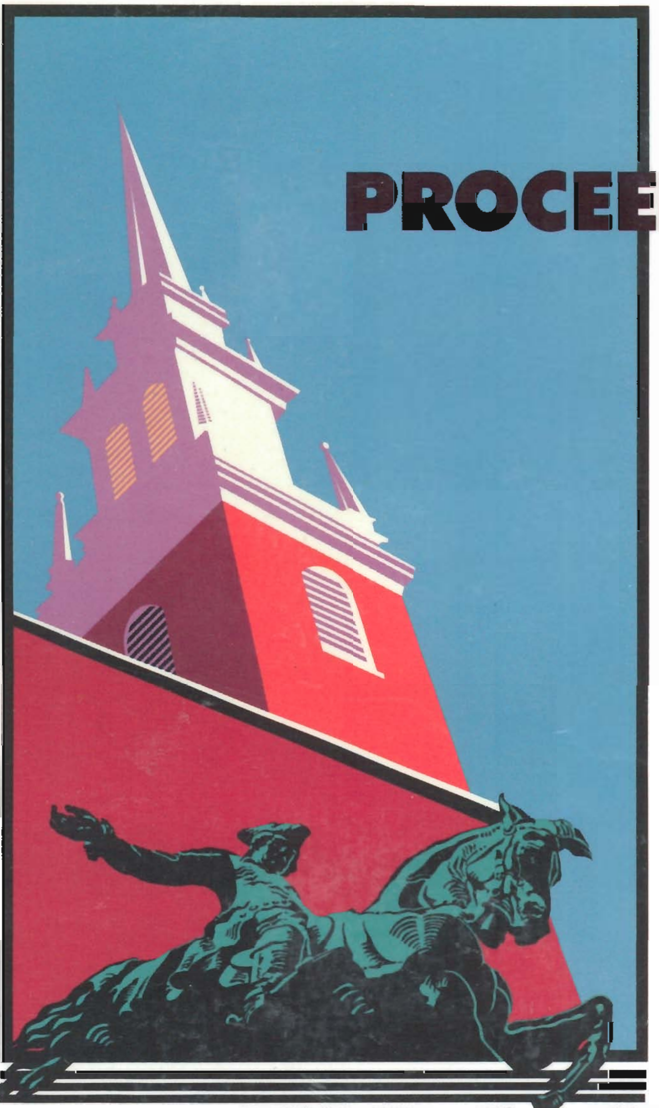
PROCEEDINGS
VOLUME 1

INTEREX
HP Users
CONFERENCE
AUGUST 20-23

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



BOSTON • 1990
INTEREX HP USERS CONFERENCE
AUGUST 20-23

Sponsored by **INTEREX** The International Association of Hewlett Packard Computer Users

INTEREX

The International Association of Hewlett-Packard Computer Users

PROCEEDINGS

of the

1990 INTEREX HP Users Conference

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

at

Boston, Massachusetts
August 20-23. 1990

Volume 1

David W. Haberman, Editor

3003 A Beginner's Guide to Programming for Maintenance and Portability
Roy Buzdor - Gateway Systems Corp.
3007 A Means to an End-System Performance is a Journey Rather Than a Destination
James Harvison, Charles Coleman - Technology Applications Group, Inc.
3009 A Programmatic Approach to Tape Loading
Paul Ferguson - Wall Street Concepts
3010 A Solution for Sales Management's Nightmares: An Omnidex Environment
Marie Gannon - United Liquors, Ltd.
3014 An Introduction to Object-Oriented Programming and Databases
Bruce Toback - OPT, Inc.
3015 And Now For Something Completely Different: International System Manager's Perspectives
Kim Barton, Wayne Laughton - Mackenzie Financial Corp.
3017 Backup Performance or How to Backup Faster!
Bryan Carroll - Hewlett-Packard Co.
3018 Basic Resource Accounting
Bruce Senn - Union College
3019 Beyond SYSGEN/VOLUTIL: Volume Management on MPE/XL
Sandra Iwamoto - Hewlett-Packard Co.
3020 Bottlenecks: How to Recognize and Resolve Them
Ralph Kotoski - Hewlett-Packard Co.
3025 Analysis-Design, CASE and OOP a "State of the Art" Assessment!
Robert Mattson - WIDCO
3028 Cobol And View Today: Intergration of Cobol 85 and Hi-Li
Rick Hoover - CIV Software
3029 Complex Data Structure, Omnidex and Relational Retrieval
Kimball Everingham - SRI International
3037 Create Your Own On-Line Help System
Allen Burns - Rutgers-The State Univ of NJ
3039 Databases Therapy: a Practitioner's Advice for the 90's
Alfredo Rego - Adager
3041 Designing User Inquiry Screens for Keyed Access
Paul Edwards - Bradmark Computer Systems
3058 Fundamentals of IMAGE: Schema & Structure for the User
Joseph Speier - Speier Assoc.
3062 How to Improve All Your Printed Reports Using Laserjets
Jason Kent - Proactive Systems Ltd.
3068 Improving Program Reusability
Larry Van Sickle - Tymlabs Corporation
3069 Improving Remote Database Access Performance
Roger Lawson - Proactive Systems, Inc.
3072 Integer Keys: The Final Chapter
Fred White - Adager
3074 JCWs: An Underutilized MPE Feature
David Largent - N.G.Gilbert Corp.
3075 KSAM 101
John Alleyn-Day - Alleyn Day International
3076 KSAM - A Different Approach
D.W. Conger - Amarillo College

3078 Looping Through Image Datasets with Protos
Brian Becker - NOMC, Inc.
3079 Advanced CI Programming - The 2.1 Story
Scott Cressler, et al - Hewlett-Packard Co.
3082 Maintaining the Integrity of Your Software
Alan Padula - Hewlett-Packard Co.
3091 No Time, No Money, and the System Needs a Complete Rewrite
John Fusek, Serena Fusek - Virginia Int'l. Terminals
3092 Omnidex/Imsam as a Database Design Tool
Katherine Newcombe - Systems Consultant
3093 One Company's Successful Implementation of Software Guidelines
Daniel Marcotte - United Liquors Ltd.
3094 Look, No Secondaries Optimum Master Set Capacities - From Art to a Science
David Wiseman - Proactive Systems, Inc.
3099 Performance in Relational Databases
Eric Savage - Dynamic Information Systems Corp.
3107 QUIZ/QTP Performance: Problem Areas
David Robinson - PowerSpec International
3108 Real World Performance Monitoring
Robert Lund - Robert Lund & Associates
3110 SQL and the Relational Data Model for the IMAGE Database
Mark da Silva - Univ of Hawaii Cancer Research
3111 Securing Your HP 3000
Dennis King, Mariette Araya - Hewlett-Packard Co.
3112 Eight Unknown Ways to Obtain SM Capability in Under Sixty Seconds
Carl Melclior - Independent Contractor's Group
3113 Software Creation with a 4GL Language: Lessons Learned
Gene Harmon - AH Computer Services, Inc.
3114 Stop Babysitting Your HP 3000
Craig Conrad - McDonnell Douglas Space
3115 Stop! Don't Pick up the Phone! Order Electronically
Kathy Spanel - GBS Consultants, Inc.
3116 Structured Online Program
Terry Warns - WL Software
3118 System Managers Toolbox
Isaac Blake - City of Tempe
3119 System Performance Through Queue Tuning
Warren Bunnell - ORBIT Software (USA), Inc.
3120 System Performance/Management For Technical/Nontechnical/Personnel
Dave Jennings - Facer-Sys. Performance Div.
3121 Table Your Database
Mike Mitchell - Hewlett-Packard Co.
3122 The Automation of Software Testing
Susan Maxwell - Hewlett-Packard Co.
3123 The Biggest Computer Security Threat
Vladimir Volokh - Vesoft
3128 TurboImage vs. HPSQL, A (Performance) Comparison
Charles Sullivan - Runningmate

3130 Procedure Exits on MPE XL
Rajesh Desai - Hewlett-Packard Co.

3132 Using MPE/XL Debugging Tools
Bruce Toback - OPT, Inc.

3134 When to Use IMSAM/When to use Omindex/ (When to Use IMAGE)
Suzanne Harmon - AH Computer Services, Inc.

3137 Your Own Private Program Library
Gerrit Altena - A&B Automatisering

3141 Making Other People's Programs Do What They Were Never Intended To Do - The "Hook under the Hood"
Eugene Volokh - Vesoft

3142 Automated Testing: One Company's Experience
Eugene Volokh - Vesoft

3184 POSIX and X Windows on MPE XL
Scott Cressler, Ryk Cressler - Hewlett-Packard Co.

3186 Analyzing MPE XL Performance: What is "Normal"?
Doug Grumann, Marie Weston - Hewlett-Packard Co.

3188 Performance Contracts: Service Level Agreements in the Real World
Rex Backman - Hewlett-Packard Co.

3189 Object Oriented COBOL: A New Initiative From HP
Megan Adams, Dmitry Lenkov - Hewlett-Packard Co.

3190 Improving the Efficiency and Accuracy of Your Information Capture Through Automated Data Collection
Raymond Agrusti - Eagle Consulting + Development Corp.

3196 Transaction Logfiles - Their Uses
David Wiseman - Proactive Systems, Inc.

3201 Evaluating the Business Impact and Risks Associated with a Disaster
Gary Fletcher - Hewlett-Packard Co.

3202 Comparing MPE V and MPE XL Customer Workloads
Julie McClung - Hewlett-Packard Co.

3203 The HP 3000 Open Systems Environment
Janet Garcia - Hewlett-Packard Co.

3205 HP-FL A Performance Growth For MPE XL Systems
Melody Armstrong - Hewlett-Packard Co.

3206 Design, Installation and Maintenance of Cabling Infrastructures
Charles Breed - Hewlett-Packard Co.

3207 The Truth About Developing Co-operative Applications
Robert Dey - Hewlett-Packard Australia Limited

3213 Quality Software -- Walking It Through
Lisa Burns - Hewlett-Packard Co.

3214 What Does This Benchmark Mean To You?
Frank Rowand - Hewlett-Packard Co.

3222 Relational vs. Network - The Dilemma of the '90's
Shelley Meiklejohn - Price Waterhouse

3230 Effective Systems Development Using 4GL
Douglas Colter - Infocentre Corporation

3707 Performance Measurement Of Multiple CPU's: A Tale of 200+ Systems
Mark Willis - American Airlines

4001 Integrated Image Applications
Malcolm Sharratt - Hewlett-Packard Co.

4005 Designing Multi-Vendor Lans That Work
Robert Yori - Hewlett-Packard Co.
4006 A Cooperative Processing Strategy for the 90's
Doug Walker - Walker Richer & Quinn
4016 Background Operation of Electronic Mail Within the HP NewWave Environment
Dave Morris - Hewlett-Packard Co.
4021 Buddy, Can You Spare Some Time?
George Blessing - City of Pasadena
4033 Cooperative Processing Today with PC's, Macintosh's and, HP Minicomputers
Ron Arons - Sybase Inc.
4035 Costing PC Integration..Is it worth it?
Jim Wowchuk - Vanguard Computer Services
4036 Create A Data Dictionary on PC
Phil Nguyen - Lockheed
4038 Data Synchronization
Lawrence Kingsbury - Gateway Systems Corporation
4040 Designing Distributive Systems for PC Networks
Paul Mole - PMC, Inc.
4043 Developing Distributed Applications using LM/X
Grant Sidwall - Hewlett-Packard Co.
4044 Developing Portable Applications in the Powerhouse PC- A Case Review
David Robinson - PowerSpec International
4045 Developing a NewWave User Interface for MPE and HP-UX Applications
Mike Broadway - Hewlett-Packard Co.
4049 Distributed Application Processing
Charles Warzecha - Gateway Systems Corporation
4057 File Serving with the HP 3000
Bud Booth - AA Software Development
4066 Implementing NewWave - A Case Study
Don Lewis - The Apex Group
4073 The HP 3386 and HP 3286 Integrating the HP 3000 with the PC The next Generation of Computer Systems
Leroy Friesenhahn - Blanket Resources
4084 Microcrimes and Megabucks
Dr. Sanford Sherizen - Data Security Systems, Inc.
4088 Network Application Development- Integrating PC's and HP 3000's
Paul Thomas, Brent Ehrman - Crowe,Chizek and Co.
4095 Organizing Information Through Technology
Barry Prokop - Fairfax County Public School
4096 PC Programming for Minicomputer Specialists
Russell Walker - CompuMed Inc.
4097 PC/HP 3000 Integration - Who Does What
Craig Fransen - Fransen/King Ltd.
4129 Understanding Printer Sharing Solutions
Thomas Durr - Wilco Communications
4131 Using Information Access to Get SQL Data to PC Users
Robert Ross - Hewlett-Packard Co.
4133 What's Behind Those Windows
John Stachowiak - NW Tri County Inter. Unit

4135 Writing Machine Independent C Programs on the HP 3000
Joseph Wm Berry - Dataform Corp.
4187 Desktop Publishing for the 1990's: What's New and What's Next
David Swinkin - Hewlett-Packard Co.
4229 A Developer's Perspective of NewWave
Scott Marshall - IISI
4231 A Client/Server Model (Sales/Lead Tracking and Management)
Bob Hoover - Infocentre Corporation
5023 Building A Software & System Support Team
Don Jones - INLEX Inc.
5024 Case Studies of MPE/XL Migrations
Paul Thomas, Brent Ehrman - Crowe,Chizek and Co.
5031 Controlling the Development Process
Thomas Eineker - PMC,Inc.
5034 Cost Justifying a New Technology
Richard Phillips - OPT (Office Products Technology)
5046 Disaster Planning-Lessons from the 1989 California Earthquake
Bud Beamguard - Syntex Research
5051 EDI - Change or Die! - A Survival Manual
Jim Himmelreich - Graco Childrens Products, Inc.
5052 EDI...A Programmer's Perspective
Joel Kushner, Donna Clark - Merit Systems,Inc.
5054 Executive Information Systems-Realizing the Potential
Mark Symonds - IISI
5063 How to Win at Your Next EDP Audit
Betsy Leight - Operations Control Systems
5067 Implementing Sales Force Automation Systems
Mark Shirman - Innovative Information Systems, Inc.
5070 Improving Software Quality
Robert Green - Robelle Consulting Ltd.
5080 Simplifying Migration with Standards: A Case Study of Effective Team Management
Dianne Willems - Hewlett-Packard Co.
5081 Maintaining a Quality Staff
Forrest Ford - Forrest Ford Consultants, Inc.
5085 Migrating Compatibility Mode Systems to Native Mode
Kevin Darling - The Gap, Inc.
5104 Project Management-Too Simple to Believe!
David Rubinstein - Innovative Information Systems
5124 The Greeting Behavior of the North American Homo Sapiens
Louis R. Mills - Bio-Rad Laboratories
5127 'Tinker-Ware:' Maximizing Your Investment In Application Tools
Leonard Block - The Apex Group
5182 Leasing - A Strategy to Reduce Costs and Manage the Future
Paul Perricone - Hewlett-Packard Co.
5191 Getting People in Your Company to use End-User Computing Tools
Kathy McKittrick - McKittrick Associates
5192 Information Management and Competitive Edge
James Leverentz - Off-Site, Inc

5193 45 Training Tips: A Tip A Minute
Tom Elliott - Distribution Resources Co.
5194 Education:In-House, Outhouse or CBT?
James Davis - Cognos Corp.
5195 Sales Force Automation Today
Michael Barry - Gateway Systems Corp.
5197 EDI in the Real World
Wayne Swafford - Computer Management Solutions
5198 Don't Be Cruel to a Heart That's True
Diane Amos - Amos & Associates
5199 Index Technology versus Relational A User's Dilemma
Daniel Filkins - Dynamic Information Systems Corp.
5200 Information Management-The Corporate Weapon of the 90's
Robert Andrews, Terry OBrien - Dynamic Information Systems Corp.
5209 Interactive Computer Graphics in Business
Jerry Kopan - Hewlett-Packard Co.
5210 Logic Programming and Expert Systems Usage Within Hewlett-Packard
Dave Elliott - Hewlett-Packard Co.
5211 Business in the 90's: Reducing Time to Action
Steve Jeffery - Hewlett-Packard Co.
5215 Fostering Creativity in a Technological Environment
Kent Hanson - Hewlett-Packard Co.
5216 Network Management Today: Fact or Fiction?
Carol Crall - Hewlett-Packard Co.
5218 Information Management Technologies Into the 1990's
Orland Larson - Hewlett-Packard Co.
5220 Stress Management Made (Relatively) Easy
Robert M. Gignac - C & M Products Limited
5234 System Management and Operations Automation
Linda Hazzan, Marge Vaillancourt - Carolian System International, Inc.
6030 Controlling External Devices Through Terminal Ports
Grant Parry - Message Transport Systems
6047 Disk Arrays - Mass Storage of the Future?
Ed Pavlinik - Hewlett-Packard Co.
6048 Disk Recording Technology - From DC to Light
Michael Rusnack - Hewlett-Packard Co.
6065 Image Processing: How should it be Utilized
Brent Ehrman, Paul Thomas - Crowe, Chizek and Company
6105 Protecting Your Data from System Failures
Gilles Schipper - G. Schipper & Associates Inc.
6125 The Technology of Data Compression and its Benefits to HP 3000 Users
David Wiseman, Bill Miller - Proactive Systems, Inc.
6126 The Increasing Number of Applications for HP's DDS Format Tape Drives
Debbie Beech - Hewlett-Packard
6138 Optical Storage Benefits, Applications, and Solutions
John Szlendak, Anne Gleason - Hewlett-Packard Co.
6139 Mass Storage Technology Positioning
Jon Schiedel, Connie Doster - Hewlett-Packard Co.

6185 Performance Predictions Techniques
Tim Twietmeyer, Dan Sternadel - Hewlett-Packard Co.
6212 C++: A New Software Development Methodology
Dmitry Lenkov, Michael Beckmann - Hewlett-Packard Co
6223 Optical Disk Technology: Past, Present and Future
Husni Sayed, Deborah Cobb - IEM, Inc.
7004 A Case Study for Networking the Production, The Administration and The Sales
Rainer Schulze - Hewlett-Packard Co.
7011 A User's View of HP/VAX Connectivity
Wayne McKinney - Lockheed Engineering
7012 ATP Ports Solve Telex Operator Problem
D.R. Kambey - Pt Arun Natural Gas Liquefaction Co.
7050 Do You Need A Fiber Optic Local Area Network
Jay Swearingen - The Apex Group
7055 Experience with Distributed X.25 Multivendor Networking
Alan MacIntosh - Hewlett-Packard Co.
7056 Experience with Networking-Moving to x.25
John Howitt - ECC Building Products
7059 Growing Your Lan into a Wan-Lan
Michael Russell - Hewlett-Packard Co.
7060 Creating A Global Enterprise Lan
Charles Breed, Brice Clark - Hewlett-Packard Co.
7061 How Can Networking On MPE XL Systems Help You Meet Your Business Needs?
Gregg Kornfield - Hewlett-Packard Co.
7064 How to be a 'Smart' User of a 'Dumb' Terminal
Michael Hornsby - Beechglen Development, Inc.
7083 Matching Network Servers to User Needs
John Green - Hewlett-Packard Co.
7086 Case Study: Migrating DS to NS, MPE V to MPE/XL and X.25 to ThinLAN"
Mary Fleischman - Washington Community College
7087 Migration From the Data Terminal Controller
James Cohen - Mecca Leisure Group Ltd.
7089 Networking in a Multi-Platform Environment
Mike Eckstein - The Gap Inc.
7090 Never Run Another Cable Again
Jay Swearingen - The Apex Group
7101 Problem Isolation Techniques for TCP/IP Networks
Mike Pozzi - Hewlett-Packard Co.
7103 Program to Program Communication - HP and IBM
Allan Gregory - EI DuPont de Nemours, Inc.
7109 Remote Process/Job Scheduling using NETIPC
Garry Osborne, Michael Mazza - Brown & Williamson Tobacco
7117 Survival Techniques For a 'No-Fault' HP 3000
J.G. Page, Shanda Snyder - Union Camp Corp.
7136 X.25-The Federal-Mogul Story
Tim Cahoon - Federal-Mogul Corporation
7143 Storage: File Servers in a Client/Server Computing Environment
Gene Jung, Jean Tang - Hewlett-Packard Co.

7145 Performance Management in the Distributed Computing Environment
Tony Engberg - Hewlett-Packard Co.
7146 Building a Network Management Solution
Steve Richard - Hewlett-Packard Co.
7147 Understanding The Bridge and Router
John Vega - Hewlett-Packard
7148 Understanding Basic Cabling, Sitewire, and Starlan
Charles Wirl - Wilco Communications Inc.
7204 ARPA Services For MPE XL
Deborah Campbell - HP-Business Network Divisions
7233 Migrating To NS - How and Why
Ann Smith, Steve Cole - Hewlett-Packard
T174 Tutorial Managing HP 3000 Performance
Brian Nogar - Hewlett-Packard Co.

