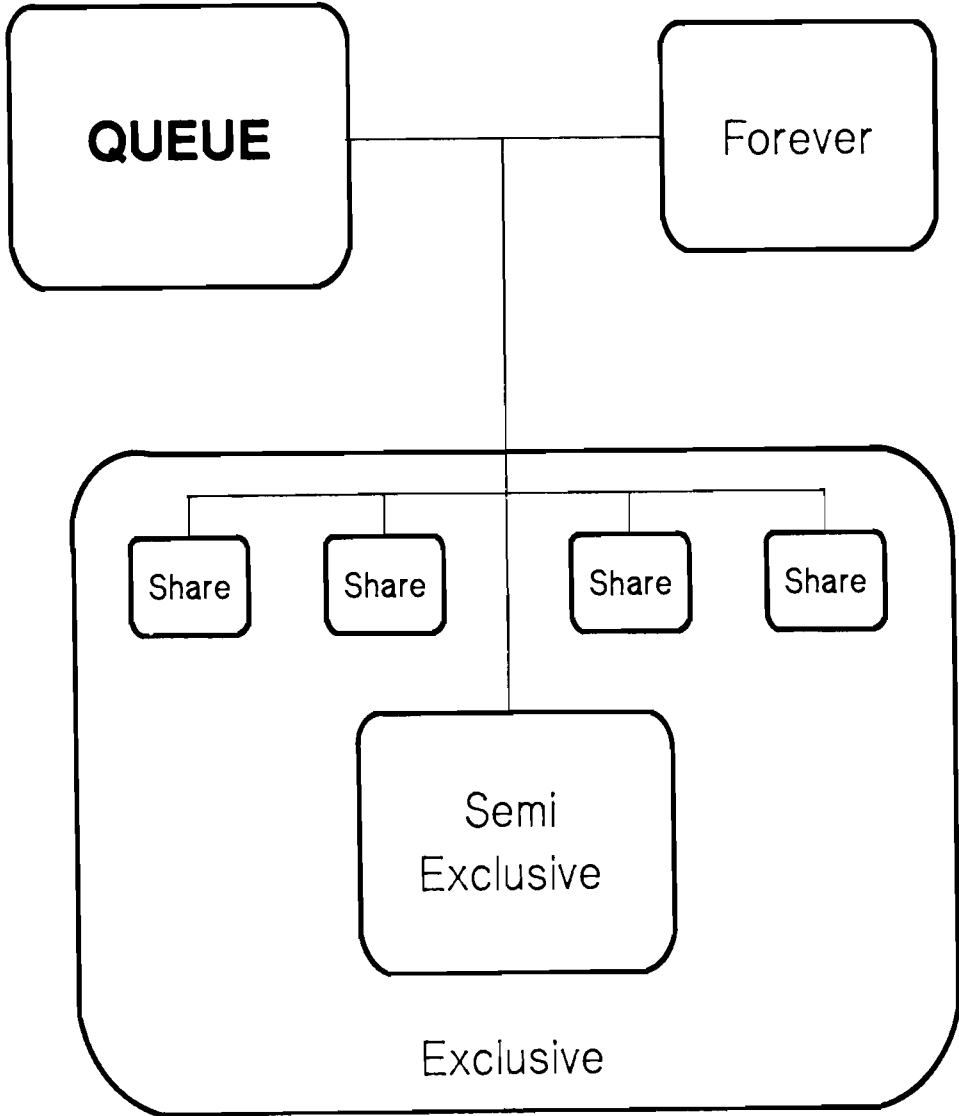


Figure 11

Error Message System

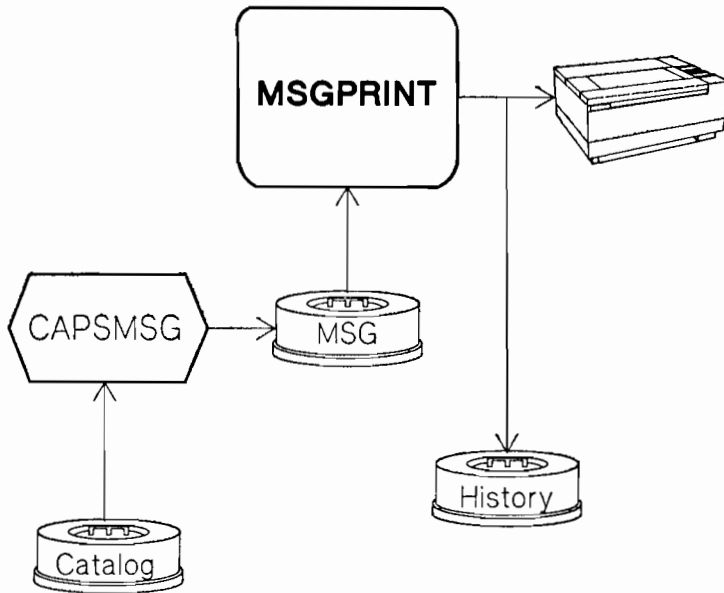


Figure 12

Receiving Series-1 Data

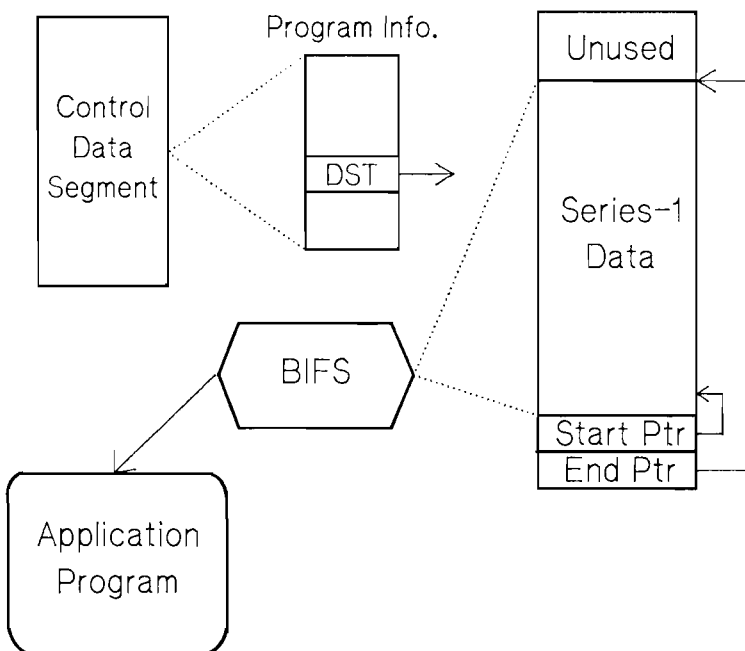


Figure 13

Transferring Data To The Series-1

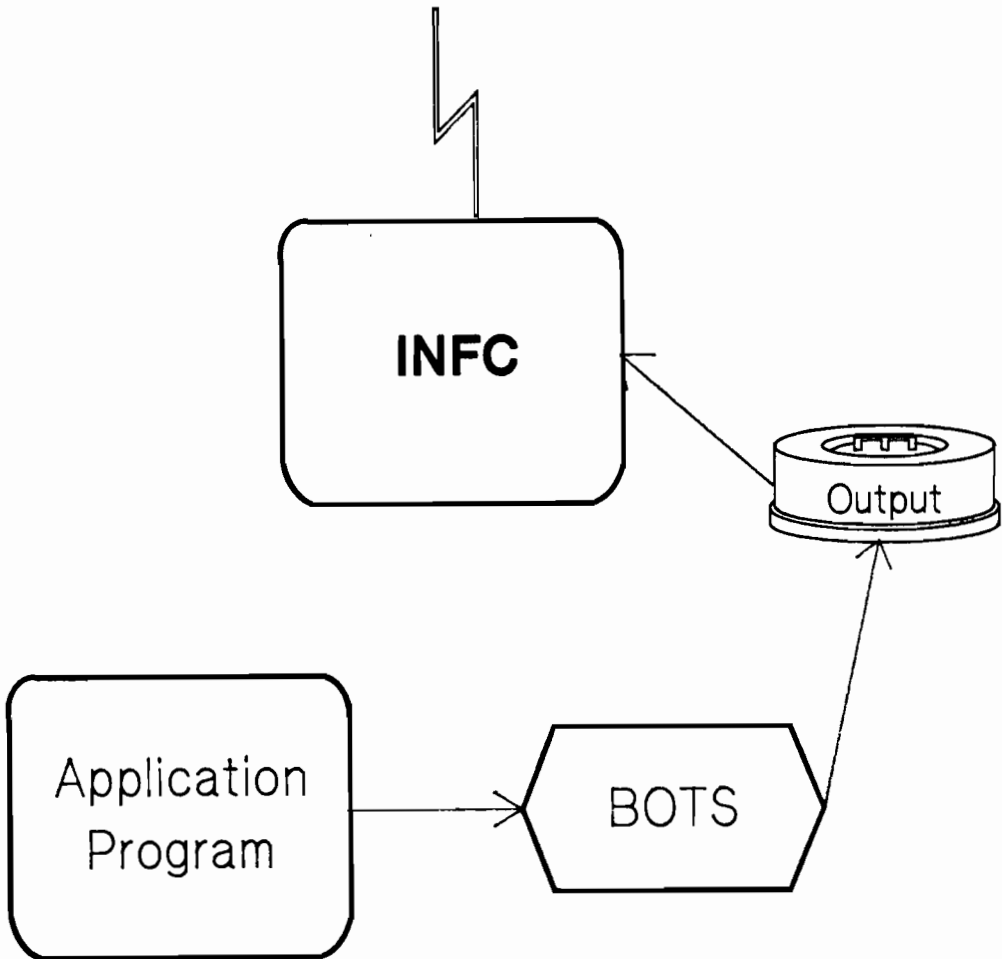


Figure 14

DL-DB Stack Area

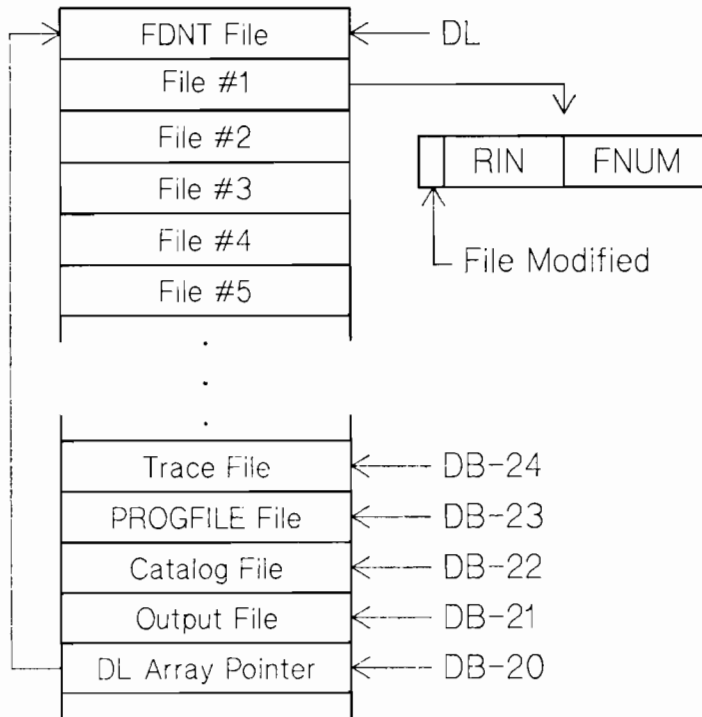


Figure 15

3039. TurboIMAGE Run-Time Options:
Balancing Performance with Data Base Integrity

Peter Kane
Information Technology Group
Hewlett-Packard
Cupertino, California U.S.A.



Summary

Along with improvements in the areas of performance and schema limitations, run-time options were added to TurboIMAGE to allow more flexibility between having high performance and high recoverability. A run-time option is an option which can be used without changes made to any application. In the case of TurboIMAGE, all of the options I will discuss can be enabled through DBUTIL. The purpose of this paper is to compare all combinations of these options, considering the performance impact and what recovery is available.

Introduction

In IMAGE/3000, two run-time options, LOGGING and Intrinsic Level Recovery (ILR) are available. A user can enable LOGGING, ILR, both, or neither. There are two differences between these options. The first is that ILR guarantees only physical integrity, i.e. no broken chains, while using logging and DBRECOV guarantees both physical and logical integrity, i.e. only finished transactions appear in the data base after recovery. The second difference is that with ILR, recovery happens automatically on the first DBOPEN of the data base following an interrupted DBPUT or DBDELETE, while using DBRECOV means restoring an old copy of the data base and waiting while all finished transactions in the log file are issued. Whatever the wait on DBRECOV, it is a much less lengthy process than recovery of a data base not using either ILR or logging, which is by DBUNLOAD and DBLOAD.

Another option which is not a run-time option since it means application changes, is output deferred mode. Output deferred can be enabled only when the data base is opened in mode 3 (exclusive modify access), and is enabled by calling DBCONTROL with mode 1. Output deferred can therefore be used only in single-user environments. This mode can drop significantly the elapsed and CPU times needed by DBPUT, DBDELETE, and DBUPDATE. The reason is that modified buffers and set labels are not written to the data base until either the buffer is needed to hold a different data block, or else a DBCLOSE mode 1 or 2 is issued. The drawback of output deferred can be inferred by this last point, which is that a system failure occurring in the middle of an output deferred application can leave a badly damaged data base. The usual use for output deferred is in a batch environment, where the data base is stored before running the applications. If the system fails while the applications are running, the stored data base would be restored and the applications would then be restarted. Output deferred mode can be used in a session environment, if logging is set up

(otherwise a DBUNLOAD and DBLOAD is necessary if the system fails). However, exclusive access is required, which usually eliminates output deferred for consideration in a session environment.

The above options allow some flexibility, but some issues have been outstanding:

If there are a lot of transactions on a log file, recovery to achieve logical integrity can mean a lot of down-time.

In shortening the maintenance cycle so that recovery can take less time, the data base must be stored more often. Users can not issue transactions against the data base during these times.

Applications using output deferred mode must be operating exclusively. Therefore batch processing can take longer than necessary.

A high volume of modifications has a major impact on performance.

Therefore two more run-time options have been added to TurboIMAGE as answers to the above issues. These are: AUTODEFER, or multi-user output deferred mode, and ROLLBACK, or Rollback recovery which is a faster recovery method than today's recovery method (which I refer to as Roll Forward recovery). The options which have been available in IMAGE have all been carried over to TurboIMAGE, giving four different run-time options. These options can be used in seven different combinations to balance performance and integrity. In this paper I will discuss each combination, specifying the advantages and disadvantages and my recommendation for its use.

Combinations Available with TurboIMAGE

The following is a list of the possible combinations:

1. No options used
2. LOGGING enabled
3. AUTODEFER enabled
4. AUTODEFER and LOGGING enabled
5. ILR enabled
6. ILR and LOGGING enabled
7. ROLLBACK enabled

I would like to now look at each of these separately.

Combination 1: No options used.

Performance: Modified buffers and labels are written to disc (or cache if caching is enabled and BLOCKONWRITE is set to NO), within the intrinsic which did the modification. This means that modify intensive applications must wait frequently for writes to occur. It also means that buffers are never left dirty after an intrinsic finishes, which has a positive impact on read intensive environments (more on this on the discussion of AUTODEFER).

Integrity: If the system fails during processing, physical corruption may result. If this happens, a DBUNLOAD and DBLOAD (other utilities from third parties may be used instead) is necessary to recover the data base. Note that logical recovery is not possible in this case. If disc caching is used and BLOCKONWRITE is set to NO, multiple corruptions can result from a single system failure.

Advantages: The multi-user, read intensive environment probably sees the best performance with this combination. The overhead of logging and ILR is not seen.

Disadvantages: Very lengthy recovery if a system failure has caused physical corruption.

Recommendation: Use for read intensive (approximately 80% reads, 20% modifications) on non-critical data bases. Modify intensive environments can be improved in performance by using other options.

Combination 2: LOGGING enabled.

Performance: Contrary to what many users believe, logging causes only a slight (ranging from about 3% to 8%) degradation in performance. The higher end of this range is usually seen in the modify intensive environments. The reason the degradation is so slight is because with only logging enabled log writes stay in memory until a DBEND or a DBCLOSE is issued, or if the log buffer in memory fills up. Therefore this combination is slightly slower in performance than using no options at all.

Integrity: Physical integrity can be achieved without using the lengthy process of DBUNLOAD and DBLOAD. Logical integrity is possible if the applications are written using DBBEGIN and DBEND. By logging to tape, disc failures can be recovered from.

Advantages: Physical recovery is far less lengthy than DBUNLOAD and DBLOAD. User specific data can be obtained from the log file. Logging to tape or to a different disc from the data base can provide recovery from media failures.

Disadvantages: Dedicated tape drive or usage of disc space for log records. To attain logical recovery, applications must have DBBEGIN and DBEND calls to define logical transactions. Periodic down-time is necessary to back up the data base.

Recommendations: Probably best use is in read intensive environments where logical transactions have been defined, where logical recovery is desired, and where application performance is more of an issue than down-time required to recover from a failure.

Combination 3: AUTODEFER

Performance: Usually will prove to allow the best performance, especially in modify intensive environments. Dirty buffers and labels are not flushed to disc (or cache) until DBCLOSE mode 1 or 2, or unless a buffer is needed to hold a different data block from the data base and all other buffers are dirty. From this one can see that TurboIMAGE will try to keep dirty buffers in memory as long as possible, in an effort to eliminate unnecessary writes to disc. This is useful if users are modifying the same buffer over and over again. However, if one user modifies a buffer containing a data block no other user ever accesses, that block may stay in its buffer for a long time. This will mean less buffers for the other users to do reads, which will in turn mean that buffers may be overlaid with other data blocks before the original user is through. This has an impact on the read intensive environment where there are occasional modifications. In the modify intensive this is not much of an issue because all of the buffers are modified in time.

Integrity: In short, NEVER use AUTODEFER by itself in session environments. This is because the user never has any idea whether the modified blocks or labels (which contains data set end of file, delete chain head, etc.) have made to disc until DBCLOSE time. A DBUNLOAD/DBLOAD is necessary to recovery anything at all if the system fails while applications are running. AUTODEFER is fine with batch processing, if a store of the data base is done first. Then if the system fails, the data base could be restored and the applications redone.