Adams, Megan, Lenkov, Dmitry	Object Oriented COBOL: A New Initiative From HP
3189, Hewlett-Packard Co.	
Agrusti, Raymond	Improving the Efficiency and Accuracy of Your Information Capture Through
3190, Eagle Consulting + Development Corp.	Automated Data Collection
Alleyn-Day, John	KSAM 101
3075, Alleyn Day International	
Altena, Gerrit	Your Own Private Program Library
3137, A&B Automatisering	
Amos, Diane	Don't Be Cruel to a Heart That's True
5198, Amos & Associates	
Andrews, Robert, OBrien, Terry	Information Management-The Corporate Weapon of the 90's
5200, Dynamic Information Systems Corp.	
Armstrong, Melody	HP-FL A Performance Growth For MPE XL Systems
3205, Hewlett-Packard Co.	
Arons, Ron	Cooperative Processing Today with PC's, Macintosh's and, HP Minicomputers
4033, Sybase Inc.	
Backman, Rex	Performance Contracts: Service Level Agreements in the Real World
3188, Hewlett-Packard Co.	
Barry, Michael	Sales Force Automation Today
5195, Gateway Systems Corp.	
Barton, Kim, Laughton, Wayne	And Now For Something Completely Different: International System
3015, Mackenzie Financial Corp.	Manager's Perspectives
Beamguard, Bud	Disaster Planning-Lessons from the 1989 California Earthquake
5046, Syntex Research	
Becker, Brian	Looping Through Image Datasets with Protos
3078, NOMC, Inc.	
Beech, Debbie	The Increasing Number of Applications for HP's DDS Format Tape Drives
6126, Hewlett-Packard	
Berry, Joseph Wm	Writing Machine Independent C Programs on the HP 3000
4135, Dataform Corp.	
Blake, Isaac	System Managers Toolbox
3118, City of Tempe	
Blessing, George	Buddy, Can You Spare Some Time?
4021, City of Pasadena	
Block, Leonard	'Tinker-Ware:' Maximizing Your Investment In Application Tools
5127, The Apex Group	
Booth, Bud	File Serving with the HP 3000
4057, AA Software Development	
Breed, Charles, Clark, Brice	Creating A Global Enterprise Lan
7060, Hewlett-Packard Co.	
Breed, Charles	Design, Installation and Maintenance of Cabling Infrastructures
3206, Hewlett-Packard Co.	
Broadway, Mike	Developing a NewWave User Interface for MPE and HP-UX Applications
4045, Hewlett-Packard Co.	
Bunnell, Warren	System Performance Through Queue Tuning
3119, ORBIT Software (USA), Inc.	
Burns, Allen	Create Your Own On-Line Help System
3037, Rutgers-The State Univ of NJ	

Index by Author

Burns, Lisa 3213, Hewlett-Packard Co.	Quality Software -- Walking It Through
Buzdor, Roy 3003, Gateway Systems Corp.	A Beginner's Guide to Programming for Maintenance and Portability
Cahoon, Tim 7136, Federal-Mogul Corporation	X.25-The Federal-Mogul Story
Campbell, Deborah 7204, HP-Business Network Divisions	ARPA Services For MPE XL
Carroll, Bryan 3017, Hewlett-Packard Co.	Backup Performance or How to Backup Faster!
Cohen, James 7087, Mecca Leisure Group Ltd.	Migration From the Data Terminal Controller
Colter, Douglas 3230, Infocentre Corporation	Effective Systems Development Using 4GL
Conger, D.W. 3076, Amarillo College	KSAM - A Different Approach
Conrad, Craig 3114, McDonnell Douglas Space	Stop Babysitting Your HP 3000
Crall, Carol 5216, Hewlett-Packard Co.	Network Management Today: Fact or Fiction?
Cressler, Scott, Cressler, Ryk 3184, Hewlett-Packard Co.	POSIX and X Windows on MPE XL
Cressler, Scott, et al 3079, Hewlett-Packard Co.	Advanced CI Programming - The 2.1 Story
Darling, Kevin 5085, The Gap, Inc.	Migrating Compatibility Mode Systems to Native Mode
Davis, James 5194, Cognos Corp.	Education:In-House, Outhouse or CBT?
Desai, Rajesh 3130, Hewlett-Packard Co.	Procedure Exits on MPE XL
Dey, Robert 3207, Hewlett-Packard Australia Limited	The Truth About Developing Co-operative Applications
Durr, Thomas 4129, Wilco Communications	Understanding Printer Sharing Solutions
Eckstein, Mike 7089, The Gap Inc.	Networking in a Multi-Platform Environment
Edwards, Paul 3041, Bradmark Computer Systems	Designing User Inquiry Screens for Keyed Access
Ehrman, Brent, Thomas, Paul 6065, Crowe, Chizek and Company	Image Processing: How should it be Utilized
Eineker, Thomas 5031, PMC,Inc.	Controlling the Development Process
Elliott, Dave 5210, Hewlett-Packard Co.	Logic Programming and Expert Systems Usage Within Hewlett-Packard
Elliott, Tom 5193, Distribution Resources Co.	45 Training Tips: A Tip A Minute
Engberg, Tony 7145, Hewlett-Packard Co.	Performance Management in the Distributed Computing Environment

Everingham, Kimball	3029, SRI International	Complex Data Structure, Omnidex and Relational Retrieval
Ferguson, Paul	3009, Wall Street Concepts	A Programmatic Approach to Tape Loading
Filkins, Daniel	5199, Dynamic Information Systems Corp.	Index Technology versus Relational A User's Dilemma
Fleischman, Mary	7086, Washington Community College	Case Study: Migrating DS to NS, MPE V to MPE/XL and X.25 to ThinLAN*
Fletcher, Gary	3201, Hewlett-Packard Co.	Evaluating the Business Impact and Risks Associated with a Disaster
Ford, Forrest	5081, Forrest Ford Consultants, Inc.	Maintaining a Quality Staff
Fransen, Craig	4097, Fransen/King Ltd.	PC/HP 3000 Integration - Who Does What
Friesenhahn, Leroy	4073, Blanket Resources	The HP 3386 and HP 3286 Integrating the HP 3000 with the PC The next Generation of Computer Systems
Fusek, John, Fusek, Serena	3091, Virginia Int'l. Terminals	No Time, No Money, and the System Needs a Complete Rewrite
Gannon, Marie	3010, United Liquors, Ltd.	A Solution for Sales Management's Nightmares: An Omnidex Environment
Garcia, Janet	3203, Hewlett-Packard Co.	The HP 3000 Open Systems Environment
Gignac, Robert M.	5220, C & M Products Limited	Stress Management Made (Relatively) Easy
Green, John	7083, Hewlett-Packard Co.	Matching Network Servers to User Needs
Green, Robert	5070, Robelle Consulting Ltd.	Improving Software Quality
Gregory, Allan	7103, EI DuPont de Nemours, Inc.	Program to Program Communication - HP and IBM
Grumann, Doug, Weston, Marie	3186, Hewlett-Packard Co.	Analyzing MPE XL Performance: What is "Normal"?
Hanson, Kent	5215, Hewlett-Packard Co.	Fostering Creativity in a Technological Environment
Harmon, Gene	3113, AH Computer Services, Inc.	Software Creation with a 4GL Language: Lessons Learned
Harmon, Suzanne	3134, AH Computer Services, Inc.	When to Use IMSAM/When to use Omindex/ (When to Use IMAGE)
Harvison, James, Coleman, Charles	3007, Technology Applications Group, Inc.	Means to an End-System Performance is a Journey Rather Than a Destination
Hazzan, Linda, Vaillancourt, Marge	5234, Carolian System International, Inc.	System Management and Operations Automation
Himmelreich, Jim	5051, Graco Childrens Products, Inc.	EDI - Change or Die! - A Survival Manual
Hoover, Bob	4231, Infocentre Corporation	A Client/Server Model (Sales/Lead Tracking and Management)
Hoover, Rick	3028, CIV Software	Cobol And View Today: Intergration of Cobol 85 and Hi-Li

Hornsby, Michael	7064, Beechglen Development, Inc.	How to be a 'Smart' User of a 'Dumb' Terminal
Howitt, John	7056, ECC Building Products	Experience with Networking-Moving to x.25
Iwamoto, Sandra	3019, Hewlett-Packard Co.	Beyond SYSGEN/VOLUTIL: Volume Management on MPE/XL
Jeffery, Steve	5211, Hewlett-Packard Co.	Business in the 90's: Reducing Time to Action
Jennings, Dave	3120, Facer-Sys. Performance Div.	System Performance/Management For Technical/Nontechnical/Personnel
Jones, Don	5023, INLEX Inc.	Building A Software & System Support Team
Jung, Gene, Tang, Jean	7143, Hewlett-Packard Co.	Storage: File Servers in a Client/Server Computing Environment
Kambey, D.R.	7012, Pt Arun Natural Gas Liquefaction Co.	ATP Ports Solve Telex Operator Problem
Kent, Jason	3062, Proactive Systems Ltd.	How to Improve All Your Printed Reports Using Laserjets
King, Dennis, Araya, Mariette	3111, Hewlett-Packard Co.	Securing Your HP 3000
Kingsbury, Lawrence	4038, Gateway Systems Corporation	Data Synchronization
Kopan, Jerry	5209, Hewlett-Packard Co.	Interactive Computer Graphics in Business
Kornfield, Gregg	7061, Hewlett-Packard Co.	How Can Networking On MPE XL Systems Help You Meet Your Business Needs?
Kotoski, Ralph	3020, Hewlett-Packard Co.	Bottlenecks: How to Recognize and Resolve Them
Kushner, Joel, Clark, Donna	5052, Merit Systems,Inc.	EDI...A Programmer's Perspective
Largent, David	3074, N.G.Gilbert Corp.	JCWs: An Underutilized MPE Feature
Larson, Orland	5218, Hewlett-Packard Co.	Information Management Technologies Into the 1990's
Lawson, Roger	3069, Proactive Systems, Inc.	Improving Remote Database Access Performance
Leight, Betsy	5063, Operations Control Systems	How to Win at Your Next EDP Audit
Lenkov, Dmitry, Beckmann, Michael	6212, Hewlett-Packard Co	C++: A New Software Development Methodology
Leverentz, James	5192, Off-Site, Inc	Information Management and Competitive Edge
Lewis, Don	4066, The Apex Group	Implementing NewWave - A Case Study
Lund, Robert	3108, Robert Lund & Associates	Real World Performance Monitoring
MacIntosh, Alan	7055, Hewlett-Packard Co.	Experience with Distributed X.25 Multivendor Networking

Marcotte, Daniel 3093, United Liquors Ltd.	One Company's Successful Implementation of Software Guidelines
Marshall, Scott 4229, IISI	A Developer's Perspective of NewWave
Mattson, Robert 3025, WIDCO	Analysis-Design, CASE and OOP a "State of the Art" Assessment!
Maxwell, Susan 3122, Hewlett-Packard Co.	The Automation of Software Testing
McClung, Julie 3202, Hewlett-Packard Co.	Comparing MPE V and MPE XL Customer Workloads
McKinney, Wayne 7011, Lockheed Engineering	A User's View of HP/VAX Connectivity
McKittrick, Kathy 5191, McKittrick Associates	Getting People in Your Company to use End-User Computing Tools
Meiklejohn, Shelley 3222, Price Waterhouse	Relational vs. Network - The Dilemma of the '90's
Melclior, Carl 3112, Independent Contractor's Group	Eight Unknown Ways to Obtain SM Capability in Under Sixty Seconds
Mills, Louis R. 5124, Bio-Rad Laboratories	The Greeting Behavior of the North American Homo Sapiens
Mitchell, Mike 3121, Hewlett-Packard Co.	Table Your Database
Mole, Paul 4040, PMC, Inc.	Designing Distributive Systems for PC Networks
Morris, Dave 4016, Hewlett-Packard Co.	Background Operation of Electronic Mail Within the HP NewWave Environment
Newcombe, Katherine 3092, Systems Consultant	Omnidex/Imsam as a Database Design Tool
Nguyen, Phil 4036, Lockheed	Create A Data Dictionary on PC
Nogar, Brian T174, Hewlett-Packard Co.	Tutorial Managing HP 3000 Performance
Osborne, Garry, Mazza, Michael 7109, Brown & Williamson Tobacco	Remote Process/Job Scheduling using NETIPC
Padula, Alan 3082, Hewlett-Packard Co.	Maintaining the Integrity of Your Software
Page, J.G., Snyder, Shanda 7117, Union Camp Corp.	Survival Techniques For a 'No-Fault' HP 3000
Parry, Grant 6030, Message Transport Systems	Controlling External Devices Through Terminal Ports
Pavlinik, Ed 6047, Hewlett-Packard Co.	Disk Arrays - Mass Storage of the Future?
Perricone, Paul 5182, Hewlett-Packard Co.	Leasing - A Strategy to Reduce Costs and Manage the Future
Phillips, Richard 5034, OPT (Office Products Technology)	Cost Justifying a New Technology
Pozzi, Mike 7101, Hewlett-Packard Co.	Problem Isolation Techniques for TCP/IP Networks

Prokop, Barry	4095, Fairfax County Public School	Organizing Information Through Technology
Rego, Alfredo	3039, Adager	Databases Therapy: a Practitioner's Advice for the 90's
Richard, Steve	7146, Hewlett-Packard Co.	Building a Network Management Solution
Robinson, David	4044, PowerSpec International	Developing Portable Applications in the Powerhouse PC- A Case Review
Robinson, David	3107, PowerSpec International	QUIZ/QTP Performance: Problem Areas
Ross, Robert	4131, Hewlett-Packard Co.	Using Information Access to Get SQL Data to PC Users
Rowand, Frank	3214, Hewlett-Packard Co.	What Does This Benchmark Mean To You?
Rubinstein, David	5104, Innovative Information Systems	Project Management-Too Simple to Believe!
Rusnack, Michael	6048, Hewlett-Packard Co.	Disk Recording Technology - From DC to Light
Russell, Michael	7059, Hewlett-Packard Co.	Growing Your Lan into a Wan-Lan
Savage, Eric	3099, Dynamic Information Systems Corp.	Performance in Relational Databases
Sayed, Husni, Cobb, Deborah	6223, IEM, Inc.	Optical Disk Technology: Past, Present and Future
Schiedel, Jon, Doster, Connie	6139, Hewlett-Packard Co.	Mass Storage Technology Positioning
Schipper, Gilles	6105, G. Schipper & Associates Inc.	Protecting Your Data from System Failures
Schulze, Rainer	7004, Hewlett-Packard Co.	A Case Study for Networking the Production, The Administration and The Sales
Senn, Bruce	3018, Union College	Basic Resource Accounting
Sharratt, Malcolm	4001, Hewlett-Packard Co.	Integrated Image Applications
Sherizen, Dr. Sanford	4084, Data Security Systems, Inc.	Microcrimes and Megabucks
Shirman, Mark	5067, Innovative Information Systems, Inc.	Implementing Sales Force Automation Systems
Sidwall, Grant	4043, Hewlett-Packard Co.	Developing Distributed Applications using LM/X
Smith, Ann, Cole, Steve	7233, Hewlett-Packard	Migrating To NS - How and Why
Spanel, Kathy	3115, GBS Consultants, Inc.	Stop! Don't Pick up the Phone! Order Electronically
Speier, Joseph	3058, Speier Assoc.	Fundamentals of IMAGE: Schema & Structure for the User
Stachowiak, John	4133, NW Tri County Inter. Unit	What's Behind Those Windows

Sullivan, Charles 3128, Runningmate	TurboImage vs. HPSQL, A (Performance) Comparison
Swafford, Wayne 5197, Computer Management Solutions	EDI in the Real World
Swearingen, Jay 7050, The Apex Group	Do You Need A Fiber Optic Local Area Network
Swearingen, Jay 7090, The Apex Group	Never Run Another Cable Again
Swinkin, David 4187, Hewlett-Packard Co.	Desktop Publishing for the 1990's: What's New and What's Next
Symonds, Mark 5054, IISI	Executive Information Systems-Realizing the Potential
Szlendak, John, Gleason, Anne 6138, Hewlett-Packard Co.	Optical Storage Benefits, Applications, and Solutions
Thomas, Paul, Ehrman, Brent 4088, Crowe,Chizek and Co.	Network Application Development- Integrating PC's and HP 3000's
Thomas, Paul, Ehrman, Brent 5024, Crowe,Chizek and Co.	Case Studies of MPE/XL Migrations
Toback, Bruce 3014, OPT, Inc.	An Introduction to Object-Oriented Programming and Databases
Toback, Bruce 3132, OPT, Inc.	Using MPE/XL Debugging Tools
Twietmeyer, Tim, Sternadel, Dan 6185, Hewlett-Packard Co.	Performance Predictions Techniques
Van Sickle, Larry 3068, Tymlabs Corporation	Improving Program Reusability
Vega, John 7147, Hewlett-Packard	Understanding The Bridge and Router
Volokh, Eugene 3141, Vesoft	Making Other People's Programs Do What They Were Never Intended To Do - The "Hook under the Hood"
Volokh, Eugene 3142, Vesoft	Automated Testing: One Company's Experience
Volokh, Vladimir 3123, Vesoft	The Biggest Computer Security Threat
Walker, Doug 4006, Walker Richer & Quinn	A Cooperative Processing Strategy for the 90's
Walker, Russell 4096, CompuMed Inc.	PC Programming for Minicomputer Specialists
Warns, Terry 3116, WL Software	Structured Online Program
Warzecha, Charles 4049, Gateway Systems Corporation	Distributed Application Processing
White, Fred 3072, Adager	Integer Keys: The Final Chapter
Willems, Dianne 5080, Hewlett-Packard Co.	Simplifying Migration with Standards: A Case Study of Effective Team Management
Willis, Mark 3707, American Airlines	Performance Measurement Of Multiple CPU's: A Tale of 200+ Systems

Wirl, Charles — Understanding Basic Cabling, Sitewire, and Starlan
7148, Wilco Communications Inc.
Wiseman, David, Miller, Bill — The Technology of Data Compression and its Benefits to HP 3000 Users
6125, Proactive Systems, Inc.
Wiseman, David — Transaction Logfiles - Their Uses
3196, Proactive Systems, Inc.
Wiseman, David — Look, No Secondaries Optimum Master Set Capacities - From Art to a Science
3094, Proactive Systems, Inc.
Wowchuk, Jim — Costing PC Integration..Is it worth it?
4035, Vanguard Computer Services
Yori, Robert — Designing Multi-Vendor Lans That Work
4005, Hewlett-Packard Co.
da Silva, Mark — SQL and the Relational Data Model for the IMAGE Database
3110, Univ of Hawaii Cancer Research

MPE PROGRAMMING

3003 A Beginner's Guide to Programming for Maintenance and Portability
Roy Buzdor - Gateway Systems Corp.

3007 A Means to an End-System Performance is a Journey Rather Than a Destination
James Harvison, Charles Coleman - Technology Applications Group, Inc.

3009 A Programmatic Approach to Tape Loading
Paul Ferguson - Wall Street Concepts

3010 A Solution for Sales Management's Nightmares: An Omnidex Environment
Marie Gannon - United Liquors, Ltd.

3014 An Introduction to Object-Oriented Programming and Databases
Bruce Toback - OPT, Inc.

3015 And Now For Something Completely Different: International System Manager's Perspectives
Kim Barton, Wayne Laughton - Mackenzie Financial Corp.

3017 Backup Performance or How to Backup Faster!
Bryan Carroll - Hewlett-Packard Co.

3018 Basic Resource Accounting
Bruce Senn - Union College

3019 Beyond SYSGEN/VOLUTIL: Volume Management on MPE/XL
Sandra Iwamoto - Hewlett-Packard Co.

3020 Bottlenecks: How to Recognize and Resolve Them
Ralph Kotoski - Hewlett-Packard Co.

3025 Analysis-Design, CASE and OOP a "State of the Art" Assessment!
Robert Mattson - WIDCO

3028 Cobol And View Today: Intergration of Cobol 85 and Hi-Li
Rick Hoover - CIV Software

3029 Complex Data Structure, Omnidex and Relational Retrieval
Kimball Everingham - SRI International

3037 Create Your Own On-Line Help System
Allen Burns - Rutgers-The State Univ of NJ

3039 Databases Therapy: a Practitioner's Advice for the 90's
Alfredo Rego - Adager

3041 Designing User Inquiry Screens for Keyed Access
Paul Edwards - Bradmark Computer Systems

3058 Fundamentals of IMAGE: Schema & Structure for the User
Joseph Speier - Speier Assoc.

3062 How to Improve All Your Printed Reports Using Laserjets
Jason Kent - Proactive Systems Ltd.

3068 Improving Program Reusability
Larry Van Sickle - Tymlabs Corporation

3069 Improving Remote Database Access Performance
Roger Lawson - Proactive Systems, Inc.

3072 Integer Keys: The Final Chapter
Fred White - Adager

3074 JCWs: An Underutilized MPE Feature
David Largent - N.G.Gilbert Corp.

3075 KSAM 101
John Alleyn-Day - Alleyn Day International

3076 KSAM - A Different Approach
D.W. Conger - Amarillo College
3078 Looping Through Image Datasets with Protos
Brian Becker - NOMC, Inc.
3079 Advanced CI Programming - The 2.1 Story
Scott Cressler, et al - Hewlett-Packard Co.
3082 Maintaining the Integrity of Your Software
Alan Padula - Hewlett-Packard Co.
3091 No Time, No Money, and the System Needs a Complete Rewrite
John Fusek, Serena Fusek - Virginia Int'l. Terminals
3092 Omnidex/Imsam as a Database Design Tool
Katherine Newcombe - Systems Consultant
3093 One Company's Successful Implementation of Software Guidelines
Daniel Marcotte - United Liquors Ltd.
3094 Look, No Secondaries Optimum Master Set Capacities - From Art to a Science
David Wiseman - Proactive Systems, Inc.
3099 Performance in Relational Databases
Eric Savage - Dynamic Information Systems Corp.
3107 QUIZ/QTP Performance: Problem Areas
David Robinson - PowerSpec International
3108 Real World Performance Monitoring
Robert Lund - Robert Lund & Associates
3110 SQL and the Relational Data Model for the IMAGE Database
Mark da Silva - Univ of Hawaii Cancer Research
3111 Securing Your HP 3000
Dennis King, Mariette Araya - Hewlett-Packard Co.
3112 Eight Unknown Ways to Obtain SM Capability in Under Sixty Seconds
Carl Melclior - Independent Contractor's Group
3113 Software Creation with a 4GL Language: Lessons Learned
Gene Harmon - AH Computer Services, Inc.
3114 Stop Babysitting Your HP 3000
Craig Conrad - McDonnell Douglas Space
3115 Stop! Don't Pick up the Phone! Order Electronically
Kathy Spanel - GBS Consultants, Inc.
3116 Structured Online Program
Terry Warns - WL Software
3118 System Managers Toolbox
Isaac Blake - City of Tempe
3119 System Performance Through Queue Tuning
Warren Bunnell - ORBIT Software (USA), Inc.
3120 System Performance/Management For Technical/Nontechnical/Personnel
Dave Jennings - Facer-Sys. Performance Div.
3121 Table Your Database
Mike Mitchell - Hewlett-Packard Co.
3122 The Automation of Software Testing
Susan Maxwell - Hewlett-Packard Co.
3123 The Biggest Computer Security Threat
Vladimir Volokh - Vesoft

3128 TurboImage vs. HPSQL, A (Performance) Comparison
Charles Sullivan - Runningmate
3130 Procedure Exits on MPE XL
Rajesh Desai - Hewlett-Packard Co.
3132 Using MPE/XL Debugging Tools
Bruce Toback - OPT, Inc.
3134 When to Use IMSAM/When to use Omindex/ (When to Use IMAGE)
Suzanne Harmon - AH Computer Services, Inc.
3137 Your Own Private Program Library
Gerrit Altena - A&B Automatisering
3141 Making Other People's Programs Do What They Were Never Intended To Do - The "Hook under the Hood"
Eugene Volokh - Vesoft
3142 Automated Testing: One Company's Experience
Eugene Volokh - Vesoft
3184 POSIX and X Windows on MPE XL
Scott Cressler, Ryk Cressler - Hewlett-Packard Co.
3186 Analyzing MPE XL Performance: What is "Normal"?
Doug Grumann, Marie Weston - Hewlett-Packard Co.
3188 Performance Contracts: Service Level Agreements in the Real World
Rex Backman - Hewlett-Packard Co.
3189 Object Oriented COBOL: A New Initiative From HP
Megan Adams, Dmitry Lenkov - Hewlett-Packard Co.
3190 Improving the Efficiency and Accuracy of Your Information Capture Through Automated Data Collection
Raymond Agrusti - Eagle Consulting + Development Corp.
3196 Transaction Logfiles - Their Uses
David Wiseman - Proactive Systems, Inc.
3201 Evaluating the Business Impact and Risks Associated with a Disaster
Gary Fletcher - Hewlett-Packard Co.
3202 Comparing MPE V and MPE XL Customer Workloads
Julie McClung - Hewlett-Packard Co.
3203 The HP 3000 Open Systems Environment
Janet Garcia - Hewlett-Packard Co.
3205 HP-FL A Performance Growth For MPE XL Systems
Melody Armstrong - Hewlett-Packard Co.
3206 Design, Installation and Maintenance of Cabling Infrastructures
Charles Breed - Hewlett-Packard Co.
3207 The Truth About Developing Co-operative Applications
Robert Dey - Hewlett-Packard Australia Limited
3213 Quality Software -- Walking It Through
Lisa Burns - Hewlett-Packard Co.
3214 What Does This Benchmark Mean To You?
Frank Rowand - Hewlett-Packard Co.
3222 Relational vs. Network - The Dilemma of the '90's
Shelley Meiklejohn - Price Waterhouse
3230 Effective Systems Development Using 4GL
Douglas Colter - Infocentre Corporation

PC INTEGRATION

4001 Integrated Image Applications
Malcolm Sharratt - Hewlett-Packard Co.
4005 Designing Multi-Vendor Lans That Work
Robert Yori - Hewlett-Packard Co.
4006 A Cooperative Processing Strategy for the 90's
Doug Walker - Walker Richer & Quinn
4016 Background Operation of Electronic Mail Within the HP NewWave Environment
Dave Morris - Hewlett-Packard Co.
4021 Buddy, Can You Spare Some Time?
George Blessing - City of Pasadena
4033 Cooperative Processing Today with PC's, Macintosh's and, HP Minicomputers
Ron Arons - Sybase Inc.
4035 Costing PC Integration..Is it worth it?
Jim Wowchuk - Vanguard Computer Services
4036 Create A Data Dictionary on PC
Phil Nguyen - Lockheed
4038 Data Synchronization
Lawrence Kingsbury - Gateway Systems Corporation
4040 Designing Distributive Systems for PC Networks
Paul Mole - PMC, Inc.
4043 Developing Distributed Applications using LM/X
Grant Sidwall - Hewlett-Packard Co.
4044 Developing Portable Applications in the Powerhouse PC- A Case Review
David Robinson - PowerSpec International
4045 Developing a NewWave User Interface for MPE and HP-UX Applications
Mike Broadway - Hewlett-Packard Co.
4049 Distributed Application Processing
Charles Warzecha - Gateway Systems Corporation
4057 File Serving with the HP 3000
Bud Booth - AA Software Development
4066 Implementing NewWave - A Case Study
Don Lewis - The Apex Group
4073 The HP 3386 and HP 3286 Integrating the HP 3000 with the PC The next Generation of Computer Systems
Leroy Friesenhahn - Blanket Resources
4084 Microcrimes and Megabucks
Dr. Sanford Sherizen - Data Security Systems, Inc.
4088 Network Application Development- Integrating PC's and HP 3000's
Paul Thomas, Brent Ehrman - Crowe,Chizek and Co.
4095 Organizing Information Through Technology
Barry Prokop - Fairfax County Public School
4096 PC Programming for Minicomputer Specialists
Russell Walker - CompuMed Inc.
4097 PC/HP 3000 Integration - Who Does What
Craig Fransen - Fransen/King Ltd.
4129 Understanding Printer Sharing Solutions
Thomas Durr - Wilco Communications
4131 Using Information Access to Get SQL Data to PC Users
Robert Ross - Hewlett-Packard Co.