Advantages: Highest performance in applications which do more than an occasionally modification.

Disadvantages: Physical integrity is highly at stake. Logical recovery not possible at all.

Recommendations: Batch processing where applications modify the data base more than occasionally.

Combination 4: AUTODEFER and LOGGING enabled.

Performance: Since the log writes are buffered by MPE until the buffer fills or until DBCLOSE or DBEND, logging adds very little performance overhead in this combination as opposed to having AUTODEFER alone. The performance advantages of AUTODEFER are still realized with this combination.

Integrity: Roll forward recovery is available. Therefore, physical integrity can be achieved, while logical integrity can be achieved if transactions have been defined in the applications using DBBEGINS and DBENDS.

Advantages: High performance in applications which do more than occasional modifications along with a recovery method in case of a system failure.

Disadvantages: Roll forward recovery is not the fastest recovery method. Down-time is necessary to back up the data base periodically. Dedicated tape drive or disc space is necessary.

Recommendations: Use in session environment where application performance is highest concern, and where data base modifications are done more than occasionally.

Combination 5: ILR enabled.

Performance: Performance degradation with modifications, for two reasons. The first is because there are at least two additional writes to disc to update the ILR file for each DBPUT and DBDELETE. The second reason is that all writes to the data base and to the ILR file use BLOCKONWRITE=yes. TurboIMAGE calls FSETMODE to set this option if it determines that ILR is enabled. BLOCKONWRITE causes the process issuing the write to wait until the I/O has completed to disc (otherwise the user will only wait until the I/O completes to a cache domain). The user has no control over this use of BLOCKONWRITE, but should be aware of it.

Integrity: Automatic physical recovery on the first DBOPEN of the data base following an interrupted DBPUT or DBDELETE. ILR has been enhanced to redo the interrupted intrinsic rather than rolling it out as in IMAGE. No logical recovery is available.

Advantages: Automatic physical recovery. Not necessary to store the data base.

Disadvantages: Performance degradation for applications using a high number of DBPUTs and DBDELETES. No logical recovery available. Not compatible with AUTODEFER.

Recommendations: Use on critical data bases where logical recovery is not applicable and where it is crucial that the data base be accessible at all times. Performance degradation less where the number of DBPUTs and DBDELETES is fewer.

Combination 6: ILR and LOGGING enabled.

Performance: Logging will cause a slight amount of degradation over having ILR enabled alone.

Integrity: Can achieve quick and automatic physical recovery with ILR, and can have logical recovery with DBRECOV.

Advantages: Quick physical recovery method. Logical recovery available. Back up method for physical and logical recovery if ILR file is corrupted.

Disadvantages: Must store the data base and start new logging cycle after each time ILR is needed to recover (reported to console), if do not wish to use DBRECOV.

Recommendations: This method had its merits with IMAGE, where rollback recovery was not available. With TurboIMAGE, I would recommend using either ILR alone or ROLLBACK.

Combination 7: ROLLBACK enabled (ILR and LOGGING must also be enabled).

Performance: This combination has some more performance degradation over Combination 6. This is because log writes will now be written directly to disc using BLOCKONWRITE instead of being buffered by MPE as in all of the other combinations using logging. DBPUT and DBDELETE will see the most performance degradation.

Integrity: Rollback recovery used for system failure or "soft crash". Roll forward recovery can be used if there has been a media failure or "hard crash". Logical and physical recovery are both available.

Advantages: Quick logical and physical recovery (rollback is much faster than roll forward). Stores of the data base do not have to be taken, but should be done somewhat periodically as protection against a media failure.

Disadvantages: Performance degradation, especially on DBPUTs and DBDELETes. DBUPDATEs also affected.

Recommendations: Use where transactions have been defined and where it is crucial that the data base be available, and logically intact.

3040. Building Your Own X.25 Data Network

Donn Lewis
Allegheny Beverage Corporation
1 Macke Circle
Cheverly, Maryland 20781

During the last several years, many HP 3000 users have begun to require greater connectivity between their systems. HP's Advancenet product with support of X.25 standards offers a cost effective solution to their problem.

Many older networks of HP 3000 computers began as two small systems located in the same room or building. Data communications between those systems was a relatively rare event.

Today many corporations are building larger networks of HP 3000's spread throughout the United States. The older distributed systems products (DS/3000 using Bisync connections) offered excellent solutions for relatively small networks, but had significant problems when dealing with networks where high connectivity was required and where the number of nodes rose above three or four. The older DS products only addressed CPU to CPU communications and did not consider the need for workstations located far away from the CPU to communicate economically over a switched voice network. Today, applications such as inventory control, stock status and order entry require reliable and economic data communications solutions for successful implementation.

The Environment

In order to structure the solution to the networking problem, the desired functional capabilities of the network were developed. The following is a brief description of the operating environment of the network:

Because the company was acquisition oriented, the size of the network over a five year period could not be precisely determined.

Due to the uncertainty of sizing the network, the exact locations of the nodes could not be predetermined either.

Because nodes would exist in different time zones and be locally controlled and operated, every node had to be able to access every other node directly without passing through an intermediate node.

Remote terminals had to be able to access the nodes in an economical and reliable manner. Block mode access was highly desirable.

Due to the volume of data projected for the first several nodes, a transmission bandwidth of at least 9600 baud was required.

Assumptions

In order to simplify the presentation of the solutions presented in this paper the following assumptions are made:

X.25 Software costs the same as Bi-Sync DS/3000 software.

Leased lines connecting nodes by both methods cost \$ 300 per month per line.

In order to insure that any node can access any other node directly and because under Bi-Sync DS each connection requires a separate INP, we will assume for analysis purposes that an HP 3000 can contain an unlimited number of INP's.

In our search for a solution we tried several alternatives early in the game:

Dial Up DS

One of the capabilities of Bi-Sync DS is the ability to use dial up modems connected to an auto-dialer. At MPE configuration time, you can define node names with phone numbers and allow one machine to dial another. At the time of our test the state of the art for dial up synchronous modems was 4800 baud. We tested the solution with three nodes and we found the following results:

The technology was unreliable. The quality of the particular voice line which we obtained was so variable that the mean time between failure of the line was less than 30 minutes.

Performance was not adequate. Because the modems are half duplex, the actual throughput of the line was approximately 300 characters per second.

Since the line could only be connected between two nodes at any one time, certain operations were delayed until the line was released (See Figure 1). Since each user had to specifically CLOSE the line, the connection was sometimes attached to the wrong node and the user who had the line open could not be detected. This made the technical staff very dissatisfied with the dial up solution.

X.25 via a Public Data Network

We next tried to link up our three nodes using UNINET, a public data network supplier of X.25 packet switching value added services. UNINET, at the time of our test, was the least expensive of the three major suppliers of X.25 services (TELENET, TYMNET and UNINET). We encountered major problems in attempting to use UNINET for HP 3000 X.25 communications. Among the difficulties we encountered were:

HP X.25 software had not been certified to operate over UNINET. There were significant problems with getting the HP software to operate. We had numerous system failures and the local HP software engineers were neither trained nor capable of dealing with the problem. There were PICS personnel from Atlanta involved in the process, but the turn around time for resolutions was so long that both UNINET and our technical people became frustrated with trying to make this work.

We eventually were able to dial into UNINET over their PAD to our HP 3000, but we could never make block mode applications function. The final information from PICS was that HP X.25 had certification problems and that it would not function correctly over UNINET until a future release.

In our discussions with PICS we came across some personnel at Atlanta who were involved in solving the problem of creating small packet switched networks within the support center to allow diagnostic services for user problems called in via PICS.

These people were essentially building their own packet switched network within the building (because at that time no local area network product existed) using standard HP software and X.25 equipment manufactured by a company in Virginia called DYNATECH.

After talking with the SE's in Atlanta, we decided to perform a financial analysis of private X.25 versus conventional DS and at the same time analyze the technology to see if it would meet our requirements for a long term solution to our data communication requirements. See Figure 2.

We decided to financially look at a private X.25 network using the following model:

Model Assumptions

INP Cost	7,560
1st Copy of DS	4,000
Right to Copy DS	2,000
X.25 Switch Cost	6,300
Modem Cost	1,800
Line Cost/Year	3,600

Standard DS Solutions Using Point to Point Lines

# Nodes	INP's	Modems	Software	Lines	Cost
2	2	2	6,000	3,600	28,320
3	6	6	8,000	10,800	74,960
4	12	12	10,000	21,600	143,920
5	20	20	12,000	36,000	235,200
6	30	30	14,000	54,000	348,800
7	42	42	16,000	75,600	484,720
8	56	56	18,000	100,800	642,960
9	72	72	20,000	129,600	823,520
10	90	90	22,000	162,000	1,026,400

Advancenet Solution using X.25 Protocol

# Nodes	INP's	Modems	Software	Lines	Cost	Savings
2	2	2	6,000	3,600	34,620	(6,300)
3	3	4	8,000	7,200	51,380	23,580
4	4	6	10,000	10,800	68,140	75,780
5	5	8	12,000	14,400	84,900	150,300
6	6	10	14,000	18,000	101,660	247,140
7	7	12	16,000	21,600	118,420	366,300
8	8	14	18,000	25,200	135,180	507,780
9	9	16	20,000	28,800	151,940	671,580
10	10	18	22,000	32,400	168,700	857,700

Several conclusions can be easily seen in the data:

Because the use of X.25 protocols requires only a single INP and associated communications equipment (modems and line), the cost of an X.25 network is significantly less expensive than older DS solutions.