4133 What's Behind Those Windows
John Stachowiak - NW Tri County Inter. Unit
4135 Writing Machine Independent C Programs on the HP 3000
Joseph Wm Berry - Dataform Corp.
4187 Desktop Publishing for the 1990's: What's New and What's Next
David Swinkin - Hewlett-Packard Co.
4229 A Developer's Perspective of NewWave
Scott Marshall - IISI
4231 A Client/Server Model (Sales/Lead Tracking and Management)
Bob Hoover - Infocentre Corporation

MANAGEMENT

5023 Building A Software & System Support Team
Don Jones - INLEX Inc.
5024 Case Studies of MPE/XL Migrations
Paul Thomas, Brent Ehrman - Crowe,Chizek and Co.
5031 Controlling the Development Process
Thomas Eineker - PMC,Inc.
5034 Cost Justifying a New Technology
Richard Phillips - OPT (Office Products Technology)
5046 Disaster Planning-Lessons from the 1989 California Earthquake
Bud Beamguard - Syntex Research
5051 EDI - Change or Die! - A Survival Manual
Jim Himmelreich - Graco Childrens Products, Inc.
5052 EDI...A Programmer's Perspective
Joel Kushner, Donna Clark - Merit Systems,Inc.
5054 Executive Information Systems-Realizing the Potential
Mark Symonds - IISI
5063 How to Win at Your Next EDP Audit
Betsy Leight - Operations Control Systems
5067 Implementing Sales Force Automation Systems
Mark Shirman - Innovative Information Systems, Inc.
5070 Improving Software Quality
Robert Green - Robelle Consulting Ltd.
5080 Simplifying Migration with Standards: A Case Study of Effective Team Management
Dianne Willems - Hewlett-Packard Co.
5081 Maintaining a Quality Staff
Forrest Ford - Forrest Ford Consultants, Inc.
5085 Migrating Compatibility Mode Systems to Native Mode
Kevin Darling - The Gap, Inc.
5104 Project Management-Too Simple to Believe!
David Rubinstein - Innovative Information Systems
5124 The Greeting Behavior of the North American Homo Sapiens
Louis R. Mills - Bio-Rad Laboratories
5127 'Tinker-Ware:' Maximizing Your Investment In Application Tools
Leonard Block - The Apex Group

5182 Leasing - A Strategy to Reduce Costs and Manage the Future
Paul Perricone - Hewlett-Packard Co.
5191 Getting People in Your Company to use End-User Computing Tools
Kathy McKittrick - McKittrick Associates
5192 Information Management and Competitive Edge
James Leverentz - Off-Site, Inc
5193 45 Training Tips: A Tip A Minute
Tom Elliott - Distribution Resources Co.
5194 Education:In-House, Outhouse or CBT?
James Davis - Cognos Corp.
5195 Sales Force Automation Today
Michael Barry - Gateway Systems Corp.
5197 EDI in the Real World
Wayne Swafford - Computer Management Solutions
5198 Don't Be Cruel to a Heart That's True
Diane Amos - Amos & Associates
5199 Index Technology versus Relational A User's Dilemma
Daniel Filkins - Dynamic Information Systems Corp.
5200 Information Management-The Corporate Weapon of the 90's
Robert Andrews, Terry OBrien - Dynamic Information Systems Corp.
5209 Interactive Computer Graphics in Business
Jerry Kopan - Hewlett-Packard Co.
5210 Logic Programming and Expert Systems Usage Within Hewlett-Packard
Dave Elliott - Hewlett-Packard Co.
5211 Business in the 90's: Reducing Time to Action
Steve Jeffery - Hewlett-Packard Co.
5215 Fostering Creativity in a Technological Environment
Kent Hanson - Hewlett-Packard Co.
5216 Network Management Today: Fact or Fiction?
Carol Crall - Hewlett-Packard Co.
5218 Information Management Technologies Into the 1990's
Orland Larson - Hewlett-Packard Co.
5220 Stress Management Made (Relatively) Easy
Robert M. Gignac - C & M Products Limited
5234 System Management and Operations Automation
Linda Hazzan, Marge Vaillancourt - Carolian System International, Inc.

GENERAL INTEREST

6030 Controlling External Devices Through Terminal Ports
Grant Parry - Message Transport Systems
6047 Disk Arrays - Mass Storage of the Future?
Ed Pavlinik - Hewlett-Packard Co.
6048 Disk Recording Technology - From DC to Light
Michael Rusnack - Hewlett-Packard Co.
6065 Image Processing: How should it be Utilized
Brent Ehrman, Paul Thomas - Crowe, Chizek and Company

6105 Protecting Your Data from System Failures
Gilles Schipper - G. Schipper & Associates Inc.
6125 The Technology of Data Compression and its Benefits to HP 3000 Users
David Wiseman, Bill Miller - Proactive Systems, Inc.
6126 The Increasing Number of Applications for HP's DDS Format Tape Drives
Debbie Beech - Hewlett-Packard
6138 Optical Storage Benefits, Applications, and Solutions
John Szlendak, Anne Gleason - Hewlett-Packard Co.
6139 Mass Storage Technology Positioning
Jon Schiedel, Connie Doster - Hewlett-Packard Co.
6185 Performance Predictions Techniques
Tim Twietmeyer, Dan Sternadel - Hewlett-Packard Co.
6212 C++: A New Software Development Methodology
Dmitry Lenkov, Michael Beckmann - Hewlett-Packard Co
6223 Optical Disk Technology: Past, Present and Future
Husni Sayed, Deborah Cobb - IEM, Inc.

NETWORKING

3707 Performance Measurement Of Multiple CPU's: A Tale of 200+ Systems
Mark Willis - American Airlines
7004 A Case Study for Networking the Production, The Administration and The Sales
Rainer Schulze - Hewlett-Packard Co.
7011 A User's View of HP/VAX Connectivity
Wayne McKinney - Lockheed Engineering
7012 ATP Ports Solve Telex Operator Problem
D.R. Kambey - Pt Arun Natural Gas Liquefaction Co.
7050 Do You Need A Fiber Optic Local Area Network
Jay Swearingen - The Apex Group
7055 Experience with Distributed X.25 Multivendor Networking
Alan MacIntosh - Hewlett-Packard Co.
7056 Experience with Networking-Moving to x.25
John Howitt - ECC Building Products
7059 Growing Your Lan into a Wan-Lan
Michael Russell - Hewlett-Packard Co.
7060 Creating A Global Enterprise Lan
Charles Breed, Brice Clark - Hewlett-Packard Co.
7061 How Can Networking On MPE XL Systems Help You Meet Your Business Needs?
Gregg Kornfield - Hewlett-Packard Co.
7064 How to be a 'Smart' User of a 'Dumb' Terminal
Michael Hornsby - Beechglen Development, Inc.
7083 Matching Network Servers to User Needs
John Green - Hewlett-Packard Co.
7086 Case Study: Migrating DS to NS, MPE V to MPE/XL and X.25 to ThinLAN*
Mary Fleischman - Washington Community College
7087 Migration From the Data Terminal Controller
James Cohen - Mecca Leisure Group Ltd.

7089 Networking in a Multi-Platform Environment
Mike Eckstein - The Gap Inc.
7090 Never Run Another Cable Again
Jay Swearingen - The Apex Group
7101 Problem Isolation Techniques for TCP/IP Networks
Mike Pozzi - Hewlett-Packard Co.
7103 Program to Program Communication - HP and IBM
Allan Gregory - EI DuPont de Nemours, Inc.
7109 Remote Process/Job Scheduling using NETIPC
Garry Osborne, Michael Mazza - Brown & Williamson Tobacco
7117 Survival Techniques For a 'No-Fault' HP 3000
J.G. Page, Shanda Snyder - Union Camp Corp.
7136 X.25-The Federal-Mogul Story
Tim Cahoon - Federal-Mogul Corporation
7143 Storage: File Servers in a Client/Server Computing Environment
Gene Jung, Jean Tang - Hewlett-Packard Co.
7145 Performance Management in the Distributed Computing Environment
Tony Engberg - Hewlett-Packard Co.
7146 Building a Network Management Solution
Steve Richard - Hewlett-Packard Co.
7147 Understanding The Bridge and Router
John Vega - Hewlett-Packard
7148 Understanding Basic Cabling, Sitewire, and Starlan
Charles Wirl - Wilco Communications Inc.
7204 ARPA Services For MPE XL
Deborah Campbell - HP-Business Network Divisions
7233 Migrating To NS - How and Why
Ann Smith, Steve Cole - Hewlett-Packard

TUTORIALS/ROUNDTABLES

T174 Tutorial Managing HP 3000 Performance
Brian Nogar - Hewlett-Packard Co.

A Beginner's Guide to Programming for Maintenance and Portability

Roy Buzdor
Gateway Systems Corporation
2400 Science Parkway
Okemos, Michigan 48864
(517)349-7740

INTRODUCTION

Recently I had a professional employment recruiter ask me if I knew COBOL or RPG. This only goes to show that although the current buzzwords in the computer industry are Expert Systems and Artificial Intelligence, the maintenance of existing programs continues to be an important issue. I doubt that this is news to anyone involved in programming. The companies who are most aware of the maintainability issue are large businesses who have been using computers to support their business for 20 or more years.

Many factors have led to this situation where maintainability is so important. Some of the reasons are:

1. The computer was brought in to help the business, but data-processing departments were not planned.

2. The "computer expert" was usually the employee who took enough of an interest in the computer to be able to turn it on and off at night, or when the power went out.

3. Computer experts either learned to program on their own, or, if they were lucky, they were sent to a seminar put on by the computer vendor. Such seminars were usually taught by the vendor's computer experts, who learned to program on their own.

4. People who wrote programs wrote them for a specific purpose, and, for the most part, did not think about the programs already in use.

5. Program maintenance meant keeping the card-deck in order and properly labeled, or (on smaller computers) the paper-tape wound and not dog-eared.

Natural selection played a part in each program's history. If someone needed a least squares program, one was written to that person's specifications. If someone else wrote another least squares program which was easier to use, or faster, or more flexible, or took less memory, the original fell into disuse, and the new one took its place. This was especially true as disk drives came along, and all programs had to vie for the 512K of disk available. University graduates in diverse fields came in and brought their "pet" programs which they had written and/or collected. Some of these were used in place of existing programs, because they were better, or because of who could be persuaded to use the programs. Everyone was responsible for their own programs.

What has all this got to do with programming today? Some of those programs, written in FORTRAN, BASIC, or Assembler, are still in use today. Unfortunately, these programs may consist of 2000 to 8000 lines, with no more than 7 comments. They may have been written to run on systems with a maximum of 8K bytes of memory, where jumping back 1300 lines with a "GOTO" statement was preferable to adding another 10 bytes of code. And these programs were probably written by programmers who taught themselves how to program simply by reading the manuals.

I know about these programs, because I have had to move some to other computers, add functionality, or fix problems. In other words, I have had to maintain them. I have learned something about maintainability from every existing program that I have worked with. This paper will present some of the techniques that I have learned in my experiences.

Maintainable versus temporary programs

Before considering how to maintain a program, it's important to understand that not all programs are meant to be maintained. Any time you write a program, you must decide whether it will be a "permanent" program or a "temporary" one. There are valid reasons to write temporary (or QD – "Quick and Dirty") programs. One is that they are quick to get up and running. The important thing to remember is that if you write a QD program, GET RID OF IT! If you keep it around, someone will have to support it. If you need to keep a program, spend the time to design it correctly and make it maintainable.

Consider the following programming example. In this example, you need to swap two fields (field 2 and field 4) in data set "X". Since you don't have ADAGER or DB General, you must:

1. Unload dataset X into a file
2. Store the rest of the database
3. Purge the database
4. Change the schema
5. Reload the database
6. Swap the columns in the file
7. Reload dataset X

The program to load and unload the data sounds handy – it is a program that you could use over and over. However, the program that swaps fields 2 and 4 is a program that will be used once and discarded. Write the programs as such: spend time designing (or locating) a good program to load/unload a dataset. (In case you are interested, there is a set of programs in the CSL to do this; DB2DISK and DISK2DB.) The program to swap fields should be written as quickly as possible, in whatever language you feel most familiar with, tested, used, and purged.

OK, you are going to design and write a program which will be handed down from generation to generation, venerated, and (when it is finally retired) enshrined. How do you go about doing it?

DESIGNING MAINTAINABLE CODE

Rule 1: Study the Problem

Many programmers make the keyword in this phrase "problem", since their ultimate goal is to solve a problem. However, the keyword in this rule is "study".

There is a story of a magnificent painting, the subject of which was a storm on the sea. You could almost feel the wind in your face, and the clouds were terrifying. One admirer asked the artist how he had been able to produce such a masterpiece. The artist explained that for many years he had set up his easel on the shore and tried to paint the storms he saw at sea. Time after time he tried to capture the storm on canvas, and time after time he discarded the painting, because it was just a painting. Finally, he hired a boat to take him out on the sea, had the captain tie him to the mast, and head into a violent storm. After feeling the storm's fury, and fearing for his life, he was able to paint a storm at sea as he had never done before.

While you may not want to be tied to a desk in the lab/office or to the bumper of someone on the traveling sales force, you must nevertheless analyze your users and the problem. Understanding the problem will assist in being able to anticipate the questions which were not asked in the original design, but will be asked the day after the program is released.

In the dataset load/unload example mentioned earlier, you must know before you begin if the program should load dataset by name and/or number, or if it should process all datasets with an "@". If you write the program to do one dataset at a time, by number, you will be asked for multiple/all dataset capability with access by name before you are back to your desk.

Rule 2: Build in Expandability

No matter how well you study the problem and the users, you will not be able to imagine what the future holds. No matter what you design a program to do, somewhere, sometime, someone will want it to do something a little bit different.

Sometimes the alterations that users request are not appropriate. If you wrote the dataset load/unload program and someone wanted to use it as a report generator (after all, it does access a database), you would need to explain that what you have is a wheel, and what they want is a freight train! While freight trains do have wheels, there is a big difference in the two.

However, after you write the dataset load/unload program, someone might want to format binary data into their text equivalents, and take the text equivalents and load them back as binary. In that case, what you have is a wheel, and what they want is a unicycle. If you can add a saddle and pedals, it might work.

If you wanted to convert the load/unload program to convert binary to text on an unload, and text to binary on a load, you might wish that you had designed the program with expandability in mind. You could have used an input buffer to read in a record, and an output buffer to write it out. In fact, you could have allowed the user to read by an arbitrary field order instead of reading the data into a buffer using the "@" identifier. That would have made the program to swap fields unnecessary. Merely reading the original dataset as "fields 1, 4, 3, 2, 5...", would unload it in the order you wanted.

As these example demonstrate, if you plan for the unexpected, new capabilities can be implemented easily.

Rule 3: Build in Test Systems

An old friend of mine built a Heath-Kit stereo amp and tuner for his home music system. It was an above-average system, but what really impressed me was what happened when the power lines in his neighborhood were struck by lightning. The system which he had spent each evening for the past 3 months on went belly up, or so I thought. As it turned out, the signal-strength meter on the front of the stereo was also a volt-meter, and by checking predetermined spots inside the system he was able to diagnose and fix the problem in one Saturday morning. Being able to use the system to test itself was a real time and money saver. In electronic and military systems this is called Built-In Test Equipment (BITE).

This is an extremely useful tool in programs. There are four types of problems that may occur in a complex program:

1. The program may have a bug.

2. The data that the program uses may be beyond the bounds that the program was designed for.

3. The environment that the program is operating in may be causing interference with the otherwise normal operation of the program.

4. There may be a problem with the computer hardware/operating system.

(These areas are listed in the order in which they generally occur. Usually you encounter bugs, and you'll hardly ever see honest-to-goodness computer failure.)

Designing in debug printout is a simple way to build in testing. You know where the logical junctions are and what data is important – simply put a "PRINT this" statement where you needs more information. When you are "done" with the program, use "IF" statements to bypass the print statements (assuming the printout is no longer needed). Print statements make the program bigger, but won't slow it down perceptibly when they are bypassed. For what it costs in design time, and what it adds in time and size to the program, you cannot beat this method.

WRITING MAINTAINABLE CODE

Rule 4: Use Mnemonic Names

Shakespeare wrote "A rose by any other name would smell as sweet" or at least something very close to that, and it may have been Bacon. However, a wise man would not call his wife "Beverly" if her name was Theresa, it just ain't smart.

Many programmers were trained on a system which limited variable names to 5, 6, or 8 characters seem to be mentally stunted in the file naming area (as was I for quite some time). COBOL programmers seem to have less of a problem in this area than do FORTRAN or BASIC programmers. COBOL has been more liberal with its name-length-allowance, permitting names like "HOURLY_SALARY" versus the FORTRAN equivalent "HRLYSL" or the traditional BASIC "H5". I have written systems in BASIC where the idea of using "D8" for "Date" was an inspiration. Most of the modern versions of languages ("Business BASIC", "FORTRAN 77", etc.) now allow at least 15 characters, and some 31 characters for a name. USE THEM.

Another useful naming convention is to group names together via a common prefix or suffix. For example, I wrote several modules for an application, one of which had to do with locking the database(s). All constants/types/variables directly related to that module had the prefix "LK_". This convention has at least three definite advantages:

1. The names are easy to spot in code or on a cross reference. This is especially true if you use a prefix.

2. When browsing through a separate module, it is simple to see if the module is using or altering any of these variables simply by looking for the prefix/suffix.

3. You are far less likely to use a name which is already in use elsewhere. Even if the body of the name is the same, the prefix/suffix will make it unique.

In systems where it is possible, do not override system names with your own definitions. In a program that was working on a classic HP 3000 system, I wanted to use 16-bit integers through the entire program. Being naive at the time, I defined the keyword "INTEGER" as being a 16-bit integer (it is normally a 32-bit integer). Things worked great, until I tried to access some external routines. The ugly part to the story was that it took two and a half days to find the problem because everything looked like it was supposed to...where the external definition said "INTEGER", my definition said "INTEGER". Turns out my integers and their integers were different (surprise!).

Rule 5: Include Comprehensive Comments

Self-documenting languages don't. I once saw a Pascal program written in English-prose style. It read beautifully. It looked kind of like the classic COBOL routine:

```
Multiply hourly_salary by regular_hours_worked add hourly_salary times overtime_differential times
over_time_hours_worked giving gross_weekly_pay.
```

This code is like Wally Cleaver, a real model of the way things probably ought to be, but nowhere near the way they really are. As a matter of fact, being an experienced programmer, it doesn't even look right to me to have code that looks like English. I would have trouble trying to follow the logic. Alternately, 30 lines of pure programming code, while easy to comprehend on a line-by-line basis, may not provide a clear idea of the purpose of the module. A simple comment like the one shown below can help the person who is maintaining your code locate the section of code that needs to be fixed when we accidently nuke Cleveland.

```
*************************************************************************
**                                                                     **
**      Here is where we check to see if the missile is still on course **
**      by comparing our theoretical position "THRPOS" to the position  **
**      computed by the last radar reading "RADPOS". If we are off by   **
**      more than the acceptable error "ACPERR", go to the self-        **
**      destruct routine at 9999.                                       **
**                                                                     **
*************************************************************************

IF (abs(THRPOS - RADPOS) > ACPERR) GOTO 999
```

A bane in programming is writing the program, and then going back trying to figure where to put a comment or two. They seldom are put in the right spot, and usually look something like:

```
/* Divide WEIGHT by VOLUME to get CONCENTRATION */

Concentration = Weight / Volume;
```

A very helpful comment - for someone who doesn't know what the "=" and "/" symbols mean. Comments should give more of an overall idea of what is going on; specifics can be deduced on a line by line basis. One useful habit is to write the skeleton of your program in the comments, then fill in the actual lines of code. For example:

```
Program   UnLoad
(Input,
Output)

Begin
{Initialize variables}
{Get the Data-Base and Data-Set name}
{Open the Data-Set}
{Open the MPE File}
While ({more records in Data-Set}) do
        Begin
        {Read the next record from the Data-Set}
        {Format it into all Text}
        {Write it out to the MPE File}
        End;
{Close up shop}
End.
```

As you write the routines to do the work, you will know what each section is supposed to do, and each main block will already have comments. On a longer module, it's helpful to leave a copy of this outline at the top of the module, so that a person going through it later could look at this and get a feel for the functioning of the module.

Rule 6: Divide the Task into Manageable Pieces

This is a rule I am just beginning to appreciate. I started out writing interrupt I/O routines in assembler for laboratory equipment, where every micro-second counted. Page after page of code was written in a line straight down. Calling a subroutine or procedure would have wasted 6 micro-seconds for the "CALL" and the "RETURN". That was too expensive, because a mass spectrometer had to call the routine every 100 micro-seconds. If you are in a similar situation, do what you must. If you're not, make wise use of subroutine/procedure calls.

Some purists cringe at any piece of code more than 20 lines long. I wouldn't go that far, but it probably is not a bad starting point. Treat a routine as you would a paragraph in a manuscript - - it should be one continuous, cohesive thought. Don't ramble on, don't stick in stray ideas.

One note about writing routines. Try to keep each routine as self-contained as possible. Avoid global variables. Global variables make tracking down problems and maintenance far more difficult. They can also cause problems globally if another routine does not expect this one to set the global variable to what it did. It may take weeks to track down problems!

Rule 7: Use Code Libraries

Code libraries help greatly with carving up tasks. Once you have a routine working and debugged, use it. Use it everywhere. I wrote a routine I call "StrUpShift", which upshifts a string (kind of obvious). Every time I have an occasion to shift a string, I call StrUpShift instead of writing a new shift-to-uppercase routine. If something is incorrect, there is only one copy of source-code to fix. There is also only one copy of the code in any program, not 12 different routines taking up space. StrUpShift is now in my code library, along with other handy routines which have already been debugged.

Another advantage that code libraries afford, is that as the libraries build up, writing and maintaining programs gets easier. When you're developing a program, if you come across a function that already exists in your library, that is one less thing to write. The same is true for maintenance. If the particular function being enhanced/fixed is one which exists in the library, you simply delete the old chunk of code and add a call to the appropriate routine in the library.

A final advantage to code libraries: an enhancement to a routine in the code library becomes available to all programs that use the library. Programs do not have to be rewritten to incorporate the feature, just recompiled. Beware though, because new bugs can be introduced in the same easy manner!

Rule 8: Beware of Short Cuts and Fancy Tricks

I went to the circus a while back; near the end, a man in tights and a cape climbed into a canon and with a drum roll he was shot all the way across the arena! I was amazed! In 2 seconds, he traversed a distance which would take me 15 minutes of going through corridors, and getting around other spectators! However, a little while later when I went to get popcorn and soda, I took the back way. There is programming code out there just like that.

```
Length := Length * (3 * Ord(Length >= 0) - 1);
```

There is a slick piece of code! Study it for about 15 minutes, write down what the final value of "Length" will be when it starts out as "5" or "-5" (assuming that Boolean value of "False" is "0" and "True" is "1"). How long did it take you to figure it out?

```
If (Length >= 0) Then
        Length := 2 * Length
Else
        Length := Abs(Length);
```

This is a very boring piece of code. It takes no inspiration to write, no special insight to read, and you probably understood its function before you started this paragraph. (It also makes no assumptions like [the Boolean value of "False" is "0" and "True" is "1"] which may or may not be true from machine to machine, or language to language.)

These two pieces of code do the same thing. If the number is negative, they make it positive, if it is positive, they double it. It is the code used to get the length of a buffer in positive number of bytes when given the length in words (positive) or bytes (negative). If you were writing the code, you might be thoroughly impressed with yourself for dreaming up the first routine. If you were maintaining the code, admiration would probably not be the first emotion to well up in your bosom for the one who wrote it. If you break the two bits of code down into machine instructions, you see that there may be very little difference in length, and that the "tricky" code may be no savings at all:

(1) Length := Length * (3 * Ord(Length >= 0) - 1);

```
LDA  LENGTH         ; Start with the length
CPI  0              ; Is it >= 0
LDA  S1             ; Get the result of the last operation
MPYI 3              ; Multiply instruction...VERY SLOW
SUBI 1              ; Subtract one
MPY  LENGTH         ; Multiply instruction...VERY SLOW
STA  LENGTH         ; Put it back in length
```

(2)
```
If (Length >= 0) Then
        Length := 2 * Length
Else
        Length := -Length;
```

```
LDA  LENGTH         ; Start with the length
CPI  0              ; Is it >= 0
BRN  LESS           ; If not, goto the LESS THAN instruction
MPYI 2              ; Multiply instruction...VERY SLOW
SKIP                ; Skip the next instruction
LESS NEGA           ; If it was negative, make it positive
STA  LENGTH         ; Put it back in length
```

Both pieces of code take 7 instructions. However, the first code uses two multiply instructions, and the second takes only one. Since the multiply instruction is the slowest in the set, the "trick" is actually a dual liability. It is more difficult to understand, and it is slower to execute.