The savings increase at a greater than linear rate due to the fact that in an older DS configuration (N-1)N INP's are needed and that this causes an additional requirement of (N-1)N/2 point to point phone lines.

X.25 only makes sense (of course) if you have more than two nodes or expect your network to grow beyond two nodes.

Our experience with setting up the DYNATECH equipment was surprisingly easy. In order to configure the switches you connect one of the DYNATECH pads to the switch, and the configuration is no more difficult than a statistical multiplexor. One terminal connected to the pad acts as a system console, and can perform diagnostics, capture utilization statistics of each X.25 trunk, or perform system network reconfiguration from a central site.

On the 3000 there is a network configuration program that essentially requires you to give your packet addressing scheme to Advancenet. The documentation from HP on the whole X.25 configuration process left a great deal to be desired. A utility program in PUB.SYS builds a small database which the X.25 software accesses at execution time to determine addressing. This configuration process was a bit tedious, but after you get the first two nodes configured, it is quite simple. HINT: Don't forget to RELEASE the database or only MANAGER.SYS can open up a DSLINE. The advantage to having this information reside within a database is that changes in the configuration of your network DO NOT require MPE reconfiguration. This allows you to add, change or delete nodes from the network during normal working hours.

All normal DS commands (DSLIN, DSCONTROL etc.) work exactly the same under X.25. Users will not notice any functional differences between the two protocols.

HP has updated some of their products to function under X.25 in an efficient manner. HPDESKMANAGER, for example, allows the administrator to define the nodes of the mail network in terms of the X.25 designation, and mail transfers between systems are accomplished under control of HPDESK using the X.25 network. Because X.25 permits all CPU's to communicate directly with one another, X.25 allows simultaneous transfers of mail over a single INP and also allows mail to be transferred directly to the target CPU without passing through an intermediate node, and delaying transmission. We transfer mail between all nodes of the network every fifteen minutes.

Phase II

After our experience with using the DYNATECH packet switches we decided to attempt to connect our small private network to TELENET. This connection would effectively allow our users located anywhere across the country to dial into TELENET pads and connect to any CPU within our network.

HP X.25 software had been certified by TELENET and our connection to the network was without incident. We now support several users spread across the country who access our electronic mail system and supply financial data into our general ledger. Our current configuration is represented by Figure 3.

Our Experience with X.25

One of the unexpected characteristics of using X.25 protocols on our network is the apparent reliability of the network. Whereas Bi-Sync DS frequently caused ghost sessions and required INP restarts or MPE warm starts to clear up certain datacomm related problems, X.25 is extremely robust. If a particular node goes down, the software seems to be very forgiving. The loss of any number of nodes does not affect the ability to communicate to any other still surviving nodes. When the inoperative node comes up, access to the node is again established, and the user may DSLINE to the node. The worst we have ever had to do when a single node cannot be addressed is DSCONTROL SHUT and then OPEN the line. An MPE warm start has never been required to assist in datacomm related problems. This has contributed significantly to the uptime and performance of our network. Our TELENET connection has had an uptime of 100%.

Performance under X.25 for certain functions appear to out perform old DS. This is supported by documentation in the new HP 3000 Performance Guide. DSCOPY and other utilities just seem to run faster over an X.25 network.

All other DS functions such as remote database access, virtual terminal capability work with no changes.

Conclusion

Advancenet using X.25 protocols is cheaper, faster, more reliable and more flexible than the old DS product. It permits both terminal to CPU AND CPU to CPU communications. If a user has a geographically distributed (wide area) network, X.25 represents the superior solution for a communications solution.

DIALUP DS/3000 COMMUNICATIONS

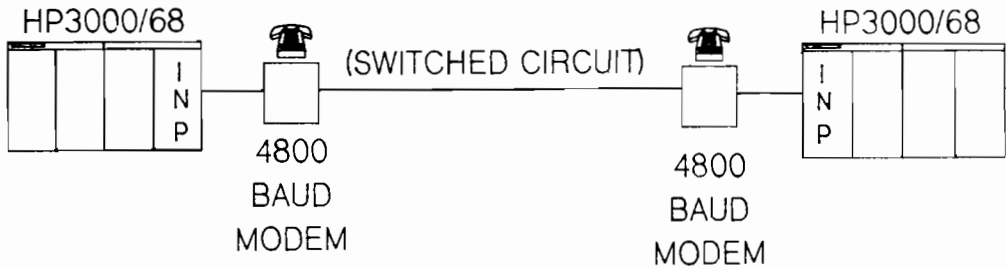
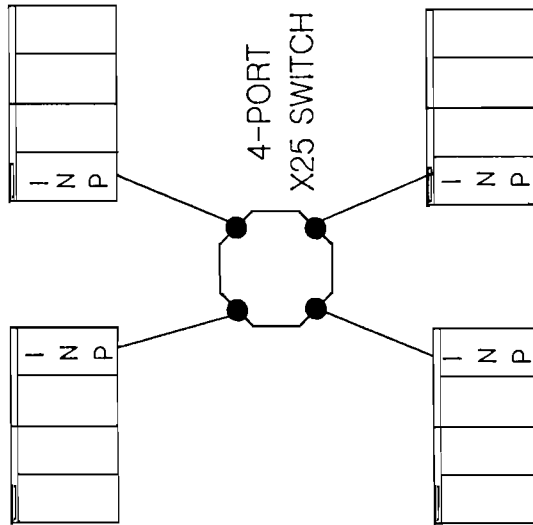


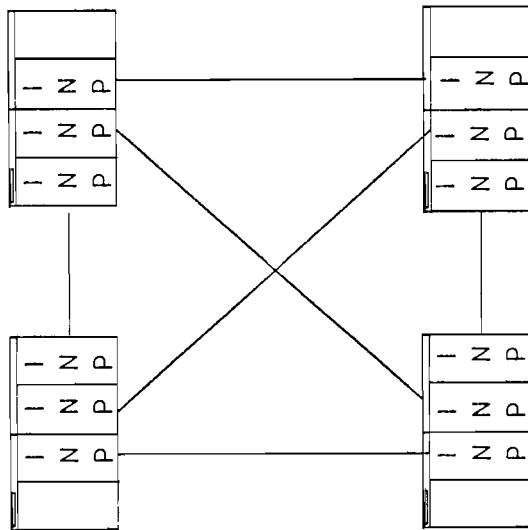
FIGURE 1

Figure 2

X.25
DS/3000



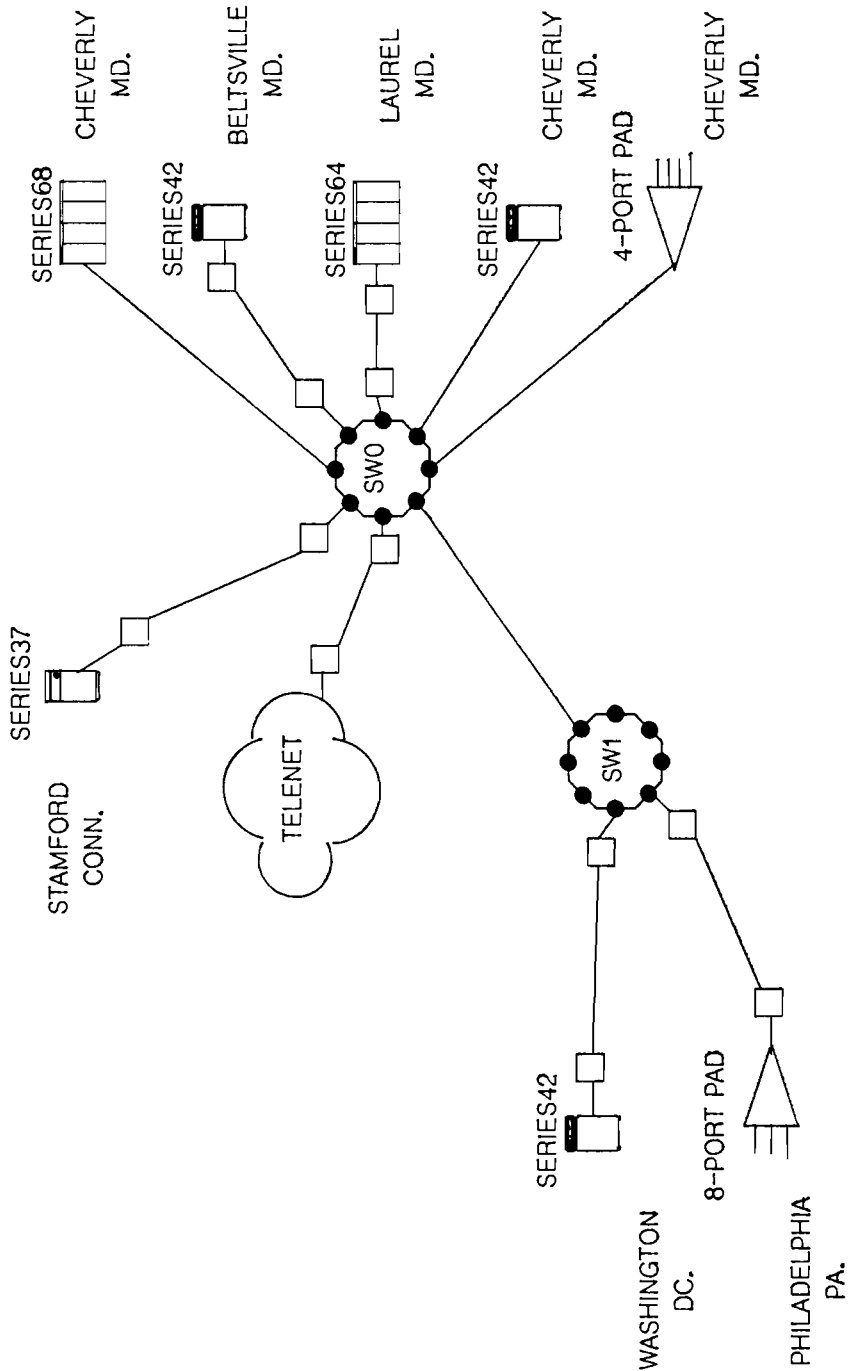
BI-SYNC
DS/3000



NOTE: ALL systems are HP/3000 CPUs

Figure 3

ALLEGHENY BEVERAGE CORPORATION X.25 DATA COMMUNICATIONS NETWORK



3042. Migrating Information Between HP-3000 Data Bases,
Electronic Spreadsheets, and Microcomputer Data Bases

Thomas J. Kaminski
Director, Data Processing
SINGER Education Division
80 Commerce Dr.
Rochester, New York 14623



I. A Need for the Migration of Information.

How many times have you watched as your company accountant enters information into a personal computer from a report produced by your HP-3000 accounting system? Has your marketing department ever built a small file of contacts on their personal computer using a list extracted from your HP-3000 sales and marketing system? Do managers enter their budgets into the HP-3000 computer from a listing created by spreadsheet software?

In each case, this information is already stored on some type of computer media. It seems redundant to manually type this information in over again. There should be a way to copy the information from one computer system to another. To do this, you would need three items:

1. A method of linking your systems together electronically.
2. Software to enable the exchange of files between systems.
3. Software to convert data from one system to another.

The first two items are of no concern in this paper. There are a number of ways to connect your personal computer to the HP-3000 computer and transfer files back and forth.

This paper will deal with the third item, software that converts data from one system to another. Some features of this software include:

- Transportability - convert any data from one system to any other system
- Flexibility - change data types, summarize data, or compute data in the process
- Easy to Use - once the routines are setup, they can be used over and over again by the users of the computer
- No Heavy Programming - the system only uses fourth generation languages, report writers, and standard interfaces

The best feature out of all these is that it's free! The methods presented in this paper use software that you already have available on your computers.

With a little practice and experimentation, you will be able to transfer data from almost any system to almost any other system.

II. Methods of Migration.

Most software packages have facilities to convert data from one format to another. Lotus 1-2-3 has a translate program. R:base Series 4000 contains a flexible unload option. Other software packages allow you to format reports which can be sent to disc files instead of the printer, allowing you to convert your data from one form to another.

Five popular software packages will be discussed with instructions on how to create interfaces between them. The methods of data conversion we will be using with each software package are listed below:

Software (computer)	Software Vendor	Reference	Type of Data Conversion
IMAGE/QUERY (HP-3000)	Hewlett- Packard	1a	REPORT (to a disc file) in DIF format
		1b	REPORT (to a disc file) in DELIMITED format
		1c	UPDATE ADD
Powerhouse (HP-3000)	COGNOS	2a	QUIZ - SET REPORT DEVICE DISC create DIF file
		2b	QUIZ - SET REPORT DEVICE DISC create DELIMITED file
		2c	QDD - Define a file
Lotus 1-2-3 (PCs)	Lotus Development Corp.	3a	.WKS to .DBF
		3b	.WKS to .DIF
		3c	.DBF to .WKS
		3d	.DIF to .WKS
dBase II (PCs)	Ashton-Tate	4a	APPEND FROM file DELIMITED
		4b	COPY TO file DELIMITED
		4c	SET ALTERNATE to FILE program (QUERY UPDATE ADD format)
R:base Series 4000 (PCs)	Microrim	5a	UNLOAD DATA FOR rel AS DIF
		5b	LOAD rel FROM file AS DIF
		5c	UNLOAD DATA FOR rel AS ASCII
		5d	LOAD rel FROM file AS ASCII
		5e	OUTPUT file & PRINT report (QUERY UPDATE ADD format)

By using these data conversions in the right combinations, we can migrate data from one system to most of the other systems.

Other software that is not mentioned here use similar interfaces.

Cross Reference Chart

Methods of Data Migration

From: To:	IMAGE/ QUERY	Power- house	Lotus 1-2-3	dBase II	R:base 4000
IMAGE/ QUERY			Note 1	4c / 1c	5e / 1c
Power- house			Note 1	4b / 2c Note 2	5c / 2c Note 2
Lotus 1-2-3	1a / 3d	2a / 3d		3c	5a / 3d
dBase II	1b / 4a	2b / 4a	3a		5c / 4a
R:base 4000	1a / 5b	2a / 5b	3b / 5b	4b / 5d	

Note 1: I have found no easy way to perform conversions from Lotus 1-2-3 to other HP-3000 systems as of yet. One suggestion is to convert it to other personal computer software file types first (if available) and then translate it. Other suggestions will be gratefully accepted.

Note 2: This type of conversion is specialized. Step 2c depends on the application and is not discussed in detail in this paper.

To use this chart, simply look up the conversion that you wish to perform, then go to the sections in this paper that describes the steps of that conversion.

III. Sample Data Bases.

To test the routines used in this paper, data bases were set up in each of the data base systems used. To understand the examples better, the commands used to create the data bases are listed here.

Sample data

To test out all of the procedures, a small budgeting system was used. Sample data for one department (#40 - Data Processing) was set up:

Account Number	Description	Jan Jul	Feb Aug	Mar Sep	Apr Oct	May Nov	Jun Dec
40-1010	Salaries	10000	10000	12000	10000	11000	13000
		11000	11000	14000	12000	12000	14000
40-2010	Staff Training	800	0	800	0	800	0
		800	0	800	0	800	0
40-2020	Memberships	0	60	0	60	0	60
		0	60	0	60	0	60
40-2030	Subscriptions	60	20	60	20	60	20
		60	20	60	20	60	20
40-2040	Travel	1400	800	1000	1200	1600	800
		1200	800	1000	1600	1200	600
40-3010	Office Supplies	300	300	300	300	300	300
		300	300	300	300	300	300
40-3020	Telephone	50	50	50	50	50	50
		75	75	75	75	75	75
40-3030	Courier Services	100	100	100	100	100	100
		100	100	100	100	100	100
40-5010	Depreciation	7200	7200	7200	7200	7200	7200
		7600	7600	7600	7600	7600	7600
40-5020	DP Hardware Maint.	1800	1800	1800	1800	1800	1800
		1800	1800	1800	2000	2000	2000
40-5030	DP Software Maint.	1500	0	200	1500	900	0
		1500	0	4000	1500	3600	0
40-5040	Contract Programming	2000	1000	500	0	1000	0
		0	1000	0	0	1000	0
40-5050	EDP Paper & Supplies	1800	1800	1800	1800	1800	1800
		1900	1900	1900	1900	1900	1900
40-5060	Micrographics	500	500	500	500	500	500
		700	700	700	700	700	700
40-5070	EDP Equip. Rental	800	800	800	1000	1000	1000
		1200	1200	1200	1200	1200	1200

The account number consists of a two-digit department number and a four-digit sub-account number.

Powerhouse

The Powerhouse schema was the first data base created. A data input screen was built and data from here was translated to all of the other data bases on the system.