In some languages there are semi-valid reasons for writing illegible code. There are structures that compile into faster code if written incomprehensibly. For example:

```
Index++
```

is logically about the same as:

```
Index = Index + 1;
```

But if the machine has an increment memory (INCM) instruction, the two lines will compile differently. The first should compile into:

```
INCM Index
```

Whereas the second may compile into:

```
LOAD A,Index
ADI  A,1
STOR A,INDEX
```

If there is a difference in the way legible programming and tricks compile, and if the extra speed/space is really important, then use the tricks. But comment them so that someone else who is familiar only with Pascal, FORTRAN, or COBOL, can read it two years from now. For example:

```
/* Here is where we increment the Index "Index = Index + 1;    */
Index++
```

Personally, I believe that a compiler should be smart enough to optimize code into its most efficient form, without the programmer needing to know the esoteric constructs which invoke special machine features.

Rule 9: Use Variables or Defined Constants

If you are a Pascal or COBOL programmer, this rule may be so obvious that you believe it's a waste of space to discuss it. However, if you are a BASIC or FORTRAN (pre-FORTRAN 77) programmer, this one may need a bit of explaining.

I stated that the second example for rule 8 made no assumptions (like the values for "TRUE" and "FALSE"). That statement is not entirely true. The example made no language-dependant assumptions, but it did make a machine-dependant assumption. HP 1000 and HP 3000 programmers may not have noticed the assumption, because they are so familiar with it. It was the "2". On the HP 1000 series and the classic HP 3000 series (0 - 99) there are 2 bytes in a word. To convert from a length in words to a length in bytes, it takes a multiplication by "2". However, on the HP-PA machines, there are 2 bytes per half-word, and 2 half-words per word, or 4 bytes per word.

If you had a program that was heavily sprinkled with conversions between words and bytes, you can see that it would not be simple to change all the "2"s to "4"s. There will probably be other "2"s which are not related to word/byte conversion, and ought to remain "2"s. It would be a great relief to the person who is doing the migration if somewhere near the beginning of the program there was the statement:

```
        Bytes_per_Word := 2;
```

All subsequent references would then use bytes_per_word

```
If (Len >= 0) Then
        Length := Bytes_per_Word * Length
Else
        Length := Abs(Length);
```

it sure would ease my day, if it were my task!

Rule number 9 mentions two things, variables, and defined constants. A defined constant has the added advantages that it can be used anywhere a constant (like "2") can be used, including in the definition of other constants, arrays, and dimensions. Additionally, the program cannot change the defined constant. Defined constants and/or variables in place of numeric constants also add meaning and readability to the program (as long as you make the names mean something like "bytes_per_word" as opposed to "bpw").

It should go without saying that you should use concept-oriented constants ("bytes_per_word") not value-oriented constants ("one"). An ugly incident (having to with IMAGE) in one of my previous jobs illustrates this. IMAGE requires that all parameters be passed as variables. Quite often the "MODE" parameter is a value of one (for all DBPuts, DBDeletes, DBUpdates, etc.). One database was opened using "mode 1", but was change to open using "mode 5". The database opened all right, but IMAGE had difficulty shortly thereafter when a DBGet was executed. The code read:

```
        DBGet(DataBase, DataSet, Mode1, Status, List, Buffer, Argument)
```

There should be no problem with a "Get" with a "Mode 1". After some inspection, it turned out that some programmer had changed the value of the variable "Mode1" to be FIVE. A better set of variable names would have been "ImageOpenMode" and "ImageGetMode" or, better yet, "OpenRWShared /OpenROShared" and "SerialRead". By the way, this can happen accidently, as easily as by inexperience. Some languages such as FORTRAN or COBOL pass parameters only as variables. This is called pass-by-reference, which means that a pointer to the parameter is passed to subroutines, rather than the actual value. The "caller" may pass in a value which is considered a constant, such as "SerialReadMode", but the subroutine may view it as a variable to be changed at whim. If available, use defined constants. Languages which provide them usually also provide protection from change.

Rule 10: Define Things Thoroughly

This rule is similar to the rule 5 (Include Comprehensive Comments). When defining a constant, type, variable, array, record, function, procedure, or any other concept, include comments describing what it means, where it is to be used, inputs, outputs, how it differs from similar items ("this is like function X, but it does not upshift the buffer"). Develop a standard set of questions which you answer for each item type. If you must err, err on the side of overkill.

Rule 11: Make It Look Pretty

Programming is now locked into being mostly a science; quite a bit of the artistry has faded, but a well-written program is still a thing of beauty. Indent loops, indent branching structures (IF, CASE, SWITCH, computed GOTO, BRNE, etc. statements), and don't cram 30 statements on a line. Everyone has preferences about where to put spaces, what should be upper case, what should be lower case, and how to specify and line up continuation lines. The specific method is unimportant, but using the method consistently is important.

This is probably a good place to put in a word about the much maligned "GOTO" statement, otherwise called "JUMP", or "BRANCH". I will not say "Don't use GOTOS". The GOTO statement has its place, although I find myself using it less and less. The main objection to the GOTO statement is that most programmers who use them make the code illegible. Anytime it is convenient, these programmers will jump anywhere. This leaves the person trying to follow the logic hanging to the end of a thread, with no idea where to go to find the next thread. It is like an Easter egg hunt where the eggs are spread out over an unmown meadow, and they are all colored green and brown speckled camouflage. That may sound like great fun to a kid who is an egg hunt enthusiast, but when it is your job, and you have no idea how many eggs are out there, and you have until 8:00 a.m. Monday morning to find them all, and you have already been looking for a week ... well, the fun diminishes quickly.

Some things to remember when using a GOTO are:

1. Only GO "down" in the code. The only time the code should go "up" is when it is looping. If the code is looping, you should be using a loop construct (DO, FOR, WHILE, REPEAT, etc.).

2. Don't use a GOTO when an IF statement is more descriptive and appropriate for the code (i.e., for languages that have "IF"s that can cover more that a single statement whether TRUE or FALSE). If the only option available is an IF/GOTO pair, use the construct as a block IF would be used, and indent the conditional code. This happens in pre-77 FORTRAN and BASIC.

3. When you are "GOTO"ing a line, make it a non-statement such as a FORTRAN CONTINUE or a BASIC REM. This allows for the easy insertion of statements before the first executable statement, and also helps to make the GOTO targets obvious.

4. Never have the same label mean two different things. If the "Exit the whole program" label is 999, and the final loop ends on the line before line 999, don't use the 999 label to exit the loop. Add another label (maybe 990) before the 999 label, and use 990 to exit the loop. This will help make your code clear.

5. If your language does not have alphabetical labels ("EXIT_PROGRAM"), define a system of numeric labels. For example, always use a last digit of 9 to indicate the exit from a loop; use a first digit of 8 for an exception routine; use 999 to exit the program; etc.

The primary valid use for a GOTO statement is to break out of other control structures. Use them as you would the emergency fire exit at K-Mart. Normally you exit via the doors at the front of the store, after you go through the check-out lanes. You can get out through the emergency exit, but an alarm goes off, and if there isn't a fire, you're in a heap of trouble. If there is a fire, it is a life-saver.

Again, make the code pretty. If the path is well marked, and the code is straight top-to-bottom, no one can complain. GOTO detractors, if shown a well-written GOTO statement, will tuck their tails between their legs and admit, "Well, that is an exception," or "It looks reasonable," or other left-handed compliments.

WRITING PORTABLE CODE

What is portability, really? Portability is just another function of program maintenance. Generally, portability is not been an issue until a company begins to look at the pros and cons of going from brand "X" computers to brand "Y" computers (or, in some instances, when the company simply wants to go to the next bigger machine). Eventually, the company decides either to stay with "Good old Betsy" or they order a feasibility study to see how much it will cost to transport all critical systems to the new computer. When it's time to transfer systems from computer "A" to computer "B", the task is generally assigned to the software maintenance group. Thus, no discussion of programming for maintenance is complete that doesn't discuss programming for portability. Programming with maintenance in mind will make programs easier to transport from system to system, and programming with portability in mind will make programs easier to maintain.

Rule 12: Use a Transportable Compiler

There are plenty of compilers available for every computer. Some were invented in universities to solve artificial intelligence problems, some were built for the military to guide missiles. However, all those compilers do not exist on every machine. If you write in SPL, you write the fastest, most efficient programs the HP 3000 is capable of running. You can forget about running those programs on an IBM/S34, VAX 11/780, or even an HP 1000 though, because the SPL language is unique to the HP 3000. Even as recently as 7 years ago, the idea of being able to take a program from computer "A" and compile and run it on any other computer was not a concern. Most data processing shops used a single brand of computers – they had always had them, and thought they always would have them.

Nowadays, if there is a handy utility on one computer, word gets around to users on other computers, and they want something similar on their machine. Sometimes, a company may find a "turn-key" application which fits a specific need that they have, but which does not run on their computer. Rather than re-invent the radial-tire, the company may opt to buy the new hardware as well as the software. This is particularly true if use of the software will be confined to a single department or segment of the company. Being able to write a program once, maintain one copy, and support multiple computers is suddenly an idea whose time has come.

If you were able to hit <CTRL><SHIFT><F3>, look years into the future and see that you would change over to computer "Q", your only concern would be what language you could program in now that will work on your current computer and computer "Q". Since this isn't possible, it's best to select a computer language which exists across major computer lines. Examples of such languages are Pascal, COBOL, C, FORTRAN 77, or Ada.

There are two categories of standardized compiler definitions and implementations:

1. Richly defined compilers with subsets (PL/1)

2. Sparsely defined compilers with supersets (Pascal)

In the first, every nuance of the language is stated in the formal definition -- if it is done, it will be done like this. For practical purposes, the implementations tend to be manageable subsets of the original with 80% of the features (the 80% that most people use) and only 20% of the size of the original. Problem: when moving from one computer to another, the 80% implementations aren't always exactly the same 80%. However, if you go to a computer that has a 100% implementation, all the programs written in all the subsets will run as written.

In the second category, the language is defined as a base set. Unfortunately, there may be some glaring holes in the language. Almost every implementation will provide a way to do the functions deemed important, but they will most probably do them differently. Problem: there is no certain way to move a program written with extensions and get it to run on any other computer. However, if you stick to a standard definition of the original language, it will run on any machine.

One thing to note about the two philosophies: In the second category, you know when you are out of the range of standard functions (some compilers are even smart enough to let you know when you step over the line, like HP Pascal's "$ANSI$" option). In the first category, there is no way to know what subsets of commands will be implemented from one machine to the next, until you start checking error messages from your conversion compilation jobs.

Ada has attempted to resolve this problem by defining everything in the language (the first category), and then declaring that if the implementation does not support all the features, it isn't Ada. Ada uses no subsets.

Once you select a standardized compiler, you must choose between two coding styles. In the first, you program according to the least common denominator, so that the code will run on other machines, with a minimum amount of adaptation. This may cause you to design the program in a bizarre fashion in order to use standard functions to substitute for the extensions which do what you really need to do.

With the second style, you use functions which may not be available in every implementation, knowing that transporting to another computer may involve weeks of work to make those things which stop working go again.

Luckily, there is a way to use non-transportable functionalities and still minimize the maintenance required to transport the programs.

Rule 13: Isolate Non-Standard Functions

This rule is an extension of rule seven. This is a fairly simple rule which is not always simple to implement. Implementing this rule involves building libraries in which you isolate system-specific code. If your code is has been broken into modules in a logical fashion, and the common modules are in code libraries, you're half-way home.

One difference between the construction of code libraries and libraries constructed to isolate system-specific code is the amount of code in individual routines. In a code library, the idea is to include all the code necessary to perform a logical function. Make the routine as general as possible, for use in as many spots as possible (within limits). Writing a routine which reads input is good. Having it shift the input to upper case may be better. Having the routine also lock a database is not a good idea. The functions included in the routine should be related.

Conversely, the purpose of system-specific modules is to isolate the minimum amount of code possible, because this code will have to be maintained when it is transported to another system. The less code in the module, the less that will have to be inspected and possibly changed. To compare it to the same example given for a code library, writing a system-specific routine to read input may be unavoidable; shifting the input to upper case is a general function, and should not be included in system-specific code.

There are several constructs provided by languages that will help you to isolate system-specific code. Some of these are described below.

1. Conditional compiling

   Most compilers provide a flag that turns the compiler off until it runs into a another flag that turns it back on. The flag could be set to 'COMPUTER_IS_HP3000', 'COMPUTER_IS VAX', or 'COMPUTER_IS_IBM_PC'. However, as discussed earlier, another Pascal compiler may use another construct to define conditional compiling.

   There are a couple of ways to overcome this problem. The editor can be used to change all occurrences of "$IF 'someflag'$" to whatever the other Pascal compiler wants, with negligible chance of changing any unwanted occurrences. A second way involves more work up front. This is to define your own internal standards for conditional compile statements and use these in your source files. Then, write a pre-processing program that will change your definition into that of the target machine. Alternately, the pre-processing program could check the value of the compile/don't compile variable, copy the source file into a temporary file, remove the lines not to be compiled, and finally, use the temporary file as input to the compiler.

2. Use include files

   Include files also allow the programmer to easily change lines of code based on the computer on which they are compiling. One advantage to include files is that the original file is not swollen with three or four copies of the same code, each written for a different computer. As a matter of fact, the code of computer "B" will not even reside on computer "A".

Two disadvantages to include files are:

- The problem of different compilers having different methods of saying the same thing that was discussed previously. This problem can be resolved in the same way as mentioned before – either with an editor, or with a pre-processing program.
- A program may have a large number of very small files associated with it. One solution to this problem is to use conditional compiling when the differential code is short (say 5 lines or less), and use include files when it is large.

3. Move system-specific code into separate functions and procedures

This approach works well because procedure calls in any language are relatively standard across all computers. Simply decide on a function to be performed (opening a file, for instance) and write a standard function or procedure interface which passes and returns all necessary information. Call this routine from the generalized code. Write the guts of the routine as system-specific as you want, keeping in mind that it will have to change for each system that you move you code to. This has the added advantage that it aids the programmer in writing modular code.

The disadvantages to this method are:

- It only works with code, not data definitions. This is a minor disadvantage, since the programmer can still use the methods mentioned previously to handle data definition.
- The second disadvantage is a bit more complicated, and its solution a bit less obvious. Problem: on machine "A", when the user opens a file, the resulting file-identifier is a 16-bit integer. However, on machine "B", when the user opens a file, the resulting file-identifier is a structure consisting of a 16 character name, four 16-bit flag words, and two 32-bit pointers. The solution is to define the type "FILE_ID" conditionally, based on the machine. But what if we're working in FORTRAN 77, which exists on both computers? A good FORTRAN programmer could very easily overcome this with an equivalence statement, overlaying the necessary variables. But what should be used when you want to refer to the file in the main unit? Unlike Pascal, where the variable is the same regardless of the type definition, we are now talking about "FILENUM" if it is the integer, or "FILEREC" if it is the structure. The variable to be used shouldn't matter. This is because it will be used almost exclusively in procedure/subroutine/function calls, and the address will be the same no matter which name is used. If there are anomalies (for example, using FFILEINFO to get the current record pointer on the HP 3000, whereas on machine "X" it is the word 12 of the 16 word array) they can be handled as specified in methods one, two, or three.

Rule 14: Use Independent or Application-Driven Type Definitions

In the previous rule, mention was made of keeping track of files via some type of file-identification tag. On one system it was a 16-bit integer, and on another a more complex structure. The method of handling this in FORTRAN was a series of EQUIVALENCE statements. While this may be tolerable, it is not a method that lends itself to easy maintenance, especially if the programmer is supporting multiple files. In addition, you may need to keep track of records with some sort of record number of record pointer. The record number or pointer may be a 16-bit unsigned integer on one system, a 32-bit signed integer on another, and a 64-bit pointer on a third. Thus, the programmer faces the problem of keeping multiple items straight, doing the arithmetic correctly, checking to see if the item is at a minimum or maximum, etc. While the problem can be overcome in most languages, the idea in good programming is not to see how much the language can be twisted to accommodate the problem, but to find the simplest solution. A more straight-forward solution which is available in other languages is to define a "type" of data which can be associated with the variable and the machine. If the variable is related to the application, rather than to the computer, then tailor the definition to the data.

```
Const
  Min_User_Name_Len      = 1;
  Max_User_Name_Len      = 40;

Type
  User_Name_Length_Range = Min_User_Name_Len..Max_User_Name_Len;
  User_Name_Index_Type   = User_Name_Length_Range;
  User_Name_Array_Type   = Packed Array [User_Name_Length_Range]
                           of Char;

Var
  User_Name_Index       : User_Name_Index_Type;
  User_Name             : User_Name_Array_Type;
```

Here, not only the size of the array, but even the values that the index can have are controlled. It is easier for the programmer to call everything an integer, real, or char. It also causes fewer problems during testing (because you do not get a "BOUNDS VIOLATION" when the 1 to 10 array index mysteriously turns up as 13). In spite of these factors, if you employ a language which provides these definition features, use them. A "BOUNDS VIOLATION" during testing is preferable to a "PROGRAM ABORT" when the user has their hands on it. As a side-effect, the program will increase in transportability, because the machine dependency has been decreased.

Rule 15: Program Parameters and Configuration Files

If changing numeric constants to defined constants, and allowing large-scale changes by simply modifying a few lines and recompiling is good, then changing constants to variables and allowing changes to be made "on the fly" without recompiling is even better.

A prime example of this rule is TYMLABS "PREVIEW" product. Because of the way they have built their product, TYMLABS does not need to maintain a version of PREVIEW for each type of terminal. All the user has to do is set a JCW to indicate the type of terminal they have. When a user has terminal type for which TYMLABS has not prepared, they simply set up a file that describes how PREVIEW is to handle the functions it performs. They do not have to wait six months for the next version to come out.

For a simple idea of how this might work, let's take the example of a graphics program which draws on a graphics terminal. There are five pieces of information which it needs to know to do its work:

1. The control sequence to "raise the pen" (move the pen without drawing a line)
2. The control sequence to "lower the pen" (move the pen while drawing a line)
3. The control sequence to move the pen
4. The horizontal size of the screen
5. The vertical size of the screen

Since there is a relatively small amount of data in that list, the programmer might be able to collect the information for the terminals used most frequently by his/her users. The configuration file for the program might look like this:

```
! An exclamation point is used to denote comments...it is a good
! idea to allow the users to put comments in configuration files
! to keep them from forgetting what means what
```

The same file for a user with an HP2627 looks like this:

```
! Definition for a HP2627
TERMINAL = HP2627
! This terminal is one for which the programmer could
! look up the data in an internal file
```

The configuration file for a user with an SUZUKI-4 looks like this:

```
! Definition for a SUZUKI-4
TERMINAL = SUZUKI-4
! The programmer knows this information too
```

The programmer can almost build the configuration file for a user with an ALAMODE-H, since he/she knows the ALAMODE-G. However, the size is different, so the user defines the terminal as ALAMODE-G, but overrides the size definition in the configuration file. It looks like this:

```
! Definition for a ALAMODE-H
DEFINE   = ALAMODE-H
TERMINAL = ALAMODE-G  ! We know this one
!
MAX-X    = 1024      ! This is the horizontal size for the "H"
MAX-Y    =  792      ! This is the vertical size for the "H"
```

Any definitions following the "TERMINAL" definition, supersede the standards.

The programmer has never seen a CHEAPO-01, so the user must define all the parameters. The configuration file looks like this:

```
! Definition for a CHEAPO-01
DEFINE   = CHEAPO-01  ! Where we have never been before
!
PEN-UP = <ESC>U       ! Pick up the pen
PEN-DOWN = <ESC>D     ! Put the pen down
MOVE     = <ESC>#x#y# ! Move to screen position "x", "y"
MAX-X    = 1024       ! This is the horizontal size
MAX-Y    =  856       ! This is the vertical size
```

This system would allow us to maintain just one program, and at the same time quickly adapt the program for many new terminals. As a matter of fact, if the configuration file is simple, straightforward, and easy to modify, the users may maintain it on their own.

Instead of using a configuration file, the program could ask the user what type of terminal they're using. If that information was not known, the program could request the proper escape sequences and limits. However, that approach assumes that the user knows this information. It also assumes that the user would not call and complain after answering the same questions, day after day, week after week. The configuration file allows the user to provide the information just once.

Three things to keep in mind when programming to read in configuration files:

1. Compare all configurations against programmed limits. If the screen graphics program is set for a maximum size of 1000 x 1000, you wouldn't let the user override the maximums to 1024 x 792. If you do, the first time you try to access the 1001st element of a 1000 element array, the program will either abort, or worse – "unpredictable results".

2. Make the configuration files easy to modify. Use text files with real names for the parameters (like "PEN-UP = <Esc>U" rather than "PARM-2 = 6997"), or provide a program to manipulate a non-text file.

3. Where applicable, keep a library of standard file entries to facilitate the configuration procedure. (E.g., having the "HP2627" and "SUZUKI-4" terminals recognized by name.) Users will send you money in the inter-office mail because they appreciate how easy you make their life! This will also cut back on the chances of problems caused by users making a mistake in configuring common items incorrectly. This in turn will cut back on the "Hey, this stupid thing won't work for me" calls.

CONCLUSION

I wish I could take credit for all these ideas, but I can not. Some are things which were taught to me in classes, some are things which I have seen done in programs that I maintained, some are from seminars and papers I have digested, and a few are my own. I hope that I have gathered them in a cohesive enough form to be of some assistance in your daily work.

Although I have worked almost exclusively with 2nd and 3rd generation languages, and the material presented here is aimed at that arena, most of the concepts are still appropriate if you are using 4 GLs. There are more cases where the language does not allow you to do stupid things, but all that really means is that we will be more clever in our stupidity. Rather than accidently wiping out a single database record, we will be able to accidently wipe out all records which match a certain criteria in multiple databases across many networked computers.

Not all rules apply equally to all programs. The only rule which applies to throwaways is THROW THEM AWAY. It would certainly be a waste to add a large routine to process a configuration file for a small program that has only one thing variable entry. Use common sense.

Programming is still in the twilight area between art and science. Given either extensive enough or restrictive enough criteria, a computer can generate quite reliable programs (Protos is a shining example). However, it still takes a programmer to work out the logic of the system. The day may be approaching when computers will look at areas of need and spew forth program upon program with no human intervention, but even then there will be a market for experienced programmer/analysts. I say this on the observation that even today, when a camera can take a picture of anything from a bowl of fruit, to a landscape, to the night sky, there is still a place for artists who draw bowls of fruit, landscapes, and the night sky.

TITLE: A Means to an End-System Performance is a Journey Rather Than a Destination

AUTHOR: James Harvison
M.S. Ginn Co.
5620 Ager Road
Hyattsville, MD 20782
301-853-4397

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 3007

A PROGRAMMATIC APPROACH TO TAPE LOADING

Paul Ferguson
Wall Street Concepts
90 Broad Street
New York NY 10004
(212) 797-1887

One problem that a system manager or operations manager may have is handling a large number of tapes from various sources. This paper will explain an approach that will automate this process. Once each source is described in a file or data base with pertinent information about the tapes, the process can be fully automated.

The first time you receive a tape to be loaded, the various facts that must be discovered in order to set up the file are:

A: Labeled or Unlabeled
B: If labeled IBM or ANSI label type
C: Data written in ASCII or EBCDIC format
D: Block size of data on tape (from this you can calculate blocking factor if the final record size of the data is known)

MPE provides some of the answers in the display message that occurs when the tape is loaded on the tape drive. The information will be displayed on the console. It will display whether the tape is labeled or not and if labeled whether it is ANSI or IBM. The same data can also be displayed by doing a SHOWDEV ldev# where the ldev# is the number of the tape drive on which the tape is mounted. An example is shown below.

```
:SHOWDEV 7
LDEV   AVAIL     OWNERSHIP      VOLID       DEN    ASSOCIATION

 7     AVAIL                 MLP022(IBM)   6250
```

The rest of the information can be obtained by using a utility in the TELESUP group. RUN TAPESCAN.PUB.TELESUP. A sample of the input and output from that program is shown on page 11 at the end of this paper.

The TAPESCAN output consists of three parts. The first displays the information from the TAPE LABEL, the second displays the first record on the tape, the third displays the size of each block (although TAPESCAN labels each block a record).