Schema "1986 Budgets"

File ACCT-NUMBERS	Type IMAGE Automatic Master of BUDGET	&
	Password "SUPER"	&
	Capacity 100	&
Description		&
	"File contains account numbers and acts as an index"	&
	"file to the ACCT-DETAIL"	
File ACCT-DETAIL	Type IMAGE Detail of BUDGET	&
	Password "SUPER"	&
	Capacity 100	&
Description		&
	"File contains account numbers, descriptions, and"	&
	"12 monthly budget figures"	
Element ACCT-NUMBER	9(6) Picture "00-0000"	&
	Significance 7 Heading "Acct.^No."	&
	Help "Account number (DDNNNN) DD=dept NNNN=number"	
Element ACCT-DEPT	9(2) Picture "^^"	&
	Significance 2 Heading "Acct.^Dept."	&
	Help "Department Number (00-99)"	
Element ACCT-SUB	9(4) Picture "0000"	&
	Significance 4 Heading "Acct.^Sub."	&
	Help "Sub-Account Number (0000-9999)"	
Element DESCRIPTION	X(20) Heading "Description"	&
	Help "Account Number Description"	
Element BUDGET-JAN	9(7) Heading "Jan"	&
	Help "Budget for January"	
Element BUDGET-FEB	9(7) Heading "Feb"	&
	Help "Budget for February"	
Element BUDGET-MAR	9(7) Heading "Mar"	&
	Help "Budget for March"	
Element BUDGET-APR	9(7) Heading "Apr"	&
	Help "Budget for April"	
Element BUDGET-MAY	9(7) Heading "May"	&
	Help "Budget for May"	
Element BUDGET-JUN	9(7) Heading "Jun"	&
	Help "Budget for June"	

Element BUDGET-JUL	9(7)	Heading "Jul"	&
Help "Budget for July"			
Element BUDGET-AUG	9(7)	Heading "Aug"	&
Help "Budget for August"			
Element BUDGET-SEP	9(7)	Heading "Sep"	&
Help "Budget for September"			
Element BUDGET-OCT	9(7)	Heading "Oct"	&
Help "Budget for October"			
Element BUDGET-NOV	9(7)	Heading "Nov"	&
Help "Budget for November"			
Element BUDGET-DEC	9(7)	Heading "Dec"	&
Help "Budget for December"			

Record ACCT-NUMBERS

Item ACCT-NUMBER Zoned Unique Key

Record ACCT-DETAIL

Item ACCT-NUMBER Zoned Unique Key Links to ACCT-NUMBERS

Redefined by

Item ACCT-DEPT Zoned

Item ACCT-SUB Zoned

End

Item DESCRIPTION

Item BUDGET-JAN Packed

Item BUDGET-FEB Packed

Item BUDGET-MAR Packed

Item BUDGET-APR Packed

Item BUDGET-MAY Packed

Item BUDGET-JUN Packed

Item BUDGET-JUL Packed

Item BUDGET-AUG Packed

Item BUDGET-SEP Packed

Item BUDGET-OCT Packed

Item BUDGET-NOV Packed

Item BUDGET-DEC Packed

Build

Exit

IMAGE

The IMAGE schema was created using the QUTIL utility provided by the Powerhouse software package.

```
BEGIN DATA BASE BUDGET;
  PASSWORDS:
    1  SUPER;          << MASTER PASSWORD >>

  ITEMS:
    ACCT-NUMBER,      Z6      (/0,1);
    BUDGET-APR,       P8      (/0,1);
    BUDGET-AUG,       P8      (/0,1);
    BUDGET-DEC,       P8      (/0,1);
    BUDGET-FEB,       P8      (/0,1);
    BUDGET-JAN,       P8      (/0,1);
    BUDGET-JUL,       P8      (/0,1);
    BUDGET-JUN,       P8      (/0,1);
    BUDGET-MAR,       P8      (/0,1);
    BUDGET-MAY,       P8      (/0,1);
    BUDGET-NOV,       P8      (/0,1);
    BUDGET-OCT,       P8      (/0,1);
    BUDGET-SEP,       P8      (/0,1);
    DESCRIPTION,      X20     (/0,1);
```

SETS:

```
NAME:    ACCT-NUMBERS,AUTOMATIC (/0,1);
ENTRY:    ACCT-NUMBER(1);
CAPACITY: 100;
```

```
NAME:    ACCT-DETAIL,DETAIL (/0,1);
ENTRY:    ACCT-NUMBER(ACCT-NUMBERS),
          DESCRIPTION,
          BUDGET-JAN,
          BUDGET-FEB,
          BUDGET-MAR,
          BUDGET-APR,
          BUDGET-MAY,
          BUDGET-JUN,
          BUDGET-JUL,
          BUDGET-AUG,
          BUDGET-SEP,
          BUDGET-OCT,
          BUDGET-NOV,
          BUDGET-DEC;
CAPACITY: 100;
END.
```

dBase II

The dBase II version of the BUDGETS file was created to contain a department's budget. Only the four-digit sub-account is used.

```
. CREATE BUDGETS
ENTER RECORD STRUCTURE AS FOLLOWS:
FIELD  NAME,TYPE,WIDTH,DECIMAL PLACES
001    ACCT:SUB,C,4
002    DESC,C,20
003    BUD:JAN,N,8,0
004    BUD:FEB,N,8,0
005    BUD:MAR,N,8,0
006    BUD:APR,N,8,0
007    BUD:MAY,N,8,0
008    BUD:JUN,N,8,0
009    BUD:JUL,N,8,0
010    BUD:AUG,N,8,0
011    BUD:SEP,N,8,0
012    BUD:OCT,N,8,0
013    BUD:NOV,N,8,0
014    BUD:DEC,N,8,0
015    <press RETURN>
INPUT DATA NOW? N
INDEX ON ACCT:SUB TO BUDGETS
```

R:base Series 4000

The R:base Series 4000 software package was used here because of its flexibility in interfacing to other systems and its ease of use. Only the four-digit sub-account is used.

```
R> DEFINE BUDGET
  Begin R:base Database Definition
D> ATTRIBUTES
D> ACCT-SUB TEXT 4 KEY
D> DESC TEXT 20
D> BUD-JAN INTEGER
D> BUD-FEB INTEGER
D> BUD-MAR INTEGER
D> BUD-APR INTEGER
D> BUD-MAY INTEGER
D> BUD-JUN INTEGER
D> BUD-JUL INTEGER
D> BUD-AUG INTEGER
D> BUD-SEP INTEGER
D> BUD-OCT INTEGER
D> BUD-NOV INTEGER
D> BUD-DEC INTEGER
D> RELATIONS
D> BUDGETS WITH ACCT-SUB DESC BUD-JAN BUD-FEB BUD-MAR BUD-APR +
D>   BUD-MAY BUD-JUN BUD-JUL BUD-AUG BUD-SEP BUD-OCT BUD-NOV +
D>   BUD-DEC
D> END
  End R:base Database Definition
R>
```

IV. File Formats.

There are two types of file formats that you should understand before continuing on at this point. They are:

DIF (Data Interchange Format)

DELIMITED FILES

These file types are recognized by a number of common software packages available on personal computers including Lotus 1-2-3, dBase II, and R:base Series 4000.

DIF (Data Interchange Format)

DIF files allow you to describe data that is arranged in columns (vectors) and rows (tuples). DIF files consist of two parts: the header and the data part. The header describes the data and the data part contains the actual values. Here is the layout of the DIF file we will be working with:

TABLE			
0,1			
""			
VECTORS			
0,0			
""			
TUPLES			
0,0			
""			
DATA			
0,0			
""			
-1,0		Beginning	
BOT		of Tuple	
1,0		Alphanumeric	Data Part
"1010"		data value #1	
1,0		Alphanumeric	V
"Salaries	"	data value #2	
0, 10000		Numeric	
V		data value #3	
0, 10000		Numeric	
V		data value #4	
0, 12000		Numeric	
V		data value #5	
0, 10000		Numeric	
V		data value #6	
0, 11000		Numeric	
V		data value #7	
0, 13000		Numeric	
V		data value #8	
0, 11000		Numeric	

V			data value #9
0, 11000			Numeric
V			data value #10
0, 14000			Numeric
V			data value #11
0, 12000			Numeric
V			data value #12
0, 12000			Numeric
V			data value #13
0, 14000			Numeric
V			data value #14
-1.0			
EOT			Next
1,0			Tuple
"2010"			
1,0			
"Staff Training"	"		
< etc. >			
0, 1200			
V			
0, 1200			
V			
-1,0			End of data
EOD			(after last record)

The header record does not need any modification. Some programs require values for the number of tuples and vectors, but from experience it seems that Lotus 1-2-3 and R:base Series 4000 get along fine without them.

Each tuple in the data part begins with:

```
-1,0
BOT
```

After that, each data value in the tuple contains two lines:

```
Number value field

String value field
```

The number value field contains two numbers separated by a comma. The first number determines whether the field is numeric (0) or a string value (1). The second number contains the numeric value (ignored for string values).

The string value field contains a string value. For numerics, the string value should be V (for value). For string values, the string enclosed in quotes should be used.

At the end of the file is an end of data indicator:

-1,0
EOD

More information on DIF files can be obtained by writing to:

DIF Clearinghouse
P. O. Box 527
Cambridge, MA 02139

DELIMITED FILES

Delimited files are much easier to understand and are used primarily by dBase II and R:base Series 4000. The format of a delimited file is simply the field values separated by commas. Strings are contained in quotes:

```
"1010","Salaries", 10000, 10000, 12000, 11000,...  
"2010","Staff Training", 800, 0, 800, 0,...  
"2020","Memberships", 0, 60, 0, 60,...  
<etc.>
```

V. Data Conversions.

1a. QUERY - REPORT (to a disc file) in DIF format

The report writer in QUERY will allow you to create a DIF file. One of the restrictions that caused some problems was the limit of 9 header records (the DIF file header has 12). To get around this, use short reports to produce the first three lines of the header, and then a big report to produce the other 9 and all of the data. The final EOD statements were also broken up in the same manner.

A file equation is needed to send the report output to disc:

```
:FILE QSLIST=DIFFILE,NEW:DEV=DISC;REC=-30,8,F,ASCII; &
:   DISC=10000;NOCCTL;SAVE
```

A small record size (30 characters) is used to save on the amount of time used to transfer the data from one computer to another.

You must create an XEQ file as follows:

```
DEF
BUDGET           <DATA-BASE>
SUPER            <PASSWORD>
1                <MODE>
ACCT-DETAIL      <DATA-SET>
                  <PROC-FILE>
                  <OUTPUT>
FIND ACCT-NUMBER >= 400000 AND ACCT-NUMBER <= 409999
OUT=LP           <find record and start output>
REPORT           <first report creates one header record>
LINES=0          <continuous page>
NOPAGE          <no page eject>
TF,"TABLE",5    <prints first header line>
END
REPORT           <second header record>
LINES=0
NOPAGE
TF,"0,1",3
END
REPORT           <third header record>
LINES=0
NOPAGE
TF,""",2
END
REPORT           <start of main report>
LINES=0
NOPAGE
S,ACCT-NUMBER    <sorting by account number>
RO,LOAD,ACCT-NUMBER <register 0 and edit 0 are used to
RO,SUB,"400000"  strip first two digits from account
EO,"9999"        number>
H1,"VECTORS",7  <H1 through H9 create header records
```

H2,"0,1",3 #4 to #12>
H3,"",2
H4,"TUPLES",6
H5,"0,0",3
H6,"",2
H7,"DATA",4
H8,"0,0",3
H9,"",2
D1,"-1,0",4 <start of data part>
D2,"BOT",3
D3,"1,0",3
D4,"",1
D4,RO,5,E0
D4,"",6
D5,"1,0",3
D6,"",1
D6,DESCRIPTION,21
D6,"",22
D7,"0,",2
D7,BUDGET-JAN,9
D8,"V",1
D9,"0,",2
D9,BUDGET-FEB,9
D10,"V",1
D11,"0,",2
D11,BUDGET-MAR,9
D12,"V",1
D13,"0,",2
D13,BUDGET-APR,9
D14,"V",1
D15,"0,",2
D15,BUDGET-MAY,9
D16,"V",1
D17,"0,",2
D17,BUDGET-JUN,9
D18,"V",1
D19,"0,",2
D19,BUDGET-JUL,9
D20,"V",1
D21,"0,",2
D21,BUDGET-AUG,9
D22,"V",1
D23,"0,",2
D23,BUDGET-SEP,9
D24,"V",1
D25,"0,",2
D25,BUDGET-OCT,9
D26,"V",1
D27,"0,",2
D27,BUDGET-NOV,9
D28,"V",1
D29,"0,",2
D29,BUDGET-DEC,9
D30,"V",1

```
TF,"-1,0",4          <first line of EOD>
END
REPORT
LINES=0
NOPAGE
TF,"EOD",3          <last line of the file>
END
EXIT                <EXIT or OUT=TERM will close the file>
```

Using QUERY, it is difficult to summarize data for a DIF file due to the limitations of the group and total statements.

After entering your file equation, use the following commands to run this routine:

```
:QUERY
> XEQ xeqfile
```

When transferring the DIF file to a personal computer, use the file extension .DIF.

1b. QUERY - REPORT (to a disc file) in DELIMITED format

The report writer in QUERY will allow you to create a DELIMITED file. In a delimited file, all of the data for a record must be contained on one line which can cause a problem since you can only have a record length of 132 characters in QUERY. You can get around this by making several passes into your personal computer's data base.

A file equation is needed to send the report device to disc:

```
:FILE QSLIST=DELFILE,NEW;DEV=DISC;REC=-132,19,F,ASCII; &
: DISC=10000;NOCCTL;SAVE
```

You must create an XEQ file as follows:

```
DEF
BUDGET                <DATA-BASE>
SUPER                 <PASSWORD>
1                     <MODE>
ACCT-DETAIL           <DATA-SET>
                     <PROC-FILE>
                     <OUTPUT>

FIND ACCT-NUMBER >= 400000 AND ACCT-NUMBER <= 409999
OUT=LP                <find record and start output>
REPORT                <begin report>
LINES=0               <continuous page>
NOPAGE                <no page eject>
S,ACCT-NUMBER         <sorting by account number>
RO,LOAD,ACCT-NUMBER  <register 0 and edit 0 are used to
RO,SUB,"400000"       strip first two digits from account
EO,"9999"             number>
D1,"",",",1           <begin definition of DELIMITED record>
D1,RO,5,E0
D1,"",",",",8
D1,DESCRIPTION,28
D1,"",",",30
D1,BUDGET-JAN,37
D1,"",",",38
D1,BUDGET-FEB,45
D1,"",",",46
D1,BUDGET-MAR,53
D1,"",",",54
D1,BUDGET-APR,61
D1,"",",",62
D1,BUDGET-MAY,69
D1,"",",",70
D1,BUDGET-JUN,77
D1,"",",",78
D1,BUDGET-JUL,85
D1,"",",",86
D1,BUDGET-AUG,93
D1,"",",",94
D1,BUDGET-SEP,101
D1,"",",",102
```

```
D1,BUDGET-OCT,109
D1,"",110
D1,BUDGET-NOV,117
D1,"",118
D1,BUDGET-DEC,125
END
EXIT
```

<EXIT of OUT=TERM will close the file>

In this example, another field would not have fit due to the 132 character limitation. Summary files can be built by using a sort and a total statement instead of the detail statement.

When transferring the DELIMITED file to a personal computer, use the file extension .TXT.

1c. QUERY - UPDATE ADD

In this method of input, an XEQ file is created that imitates an UPDATE ADD input sequence. The XEQ file must be constructed to "DEFINE" the data base, execute the UPDATE ADD command, then enter all of the fields in order. When all of the records have been entered, a "//" must be input to end the routine and an optional EXIT can be added.

To execute this XEQ file, use these commands:

```
:QUERY
> XEQ xeqfile
```

2a. Powerhouse - QUIZ - SET REPORT DEVICE DISC create DIF file

It is very easy to create reformatted disc files using QUIZ. QUIZ has much more flexibility than QUERY and allows you to summarize and calculate data very easily. The limitations are greater also. In the case of creating a DIF file, there does not seem to be any limitations here.

No file equations are needed, just a QUIZ report file:

ACCESS ACCT-DETAIL

```
DEFINE DEPT NUMERIC*2 = PARM PROMPT "ENTER DEPARTMENT NUMBER =>"
SELECT IF ACCT-DEPT = DEPT          <selection criteria>
SORT ON ACCT-SUB                    <sort on sub-account>
```

```
SET PAGE LENGTH 0                    <standard file setup>
SET PAGE WIDTH 30                    <record width 30 to
SET NOFORMFEED                       save time in data
SET NOHEAD                             transfer>
SET REPORT DEVICE DISC
SET REPORT SPACING 0
SET REPORT NAME DIFFILE
SET REPORT LIMIT 10000
```

```
INITIAL HEADING                      & <the INITIAL HEADING
'TABLE'                               SKIP & statement creates
'0,1'                                  SKIP & the header record>
',,,,',                                SKIP &
'VECTORS'                              SKIP &
'0,0'                                   SKIP &
',,,,',                                SKIP &
'TUPLES'                                SKIP &
'0,0'                                   SKIP &
',,,,',                                SKIP &
'DATA'                                  SKIP &
'0,0'                                   SKIP &
',,,,',                                SKIP &
```

```
REPORT                                & <the report statement
'-1,0'                                 SKIP & creates the data
'BOT'                                   SKIP & part>
'1,0'                                   SKIP &
',,,, ACCT-SUB ',,,,',                 SKIP &
'1,0'                                   SKIP &
',,,, DESCRIPTION ',,,,',             SKIP &
'0,' BUDGET-JAN                         SKIP &
'V'                                      SKIP &
'0,' BUDGET-FEB                         SKIP &
'V'                                      SKIP &
'0,' BUDGET-MAR                         SKIP &
'V'                                      SKIP &
'0,' BUDGET-APR                         SKIP &
'V'                                      SKIP &
```


'0,'	BUDGET-MAY	SKIP	&	
'v'		SKIP	&	
'0,'	BUDGET-JUN	SKIP	&	
'v'		SKIP	&	
'0,'	BUDGET-JUL	SKIP	&	
'v'		SKIP	&	
'0,'	BUDGET-AUG	SKIP	&	
'v'		SKIP	&	
'0,'	BUDGET-SEP	SKIP	&	
'v'		SKIP	&	
'0,'	BUDGET-OCT	SKIP	&	
'v'		SKIP	&	
'0,'	BUDGET-NOV	SKIP	&	
'v'		SKIP	&	
'0,'	BUDGET-DEC	SKIP	&	
'v'				
FINAL FOOTING			&	<the FINAL FOOTING
'-1,0'		SKIP	&	creates the EOD
'EOD'				statements>
GO				<execute the report>
EXIT				<exit QUIZ>

Data can be summarized by using a SORT, a REPORT statement with no parameters, and using FOOTING AT sort-key to create the output. To execute this routine, use the following commands:

```
:QUIZ
> USE repfile

or

:FILE QUIZUSE=repfile
:QUIZ
```

This particular report was set up so that it could be run over and over again for different departments. How automatic these routines can be set up for is up to you.

When transferring the DIF file to a personal computer, use the file extension .DIF.

2b. Powerhouse - QUIZ - SET REPORT DEVICE DISC create DELIMITED file

It is very easy to create DELIMITED files using QUIZ except for one limitation. You can only have 264 characters on a line of QUIZ output. But, for most tasks, this should give you enough room.