If the tape is labeled, the block size IN WORDS can be found in the tape label display. If the tape is unlabeled, the block size in BYTES can be found in the record length portion of the display.

TAPESCAN will display the first record assuming it was ASCII and assuming it was EBCDIC. The (A) and (E) identify which is which. If the tape is ASCII, the EBCDIC record will be garbage, and vice versa. In this case the tape is in EBCDIC format.

We now have the answer to the questions asked above and can go on with the next step which is to store this data for future use, and run the program which will generate the necessary JCL to read the tape and transfer the data to a disc file.

Given the assumption that your shop will process data tapes from a given source regularly, the first step is to store the facts and figures about this company/tape combination for future use.

There are at least two ways that this can be done. If you already have a data base that stores the application data from these tapes, I recommend adding another data set to store the "tape parameter" data. If your application doesn't lend itself to this approach (in fact the application that led to the development of the approach described in this paper did not,) another way is to create an indexed file to store the tape information. I use something about the tape which is unique (such as the broker number, my application being the securities industry) as the key to the file.

In handling large amounts of tape data from outside sources, we have discovered that there are really only two situations that occur. One, the input record automatically defines the output record, and two, the output record varies from tape to tape. The sample program included in this paper is the most general, that is the tape parameters include not only input specifications, but also output file specs. Depending on your application, you can modify this program to meet your needs.

The key that I have used to make handling large amounts of tape data (we receive data from over 200 different sources) is to programmatically generate jobstreams (JCL) to transfer the tape data to disk. This allows my computer operations staff to process incoming tapes without the requirement that they be fully cognizant of the underlying application. Once the data is on disk, the applications programs can easily utilize it as needed.

The sample program that I have included handles both possible cases; that is, if a data file record exists that describes this tape, no operator input (beyond the actual label) is required. If no data file record exists, this program prompts the operator for all the required information, and stores it for later use. In either event, the only operator intervention is to enter the code that has been assigned to the company supplying the tape along with another code that makes this tape unique from others received from the company (I use a mmdd [month day] date code). If a record has been set up for the company, then the program can produce a jobstream for the tape. If there is no record for the company on file, my program will then prompt the operator to answer the other questions that are necessary to set up a record for the company. Once these questions are answered a record is written to the index file (or data base) and the jobstream can be produced.

One additional note, the programmatic approach that I use assumes that you have MPEX available in your shop to squeeze the file down to equal the size of the number of records contained in the file, and therefore builds a file with a capacity greater than the largest file that you may have to transfer from tape to disc.

If MPEX is not available to you, and if the operator knows the number of records on the tape, the program could be modified to ask for and accept the number of records and place that in the file equation (build parameter) in the program. It has been my experience that many of the tapes we receive do not have this data recorded, hence, I use the MPEX approach.

The sample program stores both the input and output parameters for the tape file being processed. If the output parameters are fixed, or are totally dependent on the input parameters, the program could be modified to generate the output parameters based on the input parameters. Depending on the level of skill of your operators it may be worthwhile to reduce the amount of operator input to the bare minimum through this kind of approach.

The sample program provided assumes that only one data file is contained on each tape. If this is not the case, I (or my operators) utilize a somewhat simple, but very effective procedure to make use of the general purpose nature of my programmatic tape handling approach. We lower the JLIMIT before the tape handling program is executed. Then, we abort the JOB that this program streamed. We can then use the editor to edit the file named JOBOUT (created by the program with all the necessary parameters) to make the necessary modifications to read a second file from the tape.

Handling subsequent files is actually quite simple, it only requires a change to one line in the file. If it is an unlabeled tape then you add to the FCOPY line, SKIPEOF=(number of files to skip). If it is a labeled tape then add to the file equation line that describes the tape two commas (,,) followed by the absolute file number that you want to read.

For example if you want to read in the second file on a labeled tape the file equation that should be generated by the program would read:

```
!FILE TAPEIN;DEV=TAPE;REC=-0120,024,F,ASCII;LABEL=ABCFGH,IBM
```

You would modify it to read:

```
!FILE TEPEIN;DEV=TAPE;REC=-0120,024,F,ASCII;LABEL=ABCFGH,IBM,,2
```

you would then stream the modified JOBOUT file.[1]

One important thing I have discovered through the practical experience of processing tapes in various formats from over 200 sources is the importance of designing a method of assigning codes for company names and the individual tapes received from that particular company, so that the name assigned to the disc file has some meaning. In this way, when you invariably have to look through a "LISTF" (listing of the files) you can easily discern what is on each data file.

In my shop we utilize a 4 character broker code, and a 4 digit date code, either MMYY for month year or MMDD for month day. Therefor MERL0190 would be Merrill Lynch's (MERL) January 1990 input tape file.

Depending on the application, there are other modifications which could be made to my sample program. For example, you may wish to read the data into different groups based on the company's profile. Line number A12 would then be changed so that the group name came from the DATA-CLIENT file (tape parameters), and an additional entry would have to be made in the record to store this data. The operator would also have to be interrogated when a new profile was being set up as to which group the data wanted to be placed.

My particular shop processes very large amount of outside data. For that reason we make more use of "batch" processing than many HP shops. We utilize a programmatic approach to generate the

[1] If you save this file, be sure to purge it before using the JCL generation program again. Due to MPE file system rules, the program will abort since the output file for the JCL ("JOBOUT") will have changed in length.

JCL and JOBSTREAMS for almost all our processing. With a few additional modifications, the program outlined here could be made to serve as a general purpose program to generate "batch" job stream. In that case no permanent record would be stored about the company, however, the JCL necessary to read in and process the data would be generated and streamed by the program.

The constraints of this paper, and my talk necessarily limit the general purpose nature of the program I have included. For example, using the Indexed file approach there are no provisions to change the company's "tape file" profile once it is has been set up. No matter whether a data base or indexed file approach is used you must create programs to add, update, and delete profiles from the system. It has been our experience that the overhead involved in creating these programs is much smaller than the overhead involved in insuring the operations staff understands all the "facts and figures" relating to over 200 tapes/month.

One problem we discovered early on was the propensity suppliers had for changing the label option. This can be easily handled by training the operator to recognize what is displayed on the console when a tape is mounted. Of particular importance is noticing when a tape is labeled yet the program does not ask for a label. In this case, something must be wrong.

The reverse situation also holds true. If problems develop, look at the actual disc file (using FCOPY). If it doesn't "look right", then go through the initial steps again to be sure other tape parameters have not changed. We have found that the companies which supply us data tend to change the tape parameters with no regard to the fact that we have to process those tapes.

The same approach, as described here, can be used when you want to write a file from disc to tape. The company's profile record would have to contain the additional data that is necessary to produce JCL to copy the disc file to a tape file with the appropriate attributes on the tape file. The data base approach works very well for creating this set of JCL.

Wall Street Concepts processes literally hundreds of input tapes and an equal or greater number of output tapes. Since the nature of tape processing is batch, I have found that the process of creating JCL from minimal input can be used to insure accurate processing without placing an undue level of reliance on second and third shift operators. In my shop, all the JCL is programmatically generated based on the minimum possible amount of operator input.

SAMPLE PROGRAM

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLEJ.
REMARKS.   This will demonstrate one way to create a JCL stream
           that will transfer data from tape to disc.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

*   This file is where the JCL will be written.
     SELECT JOBCARDS ASSIGN TO "JOBOUT".

*   This is the INDEXED file where the pertinent data is stored.
     SELECT DATA-CLIENT ASSIGN TO "CLIENT" ORGANIZATION IS INDEXED
        ACCESS MODE RANDOM RECORD KEY CLIENT-KEY.

DATA DIVISION.
FILE SECTION.
FD  JOBCARDS LABEL RECORDS ARE OMITTED.
01  CARD1                      PIC X(72).

FD  DATA-CLIENT LABEL RECORDS ARE OMITTED.
01  DATA-CLIENT-REC.
    05 CLIENT-KEY              PIC X(04).
    05 TAPE-FORMAT             PIC X.
    05 LABEL-INFO              PIC X.
    05 RECORD-LENGTH-IN        PIC 9999.
    05 BLOCK-SIZE-IN           PIC 999.
    05 RECORD-LENGTH-OUT       PIC 9999.
    05 BLOCK-SIZE-OUT          PIC 999.

WORKING-STORAGE SECTION.
01 MISC VARIABLES.
    05 JK                      PIC 99.

*  This is where the data will go when the operator is asked to
   give name of tape being read.
    05 CLIENT-INPUT.
       10 CLIENT-KEY-IN        PIC X(04).
       10 LABEL-INFO-IN        PIC X(04).

01 JCL-LINES.

    05 FILLER                  PIC X(72) VALUE
    "!JOB LOADTAPE,user.account,group".
```

```
*    This is where we make the file description for the output
     file.

   05 FILLER.
      10 FILLER              PIC X(07) VALUE "!BUILD".
      10 FILE-RECEIVE-1      PIC X(08).
      10 FILLER              PIC X(15) VALUE ";DEV=DISC;REC=-".
      10 RECORD-LENGTH-1     PIC 9999.
      10 FILLER              PIC X VALUE ",".
      10 BLOCK-FACTOR-1      PIC 999.
      10 FILLER              PIC X(34) VALUE
       ",F,ASCII;DISC=500000,32".
*    On the above statement make the size greater than the largest
     amount of data that may be on a tape.

*    This is where we make up the file description of the tape
     input file.
   05 FILLER.
      10 FILLER              PIC X(28) VALUE
      "!FILE TAPEINP;DEV=TAPE;REC=-".
      10 REC-LENGTH-IN       PIC 9999.
      10 FILLER              PIC X VALUE ",".
      10 BL-SIZE-IN          PIC 999.
      10 FILLER              PIC X(08) VALUE ",F,ASCII".
      10 LABEL-CONST         PIC X(07) VALUE SPACE.
      10 TAPE-LABEL-IN       PIC X(06).
      10 IBM-ASCII           PIC X(15) VALUE SPACE.
   05 FILLER.
      10 FILLER              PIC X(24) VALUE
       "!FCOPY FROM=*TAPEINP;TO=".
      10 FILE-RECEIVE-2      PIC X(08).
      10 EBCDIC-ASCII        PIC X(40) VALUE SPACE.

*    If you have MPEX then you can insert the following lines of
     code to squeeze the file into the smallest amount of space
     necessary, otherwise skip down to the !TELLOP line.

   05 FILLER                 PIC X(72) VALUE "!MPEX".
   05 FILLER.
      10 FILLER              PIC X(09) VALUE "!ALTFILE ".
      10 FILE-RECEIVE-3      PIC X(08).
      10 FILLER              PIC X(55) VALUE ";SQUEEZE".
   05 FILLER                 PIC X(72) VALUE "EXIT".

   05 FILLER                 PIC X(72) VALUE
    "!TELLOP The tape has been read in please dismount".
   05 FILLER                 PIC X(72) VALUE "!EOJ".
```

```
01  JCL-LINE REDEFINES JCL-LINES.
    05 EACH-LINE              PIC X(72) OCCURS 9 TIMES.

*    The next group of statements are necessary so that the
     jobstream will be streamed as soon as all questions asked by
     this program are answered by the operator.

01  COMMAND-IMAGE.
    05 ACTUAL-COMMAND         PIC X(70).
    05 CARRIAGE-RETURN        PIC X VALUE %15.

01 COMMAND-CI-ERROR           PIC 9(04) COMP VALUE 0.

01 COMMAND-FS-ERROR           PIC 9(04) COMP VALUE 0.

01 CI-ERR                     PIC 9(05) VALUE 0.

01 JOBSTREAM                  PIC X(70) VALUE "STREAM JOBOUT".

*    The following is the area that will receive the answer's to
     the questions if a new record must be set up.

01 INPUT-DATA.
   05 RECORD-SIZE             PIC 9999.
   05 BLOCK-SIZE              PIC 999.


01 OUTPUT-DATA.
   05 RECORD-SIZE-O           PIC 9999.
   05 BLOCK-SIZE-O            PIC 999.

PROCEDURE DIVISION.
A1.

     OPEN I-O DATA-CLIENT OUTPUT JOBCARDS.
*    We are now going to ask for the CLIENT INFO and Tape
     identification data.

     DISPLAY "Enter Client ID (4 chars) and Tape Id (4 chars)"
     ACCEPT CLIENT-INPUT

*    If when we read the DATA-CLIENT file there is no entry we then
     ask more questions and set up an entry or exit from program
     to find out necessary data and then run program again.

     READ DATA-CLIENT INVALID KEY
        PERFORM SETUP-NEW-RECORD THRU SETUP-EXIT.
```

```
*    At this point we have the necessary data available to fill in
     the rest of the data in the WORKING-STORAGE SECTION.

     IF TAPE-FORMAT = "E" MOVE ";EBCDICIN" TO EBCDIC-ASCII.
     IF LABEL-INFO = "N" NEXT SENTENCE
     ELSE MOVE ";LABEL=" TO LABEL-CONST
          DISPLAY "Enter tape label 6X" ACCEPT TAPE-LABEL-IN
          IF LABEL-INFO = "A" MOVE "ANS" TO IBM-ASCII
          ELSE MOVE "IBM" TO IBM-ASCII.
     MOVE RECORD-LENGTH-IN TO REC-LENGTH-IN
     MOVE BLOCK-SIZE-IN TO BL-SIZE-IN
     MOVE RECORD-LENGTH-OUT TO RECORD-LENGTH-1
     MOVE BLOCK-SIZE-OUT TO BLOCK-FACTOR-1
     MOVE CLIENT-INPUT TO FILE-RECEIVE-1 FILE-RECEIVE-2
     FILE-RECEIVE-3.

*    We now have all the data fields filed in the WORKING-STORAGE
     SECTION and can now write the JOBCARDS file.

     PERFORM READ-WRITE VARYING JK FROM 1 BY 1 UNTIL JK > 9.
     CLOSE DATA-CLIENT JOBCARDS
     MOVE JOBSTREAM TO ACTUAL-COMMAND
     CALL INTRINSIC "COMMAND" USING COMMAND-IMAGE
                                    COMMAND-CI-ERROR
                                    COMMAND-FS-ERROR.
     IF COMMAND-CI-ERROR <> 0 AND COMMAND-FS-ERROR <> 0
          DISPLAY " ERROR - The STREAM Command failed :"
          MOVE COMMAND-CI-ERROR TO CI-ERR
          DISPLAY " Error is " CI-ERR.

     STOP RUN.

READ-WRITE.

     MOVE EACH-LINE (JK) TO CARD1
     WRITE CARD1.

SETUP-NEW-RECORD.

*    This is where we get the data necessary to create a record for
     a new client.

B1.

     DISPLAY "If the tape is labeled enter I for IBM, A for ANSI,
-      " N if no label".
     ACCEPT LABEL-INFO
     IF LABEL-INFO = "I" OR "A" OR "N" NEXT SENTENCE ELSE
     DISPLAY "You must enter either I, A, or N" GO TO B1.

B2.

     DISPLAY "Enter A if data is ASCII or E if data is EBCIDIC"
     ACCEPT TAPE-FORMAT
     IF TAPE-FORMAT = "A" OR "E" NEXT SENTENCE ELSE
     DISPLAY "The response must be A or E" GO TO B2.

B3.

     DISPLAY "Enter input record length 9999, block size 999"
     ACCEPT INPUT-DATA
     IF INPUT-DATA NUMERIC NEXT SENTENCE ELSE
     DISPLAY "You must enter 7 digits" GO TO B3.
*    You could put other data checks in here if you wanted.

     MOVE RECORD-SIZE TO RECORD-LENGTH-IN
     MOVE BLOCK-SIZE TO BLOCK-SIZE-IN.

B4.

     DISPLAY "Enter output record length 9999, block size 999"
     ACCEPT OUTPUT-DATA
     IF OUTPUT-DATA NUMERIC NEXT SENTENCE ELSE
     DISPLAY "You must enter 7 digits" GO TO B4.
     MOVE RECORD-SIZE-O TO RECORD-LENGTH-OUT
     MOVE BLOCK-SIZE-O TO BLOCK-SIZE-OUT.
     WRITE DATA-CLIENT-REC INVALID KEY DISPLAY
      "Bad write of new record tell Operation Manager"
     DISPLAY "The key of the record is " CLIENT-KEY
     CLOSE DATA-CLIENT JOBCARDS
     STOP RUN.

SETUP-EXIT.

     EXIT.
```

```
LDEV  AVAIL         OWNERSHIP           VOLID          DEN   ASSOCIATION
 RUN TAPESCAN.PUB.TELESUP

--------------------------------------------------------------------------
TAPESCAN (A.00.02)                        MPE V - HP3000 System Support Tool
MON, MAY 14, 1990,  9:26 AM          Copyright Hewlett Packard Co. 1983,1986

TAPESCAN is designed for use by Hewlett Packard Support personnel only.
      HP IS NOT LIABLE FOR DAMAGES RESULTING FROM UNAUTHORIZED USE
--------------------------------------------------------------------------

 ENTER THE LISTING DEVICE ( * FOR $STDLIST) *
 IS THIS A LABELED TAPE (Y/N) ? Y
 ENTER THE TAPE LABEL ID 001557
 IS IT IBM OR ANSI FORMAT ? IBM
 TAPE WAS MOUNTED ON LDEV 7
 SHOULD THE CONTENTS OF ANY USER LABELS BE PRINTED ? Y
 DO YOU WANT ALL RECORD LENGTHS LISTED?
Y
 DO YOU WANT TO PRINT THE FIRST RECORD OF EACH FILE ? Y
 PRINT RECORD IN ASCII   ? Y
 PRINT RECORD IN EBCDIC ? Y
 PRINT RECORD IN OCTAL   ? N
 PRINT RECORD IN DECIMAL? N
 ----------------------------------------------------
 ! TAPE LABEL INFORMATION DISPLAY(TAPESCAN A.00.02)!
 !--------------------------------------------------!
 !TAPE SET ID: 001557           VOLUME ID:001557    !
 !LABEL TYPE : IBM       EXPIRATION DATE:%000000    !
 !REEL # 1             SEQUENCE # 1                 !
 !BLOCK SIZE =  4000 WORDS    REC SIZE=  4000 WORDS !
 !BLOCKING FACTOR=    1 RECORDS/BLOCK               !
 !FILE NAME:TAPE                CREATED:%132203     !
 ----------------------------------------------------
(A)..@.........@......@.....@....@....@.@...@...@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
(E)11 009293101 AIRCOA HOTEL PTNS LPCL A DEP UTS