No file equations are needed, just a QUIZ report file:

ACCESS ACCT-DETAIL

```

DEFINE DEPT NUMERIC*2 = PARM PROMPT "ENTER DEPARTMENT NUMBER =>"
SELECT IF ACCT-DEPT = DEPT          <selection criteria>
SORT ON ACCT-SUB                    <sort on sub-account>

SET PAGE LENGTH 0                   <standard file setup>
SET PAGE WIDTH 132                  <record width is just
SET NOFORMFEED                      big enough to
SET NOHEAD                          handle the size of
SET REPORT DEVICE DISC              the record created>
SET REPORT SPACING 0
SET REPORT NAME DELFILE
SET REPORT LIMIT 10000

```

```

REPORT                               &          <the report statement
                                     &          creates the
ACCT-SUB                             '","'      &          records>
DESCRIPTION                          '","'      &
BUDGET-JAN                           '',''      &
BUDGET-FEB                           '',''      &
BUDGET-MAR                           '',''      &
BUDGET-APR                           '',''      &
BUDGET-MAY                           '',''      &
BUDGET-JUN                           '',''      &
BUDGET-JUL                           '',''      &
BUDGET-AUG                           '',''      &
BUDGET-SEP                           '',''      &
BUDGET-OCT                           '',''      &
BUDGET-NOV                           '',''      &
BUDGET-DEC                           '',''      &

GO                                     <execute the report>
EXIT                                   <exit quiz>

```

Data can be summarized by using a SORT, a REPORT statement with no parameters, and using FOOTING AT sort-key to create the output. To execute this routine, use the following commands:

```

:QUIZ

> USE refile

or

```

```
:FILE QUIZUSE=repfile  
:QUIZ
```

This particular report was set up so that it could be run over and over again for different departments. How automatic these routines can be set up for is up to you.

When transferring the DELIMITED file to a personal computer, use the file extension .TXT.

2c. Powerhouse - QDD - Define a file.

There are a number of ways that a file could be transferred to Powerhouse by defining a file in the dictionary and using it for input (using QTP). Use your imagination.

3a. Lotus 1-2-3 - .WKS to .DBF

Although this is an easy conversion to perform, you must have your file in the correct format for it to work correctly. 1-2-3 will build a data base file for you but you must supply the field names for it to be built properly. dBase II uses 10 character field names which can have embedded colons (:). You should keep this in mind.

The first row of information you transfer must contain the field names to be used in the data base file. The data must follow right under it. If the entire spreadsheet is not to be copied, you must name a data range before executing the translate program.

To execute the translation, follow these steps:

1. Enter the translate program
2. Select the .WKS to .DBF option
3. Select source disk
4. Select the file name
5. Select destination disk
6. Select entire worksheet or range
7. If range selected, input name of range
8. "Y" to execute translation
9. "Q" to quit

3b. Lotus 1-2-3 - .WKS to .DIF

Sometimes it is a good idea to reformat your data before executing this option. Simply retrieve your data file, strip out all of the headings, titles, and unwanted values, and save it under a different name.

To execute the translation, follow these steps:

1. Enter the translate program
2. Select the .WKS to .DIF option
3. Select source disk
4. Select the file name
5. Select destination disk
6. "Y" to execute translation
7. "Q" to quit

3c. Lotus 1-2-3 - .DBF to .WKS

This translation will bring over both data and headings into your spreadsheet. After executing this function, you may want to adjust your spreadsheet file by fixing left justification on string fields, formatting decimal points, adjusting column widths, and making room for headings.

To execute the translation, follow these steps:

1. Enter the translate program
2. Select the .DBF to .WKS option
3. Select source disk
4. Select the file name
5. Select destination disk
6. "Y" to execute translation
7. "Q" to quit

3d. Lotus 1-2-3 - .DIF to .WKS

After executing this function, you may want to adjust your spreadsheet file by fixing left justification on string fields, formatting decimal points, adjusting column widths, and making room for headings.

To execute the translation, follow these steps:

1. Enter the translate program
2. Select the .DIF to .WKS option
3. Select source disk
4. Select the file name
5. Select destination disk
6. "Y" to execute translation
7. "Q" to quit

4a. dBase II - APPEND FROM file DELIMITED

dBase II will accept a delimited file as input using this command. Your data base and any indexes must be set up and opened first. Here is a sequence of commands that will bring in a delimited file as input:

```
. USE BUDGETS INDEX BUDGETS  
. APPEND FROM delfile DELIMITED
```

4b. dBase II - COPY TO file DELIMITED

dBase II will create a delimited file as output using this command. Your data base and any indexes must be set up and opened first. Here is a sequence of commands that will create a delimited file:

```
. USE BUDGETS INDEX BUDGETS  
. COPY TO delfile DELIMITED
```

4c. dBase II - SET ALTERNATE TO file program (QUERY UPDATE ADD)

In this routine, dBase II is programmed to create a QUERY UPDATE ADD XEQ file. The statements here are very simple. Use an editor to create this .PRG file:

```

USE BUDGETS                <opens data base file>
INDEX ON ACCT:SUB TO BUDGETS <indexes file>
SET INDEX TO BUDGETS      <attaches the index>
SET ALTERNATE TO QUERYIN  <opens output file>
SET TALK OFF              <eliminates excess messages>
SET ALTERNATE ON          <directs output to file>
? "DEF"
? "BUDGET"                <DATA-BASE>
? "SUPER"                 <PASSWORD>
? "1"                     <MODE>
? "ACCT-DETAIL"           <DATA-SETS>
? " "                     <PROC-FILE>
? " "                     <OUTPUT>
? "ADD ACCT-DETAIL"       <UPDATE ADD command (abbrev.)>
DO WHILE .NOT. EOF        <loop until end of file>
  ? "40"+ACCT:SUB         <adds department back to number>
  ? DESC
  ? BUD:JAN
  ? BUD:FEB
  ? BUD:MAR
  ? BUD:APR
  ? BUD:MAY
  ? BUD:JUN
  ? BUD:JUL
  ? BUD:AUG
  ? BUD:SEP
  ? BUD:OCT
  ? BUD:NOV
  ? BUD:DEC
  SKIP                    <skip to next record>
ENDDO                     <end of loop>
? "/"                    <// for end of input>
? "EXIT"                  <EXIT from QUERY>
SET ALTERNATE OFF         <turn off output to file>
SET ALTERNATE TO         <close the file>
SET TALK ON               <reset TALK option>
RETURN                    <end of routine>

```

To execute this routine in dBase II, use these commands:

```
DO routine
```

5a. R:base Series 4000 - UNLOAD DATA FOR relation AS DIF

R:base Series 4000 can use DIF files for both input and output. In this example, a DIF file is created simply by entering some R:base commands:

```
R> OPEN BUDGET
R> OUTPUT A:BUDGET.DIF
R> UNLOAD DATA FOR BUDGETS AS DIF
R> OUTPUT TERMINAL
```

5b. R:base Series 4000 - LOAD relation FROM file AS DIF

In this example, a DIF file is loaded in simply by entering some R:base commands:

```
R> OPEN BUDGET
R> LOAD BUDGETS FROM A:BUDGET.DIF AS DIF
```

5c. R:base Series 4000 - UNLOAD DATA FOR relation AS ASCII

In this example, a DELIMITED file (ASCII file) is created simply by entering some R:base commands:

```
R> OPEN BUDGET
R> OUTPUT A:BUDGET.TXT
R> UNLOAD DATA FOR BUDGETS AS ASCII
R> OUTPUT TERMINAL
```

5d. R:base Series 4000 - LOAD relation FROM file AS ASCII

In this example, a DELIMITED file is loaded in simply by entering some R:base commands:

```
R> OPEN BUDGET
R> LOAD BUDGETS FROM A:BUDGET.TXT AS ASCII
```


5e. R:base Series 4000 - OUTPUT file & PRINT report (QUERY UPD ADD)

Using the REPORT function in R:base, you can create a report file that will build a QUERY UPDATE ADD XEQ file. After entering R:base and opening your data base, enter the report function and build a report that is formatted like this:

```
*****
H * DEF
H * BUDGET
H * SUPER
H * 1
H * ACCT-DETAIL
H *
H *
H * ADD ACCT-DETAIL
D * 41S E <ACCT-SUB>
D * S E <DESC>
D * S E <BUD-JAN>
D * S E <BUD-FEB>
D * S E <BUD-MAR>
D * S E <BUD-APR>
D * S E <BUD-MAY>
D * S E <BUD-JUN>
D * S E <BUD-JUL>
D * S E <BUD-AUG>
D * S E <BUD-SEP>
D * S E <BUD-OCT>
D * S E <BUD-NOV>
D * S E <BUD-DEC>
F * //
*****
```

To create the file, use these R:base commands:

```
R> OPEN BUDGET
R> OUTPUT A:QUERYIN.DAT
R> PRINT BUDGET <BUDGET is the name of the report>
R> OUTPUT TERMINAL
```

VI. Conclusion.

There are many more ways to migrate data between software than are mentioned here. This paper has only touched on a few and it is up to your creativity to find more.

Happy migrating!