RECNUM         RECORD LENGTH IN BYTES

     1:       8000     8000     8000     8000     8000     8000     8000     8000
     9:       8000     8000     8000     8000     8000     8000     8000     8000
    17:       8000     8000     8000     8000     8000     8000     8000     8000
```

A SOLUTION FOR SALES MANAGEMENT'S NIGHTMARES: AN OMNIDEX ENVIRONMENT

Marie Gannon, Systems Analyst
Alan A. Tainsh, Director, Management Information Services

United Liquors Ltd.
One United Drive
West Bridgewater, MA 02379
(508) 588-2300

I. INTRODUCTION

United Liquors Ltd. of West Bridgewater, Massachusetts is currently celebrating its fifty-fifth anniversary. United was founded in Boston in 1935 following the repeal of prohibition. At that time the operation consisted of one truck and four employees. The development of the Connoisseurs of Boston Fine Wine division, and the extensive beer business at United Liquors Ltd. is typical of the vision and leadership of our Chairman, Mr. A. Raymond Tye.

United Liquors Ltd. is now one of the largest and most successful alcoholic beverage distributorships in the Northeast. From our humble beginnings, we now have three warehouses with more than 500,000 square feet of warehouse space. We deliver over 7,000 different products to over 12,000 customers with over 100 trucks. Our 140 salesmen are just a piece of our workforce of over 600 employees.

The company made a decision in 1983 to bring in Hewlett Packard to help manage their complex business. A single HP300 Series 68 was purchased, and that configuration has grown to include two HP3000 Series 70's as well as an HP300 Series 925 with over 120 terminals and seven gigabytes of disk. To help run this sophisticated system there are eighteen employees in the Management Information Services department.

II. DESCRIPTION OF THE OLD SALES REPORTING SYSTEM

Prior to 1989, United Liquors had a very basic sales reporting system in place. It consisted mainly of a database containing one very large dataset. This system was fed daily from invoice transactions. It had a couple of on-line inquiries of limited flexibility for users, and it had a few reports. But the old system was limited in many ways:

INSUFFICIENT HISTORICAL DATA

As big as the Sales dataset was, it held only six months of information. Any history report requesting a prior year became time-consumning because the historic database had to be restored from an off-site location before the report could be run. This was also a problem because of the lack of available disc space on the system.

BULK REPORTING

All the reports were bulky detail reports. Some users needed to see only the final totals, while others were concerned only with a cross-section of data, usually consisting of a few pages. These reports took a long time to run, with usually a 24-hour turnaround. Some programs used long chain paths to retrieve their data, while other programs read just the whole dataset sequentially.

NO FLEXIBILITY

Sales reports were not flexible in their selection and sort criteria. Users were limited to a few key items such as item number, customer number or salesman number. Because the reports did not do any comparative reporting, many users had to request two or more reports to get the data they required. In addition, there were some reports that were corporate level and the data had to be combined from two computers before reporting.

III. THE NEED FOR CHANGE

Due to increasing business demands, it became apparent that the old system could not meet the needs of its users. Information was being provided to the users in an unusable format. In short, it was time to provide users with a better reporting system. Corporate management came up with system requirements for the MIS team:

PRODUCE MORE SUMMARY REPORTS

Eliminate bulky detail reports in favor of more specific and summarized reports. This would cut down on the time it took to produce the report as well as the size of the report itself. Senior management and sales managers could better focus on the information provided to them.

FOCUS REPORTING TO SPECIFIC SEGMENTS OF THE USER COMMUNITY

Create more types of reports in order to serve the different user communities: Suppliers, Finance, Sales Management, Product Management and Salesmen.

ALLOW MORE FLEXIBILITY

Allow better flexibility in selection and sorting of reports. Users should have the capability to define selection criteria.

STORE MORE INFORMATION FOR COMPARATIVE ANALYSIS

The new system was designed to store three fiscal years of history. Management wanted enough information to be able to make better day-to-day and long-range decisions based on historic information.

FLEXIBLE ON-LINE INQUIRIES

Use the pyramid approach on inquiries by starting at the company level, then go into each level of detail, i.e., company, branch, salesmen within branch, customer within salesmen. Inquiries were also needed for all combinations of brands, sizes, ethnic categories, license types, etc. This information must be available for the past and current fiscal periods, as well as for last year's future fiscal periods for forecasting inquiries. All of these on-line inquiries must be within acceptable response time parameters.

IV. THE NEW SALES REPORTING SYSTEM UTILIZING OMNIDEX

HOW THE SALES REPORTING SYSTEM HAS HELPED THE SALES STAFF USERS

To meet the system requirements presented to us by our sales system users, we purchased an HP3000 Series 925 and an OMNIDEX software package. The Series 925 with 40 megs of memory and currently 4.19 gigabytes of disc space provided us with sufficient memory, disc space and the new technology needed to design and code the new sales reporting system. With this added hardware and software and a new design of our existing sales system, we were able to provide the sales staff with several reporting options.

Omnidex, a third-party keyword retrieval tool written by Dynamic Information Systems Corporation (DISC), provides us with a COBOL and QUIZ interface for multiple keyword retrival, and the ability of range and wild card retrivals. This type of flexibility enabled us to eliminated most bulk reports, and enabled sales management to view data at several detailed or summary levels. The flexibility of Omnidex satisfied all of our sales system users.

There are different levels of users for this system ranging from a management level to a vendor/supplier level, each level of users having requirements for different reasons and at different reporting levels.

The following examples explain the need for information at a sales management level.

FORECASTING

The new sales reporting system provides an on-line forecasting screen utilizing Omnidex calls. This screen allows sales management to view data for any selected product down to which salesman is responsible for a customer, as well as which products the salesman sold to that customer. This screen provides case and dollar volume for next month last year. Sales management can now set quotas for salesmen for future month's sales down to an item level.

TRACKING SALES

Now that a quota has been established, sales management can track this information via an on-line screen utilizing OMNIDEX which allows sales information to be viewed for the current fiscal month and current fiscal year with comparisons against last year. The current month information is updated on a daily basis for an up-to-date, accurate picture of our business. This type of Omnidex on-line data retrevial allows sales management to make any day-to-day and long-term decisions.

GROSS PROFIT ANALYSIS

Sales management now also has the ability to view sales information for gross profit analysis, which is a major part of our business. The liquor distribution industry is an extremely competitive business; therefore, gross profit must be monitored closely. With our new system gross profit can be tracked yearly, monthly and daily.

INDUSTRY COMPARISON

Industry comparison is also a big part of our business. Since we now have the flexibility of batch reporting and on-line access, it is possible to gather the information needed to compare sales against our competitors. This helps management make decisions on which products and in what areas the salesmen must focus. The ability to compare this type of information gives us an edge in this very competitive business.

DAY-TO-DAY DECISIONS

Reviewing sales productivity is a good example of day-to-day utilization of the system by sales management. Each manager has quotas on specific items for that month, and monitors those items by salesmen. The manager inquires into the system to see how each salesman is performing against last year, as well as this month's quota. The previous year's sales are based on what the salesmen's customers bought during that period. If a salesman is reassigned on an account, the history follows the newly-assigned salesman. This gives the responsibility of the account to the salesman. The manager then can see what salesmen and which accounts need to have his attention. The system can also point out high-volume accounts that can benefit from our volume discount prices.

ROUTEBOOKS

The salesmen have been provided with a quarterly "Routebook" report containing the following information:

- All the customers for which they are responsible

- Case sales of all the items these customers purchased for the upcoming three months last year

The routebook provides the salesmen with a list of exactly what items a customer bought this quarter by month for last year's sales. The report is given to the salesmen quarterly for planning purposes. Volume discounting is based on buying specific quantities of the same product during a month. This routebook can assist the salesmen in showing the customer last year's sales to prove that the volume necessary for discounting can be sold by that customer during that month. It can also be useful to show how a replacement item may sell against last year's item. Another area of concern is to show the salesmen products that the customer is not buying. This report has been very valuable to the salesmen in helping them with their performance.

SUPPLIERS

Suppliers also make use of our system for sales information and market analysis. Since our batch and on-line inquiries are now flexible, product management may make inquiries of products sold under specific suppliers. Some products sell better in some areas than others. Suppliers like to analyze why this is true and, therefore, focus on certain locations for specific products.

Those are just three users of the sales reporting system, and all of their goals have been reached. In the words of one person in our senior sales management staff, "There has been a 100% turnaround in all sales reporting."

DATABASE DESIGN AND ANALYSIS

The database consists of sixten automatic and manual masters, six of which are Omnidexed; and five detail datasets, one of which is Omnidexed.

DETAIL DATASETS

The following detail datasets contain three years of historical data including some or all of the following information:

- -Cases Sold
- -Dollar Volume
- -Cost of Goods
- -Budget Dollars

SALES-HIST-C:	Summarized at a salesmen/customer/item level, over 1,000,000 entries
SALES-HIST-S:	Summarized at a salesmen/item level, over 200,000 entries
SALES-HIST-I:	Summarized at an item/sales office/premise level, over 30,000 entries
CUST-SALINK:	Summarized at salesmen/customer/product ownership level, over 30,000 entries
SM-BUDGET:	Summarized at a salesmen level, over 150 entries

SALES-HIST-C is an Omnidexed detail dataset. A new application called for a path to be added to SALES-HIST-C. We decided to make this an Omnidex path. If we added an Image path we would have had to make several system and programming changes such as invoking ADAGER to add a sixteen character path to each record, changing our dictionary, schema and copylib, and recompiling all COBOL and Powerhouse applications. Since we added an Omnidex path, the only required changes consisted of two job streams, the Omnidex install and the Buildfast. A reinstall of Omnidex was then required which was a simple process.

MANUAL MASTERS

One major advantage of an Omnidex environment is the ability to have several paths on a manual master. When designing this system, analysts may have thought of changing these manual masters to detail datasets which would allow for multiple paths, but we would not have the flexibility that Omnidex gives us, such as wild card retrievals and the ability to use more than one path for data retrieval. For example, you may wish to view sales for a product, and within this product you may wish to view one size. Omnidex gives you the flexibility to further qualify a record along several paths until all your selection critera have been met.

SALES DATA AT A SALESMEN RESPONSIBILITY LEVEL

Currently three salesmen are assigned to a customer, one salesman sells liquor and cordial products, one sells beer and water products, and one salesman sells wine products. The salesmen currently assigned to a customer are responsible for all sales to that customer. Responsibility of these customers can change for some of the following reasons:

-A salesman retires or resigns from the company

-The currently-assigned salesman cannot meet
the quotas established by sales management

-A new salesman has been hired

-A restructuring of sales territories

One goal of the new sales system was to have all historical sales follow the currently-assigned salesman to a customer for each of the 3 product classes. This type of reposting is run on a weekly basis.

For example:

Assume the date: July 21, Wednesday

Customer-Master:

	Customer-Number:	999999
(LIQ)	Salesman-Number1:	L100
(BEER)	Salesman-Number2:	B100
(WINE)	Salesman-Number3:	W100

July Sales:

SALESMAN	ACCOUNT	PRODUCT	CASES
L100	999999	LIQUOR	10
B100	999999	BEER	15
W100	999999	WINE	5

On July 22 salesman L100 resigned from the company and salesman number L200 was assigned to that customer.

On July 23 sales management took salesman B100 off of customer 999999 because he could not meet his quota. Salesman B200 was assigned to that customer.

Date July 24, Saturday

Revised Customer-Master

	Customer-Number:	999999
(LIQ)	Salesman-Number1:	L200
(BEER)	Salesman-Number2:	B200
(WINE)	Salesman-Number3:	W100

July sales after reposting historical data to follow the currently-assigned salesman:

SALESMAN	ACCOUNT	PRODUCT	CASES
L200	999999	LIQUOR	10
B200	999999	BEER	15
W100	999999	WINE	5
L100	999999	LIQUOR	0
B100	999999	BEER	0

This weekly reposting helps sales management track sales. Before this application was implemented, sales management would track several salesmen for a customer's one product class. Adding this piece of software to the design of the sales reporting system has cut down on the time that sales management used for tracking and forecasting inquiries.

PROGRAMMING ISSUES

The sales system currently contains approximately 40 reports, most of which are written in Quiz and Omniquiz. The system also consists of eleven on-line inquiries, of these, two are written in Quiz and two utilize Omnidex calls. The sales system users can access all of the on-line inquiries via menu master which is a third-party, menu-dirven tool that helps to control the user environment. The reporting menus are currently written in Quick and also utilize command files.

All of the historical datasets are posted via an invoice detail file on a nightly basis. These posting programs are written in cobol. SALES-HIST-C which is an Omnidexed detail dataset utilizes Omnidex calls.

The transition of Image calls to Omnidex calls is one of ease. There are basically few programming changes. The Omnidex calls replace regular Image calls. For example, a "DBUPDATE" only updates regular Image datasets, whereas a "DBIUPDATE" updates Omnidex indexes as well as Image datasets.

COBOL

Here are some of the changes that must take place in COBOL:

- -The DBSTATUS area must be expanded to accommodate Omnidex calls

- -Additional calling modes may be necessary

- -The regular Image calls must be changed to Omnidex calls

"ODXFIND" is an Omnidex call which in some ways is similar to a "DBFIND" in that it qualifies records along a certain path. Without this Omnidex path most inquiries would have to use a longer path or a serial read.

The following illustration compares Image and Omnidex differences:

The following example must retrieve "A" and "Z" data only

"A," "B," "C" are Image Keys

"X" and "Z" are Omnidex Keys

DETAIL DATASET RETREVIAL COMPARISON

```
        IMAGE                                       OMNIDEX
                                ________
1)    DBFIND "A"               . A,X .     1)    ODXFIND "A"
      4 Records Qualify        . A,Z .           4 Records Qualify
                               . B,X .
2)    DBGET (MODE 5)           . A,X .     2)    ODXFIND "Z"
      4 Records Retrieved      . C,Z .           2 Records Qualify
                               . A,Z .
                               --------
                                          3a)    ODXGET
                                                 Retrieves Image
                                                 Record #'s

                                          3b)    DBGET (Mode 4)
                                                 Using Image
                                                 Record #'s
```

The following example must retrieve "Z" data

MANUAL MASTER RETRIEVAL COMPARISON

```
                                ________
1)    DBGET (Mode 2)           . A,Z .     1)    ODXFIND "Z"
    -3 Records Retrieved       . B,X .           2 Records Qualify
                               . C,Z .
                               --------
                                           3a)   ODXGET
                                                 -Retrieves Image
                                                  Keys

                                           3b)   DBGET (Mode 7)
                                                 -Many Related
                                                  Records Can Be
                                                  Retrieved
```

The above examples prove what a powerful tool Omnidex is. The Image manual master must be read serially to retrieve all the data, the detail dataset must go down a large chain. To retrieve the requested data the COBOL program must programmatically exclude all data that does not meet the selection criteria. On the other hand, Omnidex retrieves only the data that falls within the selection criteria.

OMNIQUIZ

Several reports on the sales reporting system are written in Omniquiz. It serves as a powerful interface between Quiz and Omnidex. The installation and programming of Omniquiz is also fairly simple. Disc provides you with the tools needed to add the required Omnidex datasets to the database and the Powerhouse Qschema.

Omniquiz source code is similar to quiz code, therefore requiring minor modifications. Quiz code requires four basic statements: Access, Define, Select and Report:

```
ACCESS setname(s)
DEFINE...       To Prompt for Input
SELECT IF...    To Specify Selection Criteria
REPORT item(s)
```

Omniquiz required commands are: Access, Choose, Define and Report:

```
ACCESS set-odx
CHOOSE image-si
DEFINE...       "ODXSELECT"
DEFINE...       "ODXGO"
REPORT items
```

For example you may want to see customers with specific ethnic categories.

<u>QUIZ</u>

```
Access Customer
Define Ethnic-Prompt CHAR*2 = PARM PROMPT &
 "Please Enter Ethnic Category: "
Select if Ethnic-Prompt = Ethnic-CAT
Report All
```

<u>OMNIQUIZ</u>

```
Access Customer-ODX
Define Ethnic-Prompt CHAR*2 = PARM PROMPT &
 "ODXSELECT 1 'Please Enter Ethnic Category: ' "
Define ODX-GO CHAR*2 = PARM PROMPT &
 "ODXGO ' OK to Proceed (Y/N) ' "
Report All
```

The access statement in the Omniquiz example refers to the OMNIDEX INDEX, which is defined like a KSAM file in the Qshema. A choose is performed on the image path search item for that set. These calls are interpreted by the Omnidex interface which resides in the SL or XL library in the group where the reports are executing, and used internally to perform the appropriate selections. A define satement is used to prompt for keywords via a new parameter called "ODXSELECT," which eliminates the need for a "SELECT" statement.

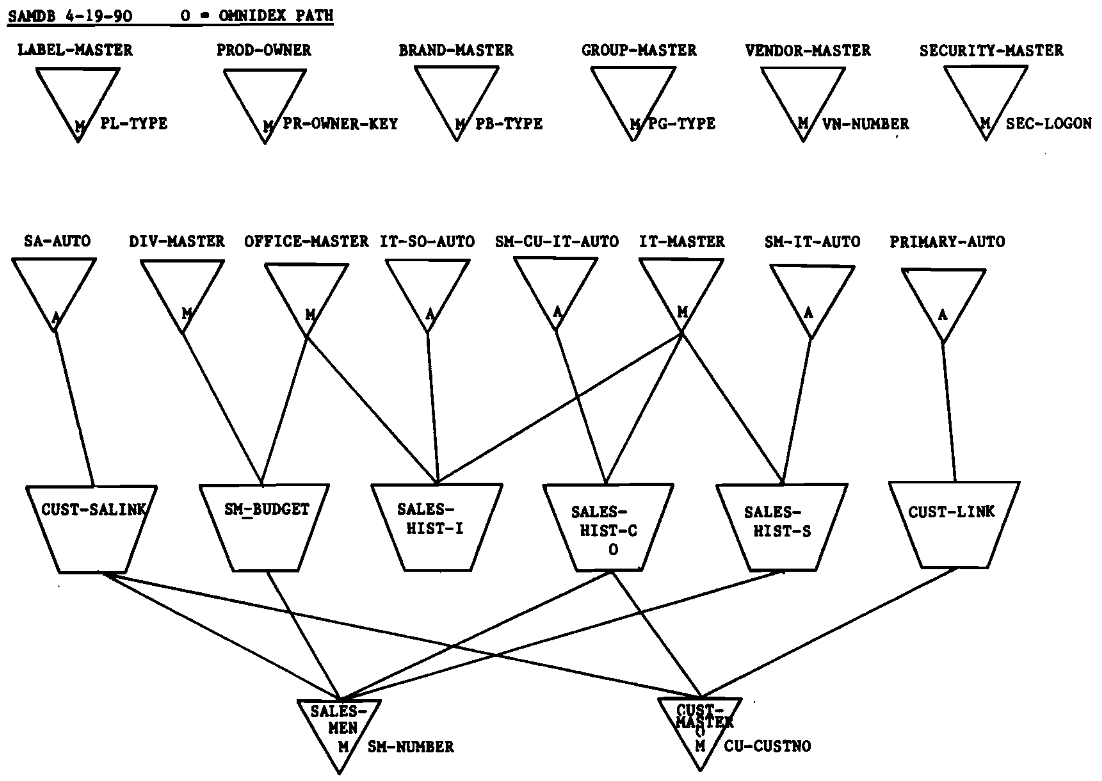
Omnidex requires a control file called "ODXQUIZ" which must contain the definitions of the fields that will be used for the "ODXSELECT," (Disc provides you with a tool called "ODXPRO" that generates this file). ODXPRO also generates a file consisting of file definitions and a file consisting of record definitions, which will be added to your Qschema.

FUTURE DEVELOPMENT

The sales reporting system was designed to be the first piece of an Executive Support System. We are working with Hewlett Packard to utilize New Wave HPAccess. New Wave will extend the value and capabilities of our personal computers. We will be able to easily access, manipulate, combine and analyze information from multiple application across PC and HP3000 platforms. Using New Wave and HPAccess will allow us to automatically take data from multiple sources, combine and present it in a final graphic form.

CONCLUSIONS

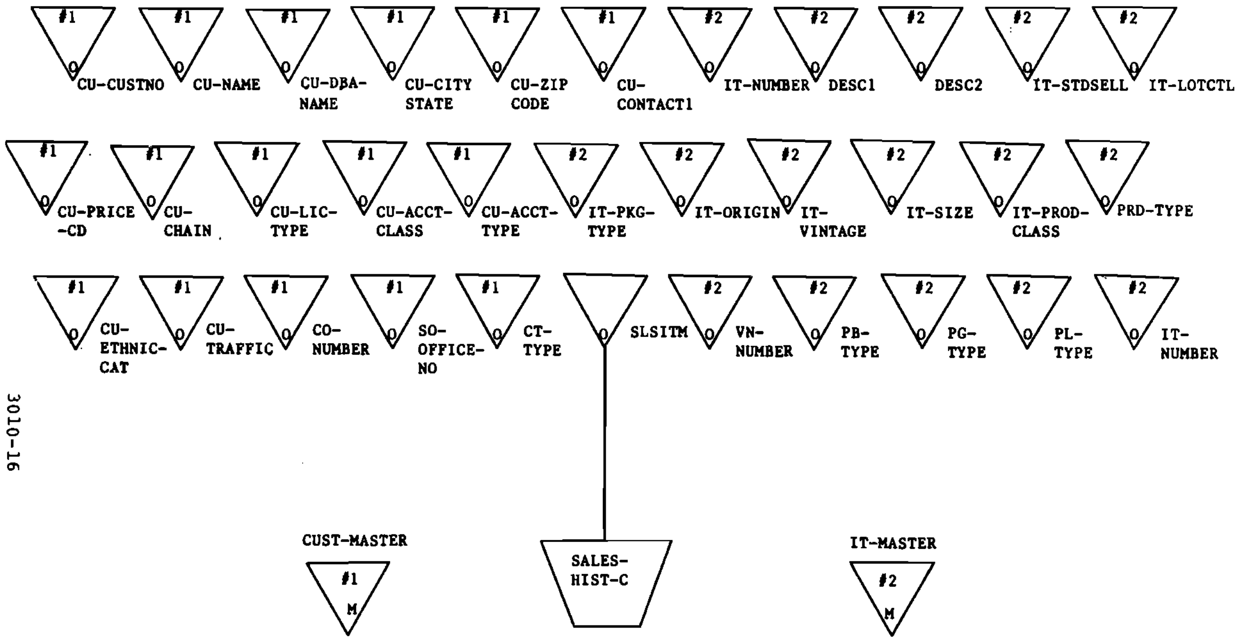
The new sales system is now able to respond quickly to changing business demands. The combination of the latest HP computer technology, the Omnidex retrieval system and the new sales reporting system has really been the winning success story at United Liquors. As one senior executive was quoted as saying, "I don't know how we made decisions before we had this system."



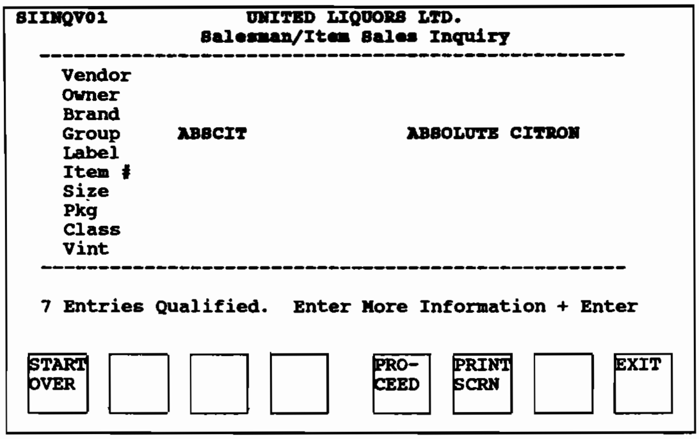
A SOLUTION FOR SALES MANAGEMENT'S NIGHTMARES

3010-15

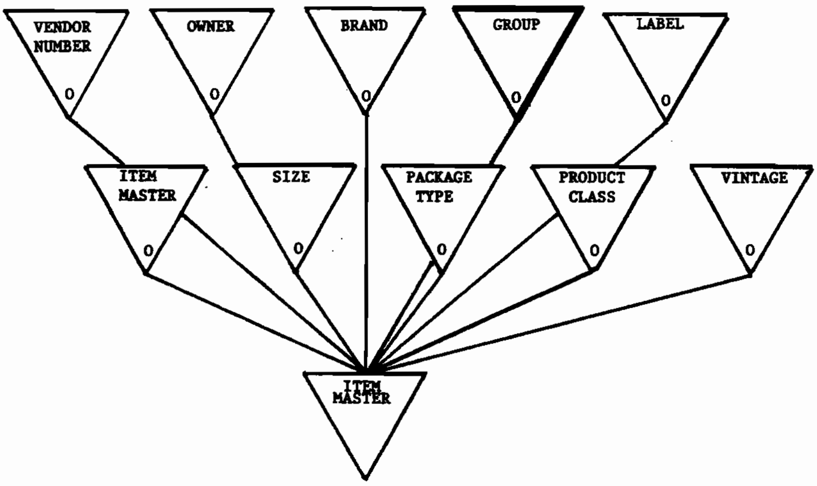
SAMDB 4/19/90 OMNIDEX LAYOUT



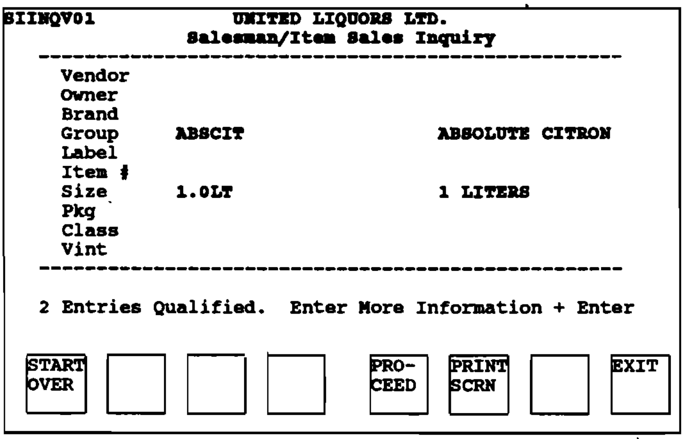
A SOLUTION FOR SALES MANAGEMENT'S NIGHTMARES



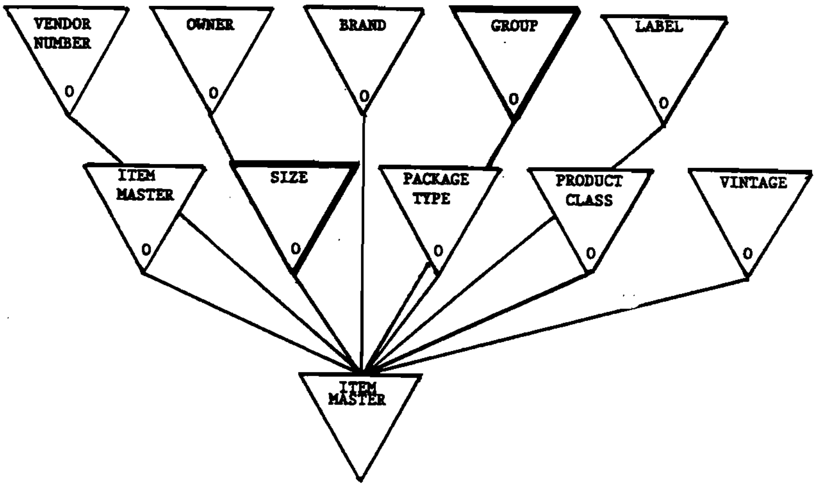
```
SIINQV01              UNITED LIQUORS LTD.
                  Salesman/Item Sales Inquiry
   ----------------------------------------------------
    Vendor
    Owner
    Brand
    Group      ABSCIT                    ABSOLUTE CITRON
    Label
    Item #
    Size
    Pkg
    Class
    Vint
   ----------------------------------------------------

   7 Entries Qualified.  Enter More Information + Enter
```



A SOLUTION FOR SALES MANAGEMENT'S NIGHTMARES



```
SIINQV01                UNITED LIQUORS LTD.
                     Salesman/Item Sales Inquiry
    --------------------------------------------------
     Vendor
     Owner
     Brand
     Group    ABSCIT                 ABSOLUTE CITRON
     Label
     Item #
     Size     1.0LT                  1 LITERS
     Pkg
     Class
     Vint
    --------------------------------------------------

   2 Entries Qualified.  Enter More Information + Enter
```

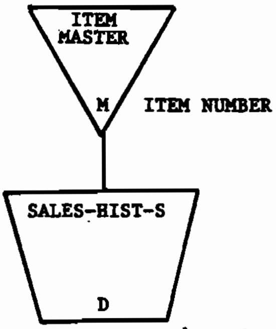


A SOLUTION FOR SALES MANAGEMENT'S NIGHTMARES

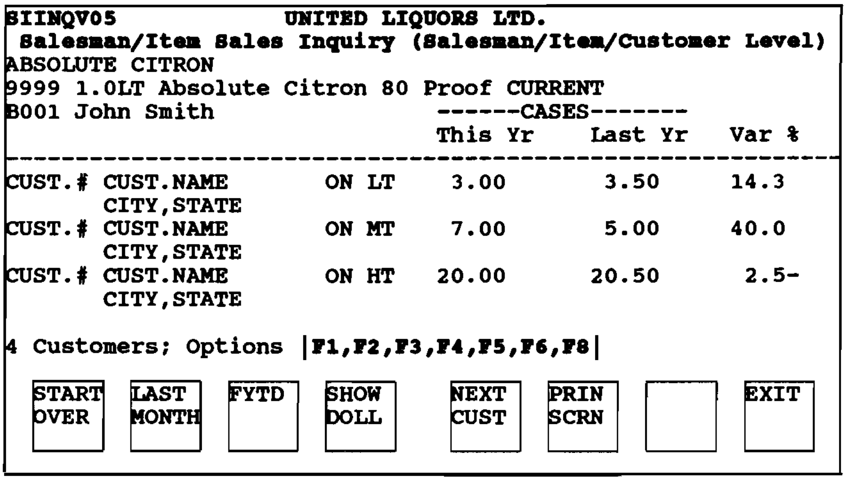
```
SIINQV02               UNITED LIQUORS LTD.
     Salesman/Item Sales Inquiry (Sales Office Level)
ITEM NAME HERE
                                        CURRENT
                       This Yr  Last Yr          This Yr Last Yr
Sales Division          Cases    Cases   Var %   Dollars Dollars
------------------------------------------------------------------
 1  DIV01 1)UNITED     150.00   140.00    7.1     27,000  25,200
          2)BOSTON     125.00   110.00   13.6     22,500  19,800
          3)CENTRAL     25.00    30.00   16.7-     4,500   5,400

[1]
Key # (To Show Salesmen Detail) + Enter
```

```
S11NQV03               UNITED LIQUORS LTD.
       Salesman/Item Sales Inquiry (Salesman Level)
ABSOLUTE CITRON
                                        CURRENT
                       This Yr  Last Yr          This Yr Last Yr
Salesman                Cases    Cases   Var %   Dollars Dollars
------------------------------------------------------------------
 B001 John Smith        70.00    68.00    2.9     12,600  12,240
 B002 Dan Toupin        45.00    42.00    7.1      8,100   7,560
 B003 Jim Turner        35.00    30.00   16.7      6,300   5,400

[1]
Key # (To Show Item Detail) + Enter
```

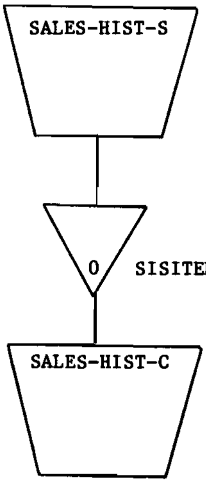
```
SIINQV04               UNITED LIQUORS LTD.
     Salesman/Item Sales Inquiry (Salesmen/Item Level)
ABSOLUTE CITRON                         CURRENT
B001 John Smith        This Yr  Last Yr          This Yr Last Yr
                        Cases    Cases   Var %   Dollars Dollars
 1   9999 1.0LT         30.00    29.00    3.4      5,400   5,220
 2   8888 1.0LT         40.00    39.00    2.6      7,200   7,020

[1]
Key # (To View Cust. Info.) + Enter
```



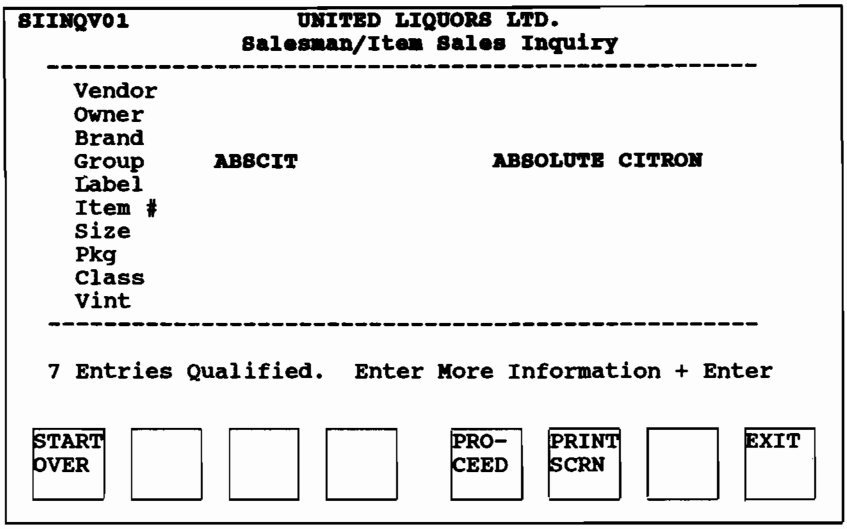
A SOLUTION FOR SALES MANAGEMENT'S NIGHTMARES



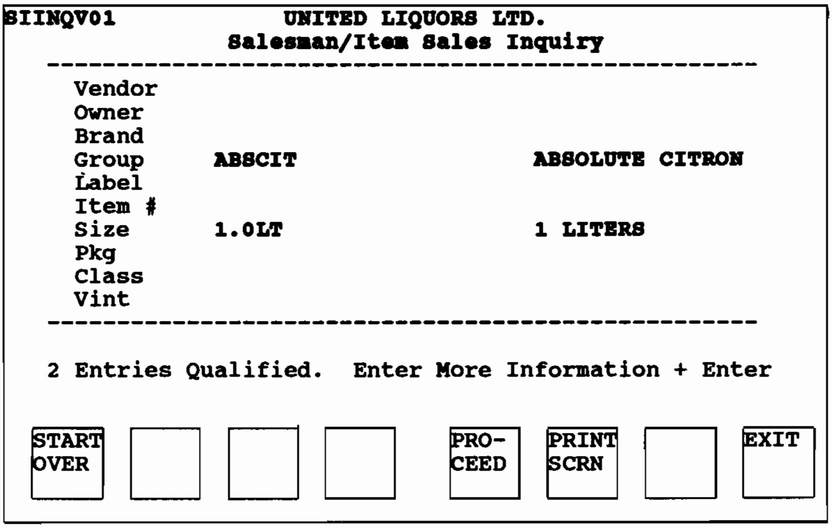
```
SIINQV05                UNITED LIQUORS LTD.
 Salesman/Item Sales Inquiry (Salesman/Item/Customer Level)
ABSOLUTE CITRON
9999 1.0LT Absolute Citron 80 Proof CURRENT
B001 John Smith                      -------CASES-------
                                     This Yr    Last Yr   Var %
------------------------------------------------------------
CUST.# CUST.NAME       ON LT    3.00       3.50      14.3
       CITY,STATE
CUST.# CUST.NAME       ON MT    7.00       5.00      40.0
       CITY,STATE
CUST.# CUST.NAME       ON HT   20.00      20.50       2.5-
       CITY,STATE

4 Customers; Options |F1,F2,F3,F4,F5,F6,F8|
```



A SOLUTION FOR SALES MANAGEMENT'S NIGHTMARES



```
SIINQV01                 UNITED LIQUORS LTD.
                      Salesman/Item Sales Inquiry
   ----------------------------------------------------
     Vendor
     Owner
     Brand
     Group      ABSCIT                   ABSOLUTE CITRON
     Label
     Item #
     Size
     Pkg
     Class
     Vint
   ----------------------------------------------------

   7 Entries Qualified.  Enter More Information + Enter
```

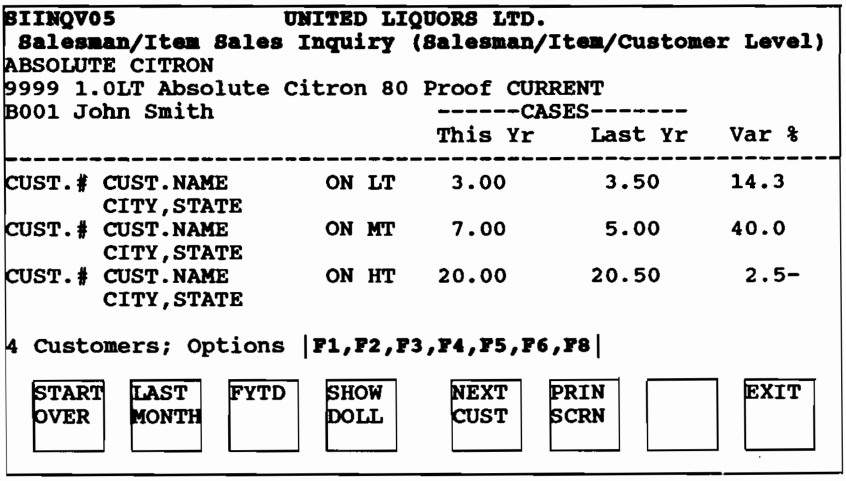


```
SIINQV01                UNITED LIQUORS LTD.
                   Salesman/Item Sales Inquiry
   ---------------------------------------------------
     Vendor
     Owner
     Brand
     Group     ABSCIT                 ABSOLUTE CITRON
     Label
     Item #
     Size      1.0LT                  1 LITERS
     Pkg
     Class
     Vint
   ---------------------------------------------------

   2 Entries Qualified.  Enter More Information + Enter
```

```
SIINQV02                    UNITED LIQUORS LTD.
     Salesman/Item Sales Inquiry (Sales Office Level)
ITEM NAME HERE
                                          CURRENT
                       This Yr  Last Yr            This Yr Last Yr
Sales Division          Cases    Cases    Var %    Dollars Dollars
------------------------------------------------------------------
 1   DIV01 1)UNITED    150.00   140.00     7.1     27,000  25,200
           2)BOSTON    125.00   110.00    13.6     22,500  19,800
           3)CENTRAL    25.00    30.00    16.7-     4,500   5,400

[   ]
Key # (To Show Salesmen Detail) + Enter
```

```
S11NQV03                    UNITED LIQUORS LTD.
       Salesman/Item Sales Inquiry (Salesman Level)
ABSOLUTE CITRON
                                          CURRENT
                       This Yr  Last Yr            This Yr Last Yr
Salesman                Cases    Cases    Var %    Dollars Dollars
------------------------------------------------------------------
 B001 John Smith        70.00    68.00     2.9     12,600  12,240
 B002 Dan Toupin        45.00    42.00     7.1      8,100   7,560
 B003 Jim Turner        35.00    30.00    16.7      6,300   5,400

[   ]
Key # (To Show Item Detail) + Enter
```

```
SIINQV04                    UNITED LIQUORS LTD.
     Salesman/Item Sales Inquiry (Salesmen/Item Level)
ABSOLUTE CITRON                           CURRENT
B001 John Smith        This Yr  Last Yr            This Yr Last Yr
                        Cases    Cases    Var %    Dollars Dollars
 1   9999 1.0LT         30.00    29.00     3.4      5,400   5,220
 2   8888 1.0LT         40.00    39.00     2.6      7,200   7,020

[   ]
Key # (To View Cust. Info.) + Enter
```



```
SIINQV05               UNITED LIQUORS LTD.
 Salesman/Item Sales Inquiry (Salesman/Item/Customer Level)
ABSOLUTE CITRON
9999 1.0LT Absolute Citron 80 Proof CURRENT
B001 John Smith                     ------CASES-------
                                    This Yr    Last Yr   Var %
-------------------------------------------------------------
CUST.# CUST.NAME          ON LT      3.00        3.50     14.3
       CITY,STATE
CUST.# CUST.NAME          ON MT      7.00        5.00     40.0
       CITY,STATE
CUST.# CUST.NAME          ON HT     20.00       20.50      2.5-
       CITY,STATE

4 Customers; Options |F1,F2,F3,F4,F5,F6,F8|
```

TITLE: An Introduction to Object-Oriented Programming and Databases

AUTHOR: Bruce Toback
OPT, Inc
10681 Foothill Blvd., Suite #201
Rancho Cucamonga, CA 91730
714-985-1581

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 3014

AND NOW FOR SOMETHING COMPLETELY DIFFERENT: INTERNATIONAL SYSTEM MANAGERS' PERSPECTIVES

KIM BARTON
MACKENZIE FINANCIAL CORPORATION
150 BLOOR STREET WEST
TORONTO, ONTARIO M5S 3G5
(416) 922-5322

WAYNE A. LAUGHTON
SABH PACKAGING GROUP
BRODIE STREET
RYDALMERE, NSW, P.O. BOX 6
RYDALMERE, 2116, NSW, AUSTRALIA
(02) 684-9100

INTRODUCTION

The writing of this paper was the direct result of meeting some very interesting people at last year's Interex Conference in San Francisco. During a late night session of swapping funny stories, the idea was conceived of a paper whose authors came from different parts of the world, giving it certainly an international quality.

The topics we will be discussing include:

1) Differences between MPEV and MPEXL based operating systems
2) Single and multiple machine management
3) Multisite management
4) International considerations and differences - are there really any?

Unfortunately, due to the distance factor, we were unable to complete our paper before the printing deadline; however, the completed paper will be available at the presentation in Boston.

DIFFERENCES BETWEEN MPEV AND MPEXL BASED OPERATING SYSTEMS

When our 925 arrived together with approximately 75 manuals, I did not quite know where to start. I began asking other MPEXL users for a brief summary of the operating system differences; however, I was unable to obtain a clear picture.

After moving our largest database over to the 925, I attended Hewlett Packard's course "Moving from MPEV to MPEXL: System Manager". Although I now had a much better understanding of the new operating system, my curiosity had not been completely satisfied. We also kept our series 52, so I wanted a good summary comparing the terminology of the two systems for existing staff and new staff coming on board. For this purpose, the following portion of the paper provides a comparison of the two operating systems: **MPEV** and **MPEXL**.

CONFIGURATION

There have been a number of changes made to the system configuration. Configurations are now completely menu driven and we no longer have to worry about those little scraps of paper reminding us which device name to use.

Another significant difference to **MPE**, is that the configuration has been separated into two parts: **Sysgen** and **NMMGR**. Sysgen allows us to configure system changes other than terminals and printers which are now handled by NNMGR. Both modules also feature an easy to use help facility.

SYSGEN VERSUS SYSDUMP

Sysdump is an interactive program in which you are prompted to answer questions. A cold load tape is then created to reflect these changes.

Sysgen is a command-based interactive program in which you have more control over the process of entering configuration changes or new data because it is command-based. An **SLT** tape, known as a system load tape, is then created to reflect these changes. However, it should be noted that an SLT tape does not include the accounting structure, a separate tape must be created. The same applies to backups done on the system.

SYSGEN VERSUS SYSDUMP

Below are two simple job streams which could be set up to run the SLT tape and a tape containing the directory:

```
!JOB SLTJ,MANAGER/PASSWORD.SYS/PASSWORD
!SYSGEN
TA
EXIT
!EOJ
```

```
!JOB DIRJ,MANAGER/PASSWORD.SYS/PASSWORD
!FILE T;DEV=TAPE
!STORE ;*T;DIRECTORY
!EOJ
```

SM capability is necessary to create, modify or save I/O configuration changes. **OP** capability will allow you to view the data only. Sysgen consists of a global module and four configurators. They are as follows:

IO configurator	-	configures local devices
Log configurator	-	changes user and system logging criteria
MISC configurator	-	controls system limits, startup values such as jobs, sessions and processes.
Sysfile configurator	-	changes the list of files dumped to a boot tape
	-	changes a copy of the CM segmented library which can be dumped instead of the original library.

One feature of this system is that all the configurators have a help facility. The only thing that was completely omitted from this menu-driven module was a quick method to print out everything.

I suggest setting up a jobstream to print all of the configurations at once; otherwise, you will have to go into each configurator and list each section off to the printer. Below is an example of a jobstream which could be used:

SYSGEN VERSUS SYSDUMP

```
!JOB CONFIG,MANAGER/PASSWORD.SYS/PASSWORD
!SYSGEN
PERMYES ON
IO
LDEV  DEST=OFFLINE
LPATH DEST=OFFLINE
LCLASS DEST=OFFLINE
LPATH DEST=OFFLINE
OCLOSE
EXIT
LOG
SHOW  DEST=OFFLINE
OCLOSE
EXIT
EXIT
!EOJ
```

NMMGR VERSUS SYSDUMP

In MPE, Sysdump assigns devices and their characteristics individually, while NMMGR separates the assigning of device numbers and their characteristics into two sections. Creating device characteristics is done by setting up profiles which can be applied to as many devices as you like.

Listed below are the statistics the profile needs. This is fairly similar to the Sysdump logical device setup.

Profile name
Terminal Type
Name of the custom terminal type file
Line speed
Record width
Is the hello command allowed
Is the terminal attached to a modem
Type of parity used
Does the port perform speed and parity sensing
Will the terminal reset after a lost connection
Device class name associated with the terminal.

The configuration information is contained in a file called **NMConfig.Pub.Sys**. When you are finished making any changes the file can then be validated. At this time you can send a listing to the printer. The output looks quite a bit more complicated than the simple format we have been used to on the MPE machines.

NNMGR VERSUS SYSDUMP

If you have made changes in both Sysgen and NMMGR there is a cross validation feature. This will check for any errors between the NMConfig.Pub.Sys and the system configuration file, and can be found in Sysgen under the I/O configurator.

As of version 1.2, MPEXL supplies only 3 terminal types and 4 printer types. MPE on the other hand has a choice of 15 terminal and printer types.

A brief description of the terminal and printer types supported on MPEXL is listed below.

Term types	Description
10	- HP terminals and PC's
18	- non HP devices - block mode not supported - data protection through **Xon / Xoff** - parity checking is not used
24	- pad terminals - does not support binary mode transfers typeahead blockmode applications parity generation and checking

Printer types	Description
18	- application printer - printer is controlled by a program running on the **XL**
21	- remote spooled printer - must be asynchronous - Xon / Xoff protocol
22	- spooled printer - can use the spooler but without a modem
26	- remote spooled printer connected through a network - connected through a modem multiplexer or pad

Also, it should be noted that the printer types on the MPEXL do printer status checks less frequently.

NNMGR VERSUS SYSDUMP

One other important note with regards to ports is that MPEXL only supports Xon / Xoff protocol. You may want to check any outside sources you communicate with if this will be a concern. A problem we have encountered is with a stock market source we load quotes from at the end of the day. Certain data in the price transfer file which acts as a control character to the HP, causes the port to hang.

DTC'S

DTC's are what have replaced **ATP's** and **ADCC's**. They are quite a bit different, so I think it is important to give you a little bit of an explanation. DTC stands for datacommunications and terminal controller, which is a local area network based communication. Each DTC has a processor card which handles the overall running of the DTC. The processor card, which is referred to as the **CPU** of the DTC, contains firmware and download software for most of the communication processing. An additional card to understand is the SIC card. The SIC card, also known as the serial interface card, controls the processing, storage, multiplexing of data, and serves the interface between the connector cards and the process card of the DTC. Connector cards are the ports on the system. Depending on the connector card, you can have six 25-pin ports or eight 3- or 5-pin ports on each SIC card.

DTC's go through their own self tests automatically when powered up. On the front of the unit there is a display window which always shows a code. If a self test and software download completes successfully, a sequence code of **F2** will appear in the display window. This code will remain on at all times as long as the DTC continues to function normally. A good book to read for a more in-depth understanding is Hewlett Packard's "Getting Started With The DTC - An Overview", Part #32022-90004.

A few disadvantages about DTC's are:

They are still new in comparison to ATP's and ADCC's.
We have had to install several patches to alleviate some of the hangs which we have encountered. An entire SIC card can hang, and powering off the DTC does not always work; therefore, you may have to do a start norecovery. There may be some problems with your site because they only support Xon/Xoff protocol.

SYSTEM STARTUP AND SHUTDOWN

System startup on the XL machines is quite a bit different from the classic machines. We are now faced with more than just a **halt>** prompt and a lot more waiting time. Do not expect to be able to take your system down and back up in 5 minutes. Some of the reasons for the increased waiting period are: i) there is more memory on the system, ii) recovery of lost disc space is automatically handled by the Transaction Manager, and iii) DTC's take a while to be downloaded.

Starting the system at the console is done through the access port (**AP**). A nice feature of the access port is that you can enable a remote console by issuing a command of **RE**. On MPE a switch had to be flipped on the back of the CPU. When enabling remote, you can specify the mode as either multiple or single remote access. If you select single access, a person can dial into the remote only once. This is another feature which is not available on MPE.

There are two methods available to booting up the XL - autoboot and interactive. The autoboot takes its commands from an autoexecute file also known as an autoboot file. The interactive boot prompts for the utility commands and options to start the operating system.

There are two kinds of resets - soft and hard.

Soft reset
- The command is **TC** (transfer of control) to issue a soft reset. This can also be done by hitting the reset switch and is applicable to the 925 and 935.
- The current CPU state is preserved.
- Retains all pending I/O operations before executing recovery software.
- Can also be done after an install update or dump.

Hard reset
- The command is **RS** (reset **SPU**) to issue a hard reset.
- Initiates the processor self test that resets memory and terminates all pending I/O operations. This is necessary after a power failure (if it is past the life span of the battery backup), hardware failure or due to an error from which the system is unable to recover.

A **CM>** prompt is displayed. You would enter your choice as to the type of startup desired and confirm your intention to boot up the system. It will take a few minutes before anything comes back to the console. If you enter a **Ctrl B** again, the system assumes you wanted to break out of your startup.

The next console message that comes up could be alarming to some people.

```
Autoboot from primary path enabled
To override press any key within 10 seconds.
```

Presently there is no autoexecute file on the system; therefore, nothing will happen if you forget to press a key. When one comes available you may still wish to avoid using it. In a crisis situation you would not want the system to automatically boot up if you had planned on booting from tape.

The next stage is a series of replies on how you are going to bring up the system:

```
Boot from primary boot path (Y or N)?
- boot from disc

Boot from alternate boot path (Y or N)?
- boot from tape

Interact with IPL (Initial program load) (Y or
N)?
```

After making your decision as to where you are booting from, the **ISL** is loaded in. It is at this step that you would issue the type of startup you would like to do.

A start recovery will use the old configuration file called NMConfig.Pub.Sys. A start norecovery will use a new configuration file call NMConfig.Pub.Sys.

MPEV does not have this facility. It uses the system configuration on the system regardless of its validity. You would have to return to a cold load tape to get a clean configuration.

At each of the various startup prompts you are able to issue a **Help** command for an explanation of the commands available.

Listed below is a comparison of what types of MPEXL starts there are, and the MPEV equivalent:

MPE XL	Start Recovery	Start Norecovery	Update NoConfig	Update Config	Install
MPE V	Warm Start	Cool Start	Update	Cold Load	Reload

UTILITIES AND OTHER FEATURES

There have been a number of new utilities and features added to MPEXL, as well as a few changed to existing MPEV utilities when they were moved over to the XL system. I am not going to go into a lot of detail because this section has been covered in previous Interact issues (October 1988, November 1988)

MPEV	MPEXL
Free5	DiscfreeA - histogram format DiscfreeB - allocation format
Listdirs	Listf Listuser Listgroup Listacct
Listlog5	Logtool - found in Sysdiag
Sadutil	Discutil
Termdsm	Termdsm - found in Sysdiag - can reset ports, SIC card or entire DTC
Dirtest or Flutil in Telesup	Fscheck - verification and correction of directory integrity
	Oct - translation of 3000 to compatibility mode
	Implied run
	Command line history stack - keeps track of the least 20 (default) commands entered in a session
	Listredo & DO - applies to the command stack
	Print command
	Chgroup
	Copy
UDCS	UDCS - can add, append and delete UDCS without having to reset the catalog

Sysdiag is a new feature on the MPEXL. It is an online diagnotics subsystem. Utilities such as **Termdsm** and **Logtool** have been moved into this subsystem. There are diagnostic programs as well as subsystem utilities.

UTILITIES AND OTHER FEATURES

Listed below are the programs and utilities which are available:

CS80DIAG	CS/80 disc diagnostic
DIAG7478	7974/7978 magnetic tape drive diagnostic
CIPERLPD	Ciper line printer diagnostic
PPDIAG	Paper printer diagnostic
HPIBDIAG	HP-IB device adapter diagnostic
MUXDIAG	Mux diagnostic
LANDAD	Lan diagnostic
LOGTOOL	System and memory log analysis tool
SYSMAP	System map
TERMDSM	Terminal diagnostic system monitor
CADIAG	HP-CIO channel adapter utility
IOTT	I/O test tool
INSTALL	Online diagnostic installer

For further information on these programs, there is a help facility in **Sysdiag**.

STORE/RESTORE

There have been some new features added to the store/restore utility, such as multiple exclusion on filesets. Up to 8 filesets can be entered to be excluded, whereas MPEV allowed only one fileset exclusion. One important note is that the directory is not automatically stored on tape; instead, you must use the **dir** option to have it included.

Another parameter available only to the restore utility is **Listdir**. This displays information from the tape label and directory. However, this can be used only on native mode tapes. **Forange** is a new parameter which allows you to store and restore files with certain filecodes. MPEXL distinguishes the different types of file codes. You may choose up to 8 file code ranges.

If you intend to create tapes on your XL machine for a MPEV machine, the transport option must be used. Keep in mind that if you are creating many tapes for MPEV, you may consider a faster backup product such as Backpack or Fastback, because the store process is slower with the transport option.

STORE/RESTORE

The sysdump tape on MPEV contained all the necessary information to reboot your operating system, as well as to restore the directory and user files; however, on MPEXL two sets of tapes are required. The SLT tape contains your configuration and system files necessary to run the system, as well as a store tape containing the user files and optionally including the system directory.

Vstore is a new feature to MPEXL. It is a command used for verifying data on the backup media and reports any errors incurred during the store process. On MPEV you could use validate in the **telesup** account.

SUMMARY

The MPEXL machines are really quite fascinating. I expect we will be seeing many new and interesting features in the years to come. If you are planning on moving to an XL machine, I suggest that you go through the system administrator's series once your 75+ manual arrives. If at all possible before moving on to the new system, try to get a chance to experiment with some of the new utilities and take advantange of the processing power.

Backup Performance
or
How to Backup Faster!

Bryan Carroll

Hewlett Packard
19111 Pruneridge Ave
Cupertino, Ca. 95014

One of your users has called to say they accidentally purged a file they must have to do their processing. Your operator has just informed you that a head crash has occurred on one of the disks on another machine. Then there is always the occasional hurricane, tornado or earthquake! All of these situations require that ability to recover the data that was lost from the disk. In order to recover the data however, a copy of the needed data must have been made prior to encountering the problem. These are the basic reasons behind every system backup. The backups we perform are valuable insurance against events we hope never happen. The first two problems mentioned above are not uncommon, but happily both of these and many other problems like these do have solutions.

Purging of files is an everyday occurrence. In fact most programs which manipulate data contained in disk files will often make temporary copies of the data in disk files which are used and then purged. It is a rare user who has not at one time or another purged a file that was still needed. I have heard it said that there are two kinds of airplane pilots, those who have landed without extending the landing gear and those who WILL land without extending the landing gear. The same can be said for users purging files they still need.

What really has to happen in order to accomplish a backup? A backup is simply a reserve or substitute according to Webster. For us a backup is the process of making a copy of something we may later have to substitute for the original. Any copy of a disk file, then can be a backup. This would include a hard copy printout of the data in the file. Although this would satisfy the definition of a backup, recovery would be a very difficult and lengthy process. We should modify our definition then to say a backup is the process of making a copy of a disk file such that recovery (or substitution) of the copy can be done quickly and easily.

The default backup procedure provided with most computer systems involves dedicating the computer to the task of the backup while everyone and everything else waits. For this backup procedure, it is very important to improve the performance of the backup software so as to minimize the time that other activities are waiting. The default backup procedure also usually involves copying the disk files to some form of tape media. Bottlenecks can occur in this configuration in three areas: the transfer of the data from the disk to main memory, the transfer of the data from main memory to the tape drive, and the CPU required to set up and execute these transfers. The slowest component of these three is usually the tape drive and it is often the bottleneck in the default backup environment.

The primary goal of most backup procedures, then is to minimize the time users are kept waiting for the backup to occur. A strong secondary consideration which helps determine how long users are kept waiting for the backup is the issue of cost. Given the primary goal of minimizing the time users are kept waiting for the backup, let's explore the different options based on cost.

MINIMIZE COST

The option to minimize all possible costs is usually the default backup procedure provided with most systems. Although this alternative is limited in many ways, there are still several alternatives to consider. This backup environment usually involves backup to a single tape drive with no special software or hardware to speed up the process. It also requires the most human labor of any of the options since someone must see to the chore of mounting, dismounting, and labeling tapes as they are needed.

Native Mode Backup

The first alternative that can be taken to improve the performance of your backup on MPE XL systems is to use native mode store. Stated differently, this means do NOT use the TRANSPORT option of the store command. This option will cause the resultant tape to be compatible with MPE V STORE tape formats, but at a significant cost. Unless compatibility is absolutely necessary for recovery reasons, I would strongly recommend that you use the native mode store (ie. NO TRANSPORT option).

Straight Backup

The most common backup process is to have everyone logoff and do a STORE or SYSDUMP of the entire system. I will refer to this as the 'Straight' backup. Various backup strategies can be used to alter the backup times such as full and partial backups. A full backup is an unconditional backup of every file on the system. A full backup can be performed at regular intervals such as weekly followed by regular partial backups. Partial backups are backups which only store the changes which have occurred since a particular event or date.

A common backup strategy is to perform a complete backup of the entire system (full backup) every weekend followed by daily partial backups which only store the changes since the full backup the previous weekend. Different needs and schedules at your shop may require different full and partial backup times, but you must consider the required recovery procedures when selecting a strategy. In the case of weekly full backups and daily partial backups, a full recovery would only require restoring the full backup followed by the most current partial backup. Longer cycles could require longer partial backups sometimes approaching the length of a full backup.

Application Backup

An alternative to the straight backup is the application backup. An application backup takes advantage of the unused resources (like the CPU) available during the store process. An application backup is a backup of only selected accounts or applications at one time. Assume a system had applications A, B and C normally running. An application backup would logoff users of applications A and B and let users of application C continue to process while the store of applications A and B occurs. When this store completes, application A and B users would be allowed to continue processing while application C users logged off so their files could be stored. Since the time required to store one or two applications would be smaller than storing the entire system, the user down time would be shorter and make better use of resources.

Backup Concurrent With Read Only Processes

Another variation on this idea is to allow read only activities during the store time. The first thing a store process does is to mark each file to be stored such that modifications cannot be made. After this, the files are written to the tape. Once the tape starts spinning (after the files have been marked to prevent modifications) read only jobs can be initiated to make use of the resources not required by the store process. In this case, read only users or jobs would only be waiting on the backup for a few minutes. It is important to ALLOCATE the programs to be used in this way BEFORE the store process begins.

MINIMIZE COST, LIMITED BUDGET

The first step beyond the most general backup environment involves the purchase of a faster or additional tape drive and/or backup software. Solutions in this category involved the purchase of one product (either software or hardware) to minimize costs. In many cases, additional products can be added to this first backup product to further enhance performance. These combinations will be discussed later.

Faster Tape Drive

The addition of a faster tape drive such as the HP7980XC or HP7980 drives will address the bottleneck at the tape drive. Backup performance will increase relative to the increase in the speed of the tape drive. In one case, replacing an HP7978B tape drive with an HP7980 improved the transfer rate to the tape about 37%. The gain in backup performance will vary depending on your file mix (fewer larger files can benefit more from the faster tape drive).

Additional Tape Drives

The addition of a tape drive will allow multiple concurrent application backups (as described above). The addition of one tape drive (for a total of two) can reduce the application backup almost in half. This would, of course, require some operations changes, but the benefit will usually be worth the effort.

Software

There are many software packages available to help you increase your backup performance. The features used by these packages include data compression, unattended backup and online backup of read only files. The software I am familiar with and have researched for this paper include TurboStore and the new TurboStore/XL II from HP, Mirrored Disk/XL also from HP, HIBACK/XL from HI-COMP America, Inc., Backup/3000 from Orbit Software USA Inc., and BACKPACK from Tymlabs. I have not purposely omitted any software and apologize if anyone was skipped. (If you have a backup solution for MPE XL systems not listed above, I would like to hear from you at 19111 Pruneridge Ave. 44MV, Cupertino, Ca. 95014)

Data Compression

Data compression is the process of combining like characters into a count and single copy of the character. An example would be a text file with lots of spaces at the end of each line. A compression algorithm would count the spaces and write on the tape the count along with one space. Then when the file was restored, another algorithm would expand the count and space into its original form. This is one example of many different compression algorithms. The major savings from data compression is that less data must be written to the tape which is the bottleneck. Another benefit is that you may be able to use less reels of tape and eliminate the significant rewind time for one or more reels, not to mention the reduction in tape costs.

The cost of data compression depends on the implementation of the compression algorithm and the algorithm itself. Most algorithms are implemented in software and therefore require some CPU time to execute. Therefore, most backup tools which offer a compression feature will require more CPU time to perform the same backup as the 'straight' backup. This is not a problem unless you are using a form of the application backup mentioned previously. If the algorithm is implemented in new hardware (such as the HP7980XC), the compression will not impact the CPU. This alternative also has some drawbacks however since you will require the same hardware to restore the data.

Unattended Backup

Unattended backup is a very attractive option for many shops and can have at least three different implementations. One alternative is to purchase as many tape drives as you will require reels of tape for a backup. Then you can start one backup process per tape drive storing one reel's worth of data per process. This solution can work for some sites whose disk space consumption is stable, but is probably not a good alternative for most shops. Another alternative would be to purchase a tape drive with an automatic reel (or cartridge) changer. This would allow a single backup process to write to a series of reels which are changed automatically.

Using a high capacity storage device is another way to achieve an unattended backup. Two new products available from HP combine high capacity with the ability to automatically change storage media to provide the capability of unattended backup for many sites.

Digital Data Storage (DDS) technology is now available in the Digital Audio Tape (DAT). DAT has the capacity to hold 1.3 gigabytes of uncompressed data and about 4.7 gigabytes of compressed data on each cassette. This compares very favorably with the 140 megabytes of capacity available on a 2400 foot reel of half inch tape written at 6250 BPI.

Rewritable Optical Disk technology will soon be available as the HP Series 6300 Model 20GB/A. This product can be used with TurboStore/XL II to store up to 73 gigabytes of data without operator intervention. This capacity is equivalent to the storage of over 100 HP7937 disk drives!

A more common unattended backup implementation offered by most backup products involves the use of disk as an intermediate storage location. The backup software can perform an unattended backup from disk to disk. The copy could then optionally be written to tape at a later time. A normal backup procedure with this strategy might be to perform a disk to disk backup during the night while no other processing is scheduled. Then when the operator comes in the next day, a tape copy could be made to store off site. The major drawback to this procedure is the disk space involved. Although packages which offer this feature usually combine it with the data compression feature, a significant amount of disk space must be dedicated to this process.

Uninterrupted Backup

Uninterrupted backup is a form of online backup which we will discuss in more detail later. Most packages offer a version of online backup with the significant restriction that any changes made to the files while the backup is in progress may or may not be reflected in the backup. What is really being offered by this feature is the ability to allow read only functions to co-exist with the backup process. A backup of this nature simply skips the step of marking the file as being stored (store bit in the file label is NOT set). The net result is the same as allocating the program before the store begins as mentioned before.

MINIMIZE USER DOWN TIME

The final category of backup alternatives is the most expensive, but provides the lowest (sometimes eliminates) user down time. The solutions involve the purchase of both software and hardware to achieve this purpose. Most of the software packages available can provide the benefits mentioned in the previous section and can expand to provide additional benefits as additional hardware is added. For example, most products provide a data compression feature which will help backup performance with no additional purchases. The same product, however, can often make use of multiple tape drives if a second or third tape drive is later purchased.

Multiple Tape Drives

Some backup products can make use of multiple tape drives to speed up backups. There are two basic ways this is done. The first is to write to one tape drive until reaching the end of the reel, then switching to a second tape drive. This procedure allows you to eliminate the rewind time since you will be writing to the next tape while the first tape is rewinding. This keeps all the data in the order you would expect (usually alphabetical) on the reels, but only can make use of two tape drives.

The second procedure involving multiple tape drives is to write to multiple tape drives at the same time. This addresses the tape drive bottleneck which is usually present during backup time. If enough tape drives are available so that the number of reels required by the backup is matched by the number of tape drives, this feature also becomes an unattended backup as well. This option will perform much better than the first multiple tape drive option, but the files may not be in the order you expect on the tapes. This may only be a minor inconvenience, but it could mean that recovery of a single file may be more difficult.

Disk Backup as Permanent Backup Media

Another alternative is to dedicate one or more disks to the backup function. A private volume set can be created from removable media disks and backups performed to this private volume set. The media can then be removed from the private volumes and archived and new media can be used for the next backup.

This option can get quite expensive since it requires several disks drives and many more disk packs to be dedicated to the backup function. A system configuration involving multiple private volume sets can make this a better alternative. One private volume set can be removed and the backup disk packs mounted in these drives to backup the other private volume sets. Then a swap could be made to backup the volumes first removed. This solution would have many details to work out, but may be possible for some shops.

TurboStore/XL II with Online Backup

TurboStore/XL II is a new backup product for MPE XL systems. This new product can be ordered with online backup capability providing 24 hour system availability. A new MPE XL subsystem called Shadowing has been created to provide the file management necessary for this functionality.

When the Store operation is started with the ;ONLINE keyword, Shadow logging will be established for each file in the store file set. This process will maintain one logically consistent copy of each file to be used with the backup operation. Any modifications to these files will be kept in another area until the backup is complete. At this time, the modifications made to the file since the backup began will be incorporated into the file.

This capability provides 24 hour a day up time for users while protecting the data by backing it up. Most other backup functions can be combined with the online backup feature including compression and the use of multiple tape drives.

Mirrored Disks - Online Backup

A new product called Mirrored Disk involves the use of disk pairs. Mirrored Disk is a facility to protect against disk failures and was not created specifically to address backup issues. Although usually purchased to protect against disk failures, Mirrored Disk can provide excellent performance alternatives when it's backup time.

There is a master and slave designated in each pair and both are updated with every disk modification. Each disk is then kept up to date as the changes are made (real time). When a backup is performed, the slave disk is split apart from its partner and mirroring is suspended. Updates are stored in another location until the slave is reunited with its partner. Once the slave is split, it can be backed up while users continue to access the master. Except for a few minutes of user down time to get the backup started, read and modify users will not be impacted by the backup. Once the backup is complete, the disk pair is reunited and the slave is updated with all the changes that took place while the backup was in progress.

This new feature provides another good alternative in minimizing user down time even though protection from disk failure is its main goal. Mirrored Disk works with user volumes at first release and it requires twice the number of disk drives actually required by your application. Depending on your application disk drive requirements, this could get expensive, but it does provide almost no user down time for backup.

EXAMPLE PERFORMANCE IMPROVEMENTS

The following examples are taken from case histories from several sites along with a few specific tests created specifically for this paper. The results shown here are intended to provide a single reference point to help the reader gain a feel for the general performance gains possible. Your experience is likely to be different since there are many variables involved in determining backup performance and it is likely that one or more of these variables are different between your site and these cases. Please use these examples carefully.

Minimize Cost

In one case a particular site was encouraged to use the native mode store rather than the compatibility mode (or TRANSPORT mode) store. Their normal backup using the TRANSPORT option was taking about 2.4 hours (or 2 hours, 24 minutes). After being encouraged to use the native mode store, their backup was completing in about an hour.

In another case where the TRANSPORT option was being used, backup time for a particular account (application backup) was taking about 18 minutes elapsed time and 230 CPU seconds. This same backup done with native mode store took only 11 minutes elapsed time and 95 CPU seconds. This represents an improvement of almost 40% elapsed time and almost 60% CPU time.

Minimize Cost, Limited Budget

I took the opportunity to run some tests of my own involving data compression. I took a file set which is often stored on our system which consists of 1,000,000+ sectors of disk space spread across 897 files. The average file size is 1173 sectors (arithmetic mean). This file set includes all different types of files including Turbo Image Data Bases, SD files, FIGure files, text files, etc. In my test, I stored these files in three different environments, TurboStore to an HP7980XC (no data compression), TurboStore to an HP7980XC with the compression on, and a Store using a popular third party product which does compression in software. This set of tests will provide us a comparison between the two different types of compression, hardware compression and software compression. Here are the results:

	Connect Seconds	% Relative to NO Compression	CPU Seconds	% Relative to NO Compression
TurboStore I HP7980XC No Compression	696*	-	128	-
TurboStore I HP7980XC Compression ON	489	70%	128	100%
Third Party HP7980XC Software Compression	560	80%	395	308%

* The test without compression required two reels of tape. The rewind and reel change time of about 2.5 minutes has been included in this measurement. The other tests only required one reel to complete the store.

What I learned from this test is that adding the compression feature in hardware will not add any additional CPU overhead and will reduce the elapsed time to about 70% of a store without compression. The major drawback is that the compression hardware (ie 7980XC) will be required to restore the tape created this way.

When doing the data compression in software, the elapsed time is 80% of the uncompressed store, but it costs a significant amount of CPU. Doing the compression in software has the same drawback as the hardware in that the software must be present to restore from a tape created. This limitation is usually must less of an issue, however, since it is usually less expensive to duplication software (right to copy license) than hardware. The bottom line demonstrated by this test is that both forms of data compression are faster than a store without data compression.

Another important issue not measured in this test is the media reduction. The store without any compression required almost two full 2400' reels of tape. Both of the compressed stores required less than one full reel. Another site I am familiar with frequently stores the same type of file set and has seen a reduction of seven tapes into a single tape using the HP7980XC hardware compression. This can be a significant advantage in tape costs, ease of transportation (have you ever had to carry seven tapes on an airplane?) and allows uninterrupted stores in many cases.

Further media reductions can now be realized with the new Digital Data Storage (DDS) and Rewritable Optical Disk storage technologies. These technologies which will soon be available in HP Products, can reduce several reels of half inch tape into a size that fits in your pocket! The size and flexibility of this storage media will allow us to view storage media in a new light.

Minimize User Down Time

A large system with lots of disk space can greatly benefit from using multiple tape drives concurrently. In another case I am familiar with the configuration included 16 HP7937 disk drives connected to a Series 955 with HPFL cards. Their normal TurboStore Full backup was taking a little over three hours to a single HP7980 tape drive. The site was able to add a second, then a third and finally a fourth HP7980 tape drive. The reduction in backup times is listed below.

	1 HP7980	2 HP7980's	3 HP7980's	4 HP7980's
Backup Time	3:00	1:42	1:14	1:00
Performance Relative to 1 HP7980	-	1.76	2.43	3.00

It is easy to see that a substantial reduction in backup time can be made over an already optimized backup environment by adding additional tape drives.

SUMMARY

We have discussed three major categories of backup solutions, Minimize Cost, Minimize Cost with a Limited Budget and Minimize User Down Time. There are many solutions in each of these areas.

Without any special hardware or software, using native mode store and doing application backups can help minimize user down time. These solutions are 'free' (no additional cost) and only require operational changes.

Several solutions are also available if you have the budget to buy one hardware or software product. These include upgrading to a faster tape drive, purchasing an additional tape drive, purchasing a high capacity device, or purchasing one of several software products. Improvements can be observed with the software products from features like data compression, unattended backup and uninterrupted backup.

If your users are complaining about down time for backups and you must satisfy them even if it means spending more money, there are several alternatives which will reduce the down time for backup even more. Purchasing a software product in conjunction with additional hardware will usually address these concerns.

There are many backup solutions to choose from. You are no longer limited to just one or two alternatives on MPE XL machines. Solutions now exist from the straight backup with long user down times to TurboStore/XL II with Online Backup using a high capacity backup device to provide minimum user down time. Match your needs with one of these solutions to satisfy your users and minimize your down time.

Basic Resource Accounting

Bruce J. Senn
Senior System Manager - HP3000
Union College
Schenectady, NY

This paper begins by describing the measures used in two resource accounting systems. Following these descriptions is a brief discussion of the importance of involving management in the development of a resource accounting system. Finally, the paper describes the tools used to collect, reduce, and report data for each measure. These tools are available at most HP3000 sites.

The paper begins by describing measures because it is believed that deciding what to measure is the most important part of setting up a resource accounting system. Carefully thinking about and agreeing on what the system should measure ensures that it is meaningful for a particular business situation.

Describe what you want to measure.

Eight items will be described. These items are taken from the system currently in use at Union College. A subset of these items was used in an earlier effort at E. G. & G. Rotron. Each item measures the consumption of a particular system resource or gives an indication of how much work the system is doing. Each item is derived from a question about system resources. Where available, account level detail is used to understand, categorize, and manage the various types of applications which run under each account.

1. Uptime. What percentage of the time is the computer available?

Two categories of uptime are distinguished, hardware uptime and system uptime. The HP3000 at Union College serves a wide variety of applications including fund raising, admissions, payroll, and financials. The system is used for online data entry, online inquiry, batch reporting, and batch updating. Batch work often runs all night and on weekends, hence, at Union total time is 7 days * 24 hours.

The hardware available percentage is the proportion of total time the hardware and operating system are available. The time for system failures, hangs, halts, hardware reconfiguration, preventive maintenance, and recovery is subtracted from total time and the percentage of total time calculated.

The system available percentage is the proportion of total time the system is available to service the work of end users. The additional time needed by the system to maintain integrity and keep itself in working order is also subtracted from total time and the corresponding percentage calculated. At Union, time for backups, for condenses and reloads, and for global database and file maintenance is subtracted.

The application available percentage is the proportion of total time a particular application is available to its users. Such things as the time to expand or repack datasets and the time to remove deleted entries from KSAM files are also subtracted from total time and the corresponding percentage calculated.

2. Disc Consumption. How much disc storage is used by the system in total, and, as detail backup, how much is used by each account?

An absolute number expressed in sectors or megabytes is used rather than a percentage. Only permanent files are counted because permanent files consume most of the disc space on the systems at Union. A case can also be made for including temporary files, spoolfiles, virtual memory, and disc based MPE tables if they consume significant amounts of disc space.

3. Disc Percentage Used. What percentage of total disc space is not free space?

Stating that disc space is 75 % consumed conveys a truer message than 25 % free space. Hence, free sectors are subtracted from the total sectors on all drives and the corresponding percentage calculated.

4. CPU Consumption. How much time does the CPU spend working? How much time does it spend working for each account?

Again an absolute number, expressed as CPU hours or CPU seconds, is used to measure the work of the CPU. The units are less important than the trend over time and the detail by account.

5. Connect Time. How much time do users spend connected to the system?

As with CPU time, an absolute number, connect hours or connect seconds, is used to track the trend in total connect time and detailed usage by each account.

6. Disc Activity. How active are the disc drives in total and by account?

"Disc Records Accessed" is reported by the LOGREPT program. It is used as a measure of disc IO, to identify IO intensive applications, and to show the trend of disc IO in total.

7. Jobs and Sessions. How many jobs and sessions are created in total and by account?

The number of jobs is a measure of batch work and the number of sessions is a measure of interactive work. While not currently used, CPU consumed by jobs and sessions respectively, are also measures of batch and interactive work.

8. Printed Pages. How much printed output is generated by users?

Printed pages measures the quantity of printed output.

Check your list of items with management.

What has become clear to me during the description process is how site-specific most of these items can be. How each item is defined depends on what resource accounting is to achieve and what the system is called on to do. For example, in an earlier project, total time was defined as 6 days * 10 hours because the system was used only during normal working hours for an online order entry system. After generating simple reports and mainframe transactions the system had nothing to do. Also in that earlier project, account level detail did not fairly allocate disc consumption, therefore, a more involved allocation algorithm had to be developed.

I believe that it is this site and mission specificity which creates the importance of checking resource accounting definitions with management, both within the MIS organization and at a more general business level. Secondly, I believe that in order to manage the system at a technical level, the system manager needs periodic reports about how available the system is, how much work it does, and what resources are consumed. The resource accounting done at any specific site needs to meld these management and technical needs and varying the level of detail can often accomplish both management and technical purposes.

For example, a system manager needs to know about disc consumption by account and group so that action can be taken to purge files. The system manager also needs to know which applications consume CPU so that the workload can be balanced. At a higher level, MIS management needs to know about trends in disc consumption, CPU load, and system availablility in order to anticipate and budget capital expenditures. At the highest level, the General Manager may only need to know how available the system is. Finally, at all levels a customer service orientation requires sufficient knowledge to identify major customers, and, with judicious system management, to serve them better.

Because a computer is an asset which serves the goals of an organization, I submit that management should be involved in determining how to monitor the productivity of that asset. I also submit that management should think about this as soon as the system is installed. In this context, I think a computer is similar to a general purpose NC machining center: management wants to know how many pieces of what part it makes, how long it is down for repairs, and which product lines it serves.

While I submit that management involvement is important, I must also say from experience that many managers fail to see its importance until there is a crisis. My exhortation to the technical system manager is: think about monitoring your system early, describe what you want to monitor, check out your monitoring plan with management, and finally, report your results regularly to the appropriate levels of your organization.

Two Systems of Resource Accounting.

In the past I have created two schemes of resource accounting, each several years apart. The first system, at E. G. & G. Rotron, was very basic. It used the REPORT command, console logging, the LISTLOG utility, hand written logs, and VISICALC spreadsheets. This first system is described in more detail as System A. System A was a "spare time" project, but driven by the need to monitor a system which was very close to capacity. System A had to report what departments used the machine for what applications and detail by both account and application were essential.

The second system is more sophisticated and is currently in use at Union College. It uses the LOGUTIL programs in the TELESUP account (LOGSNAP, LOGAUDIT, and LOGREPT), TDP, DSCOPY file transfer, and 2020 spreadsheets. This second system is described in more detail as System B. System B was a project driven by MIS management from a desire to report resource consumption by account to the user community on an annual basis. At the same time, as a new system manager, I needed to analyze this resource consumption in more detail on a monthly basis in order to better understand the systems and manage their workload.

System A.

1. Uptime.

In System A a paper log of downtime events was kept. These events happened infrequently and the time to write them down was minimal. Console messages were sent to the system log files and each month HP's LISTLOG was used to print the console log. TDP was then used to scan a disc copy of the printout for LIMIT entries and from these entries times of restricted access were noted. The paper log became the source document for calculating the percentage of uptime in each category.

2. Disc Utilization, CPU Time, and Connect Time.

Each month a job was run which issued the REPORT command to list, by account and group, disc space consumed by permanent files, CPU time, and connect time. The job then zeroed out counters for the latter two items. From the printed report the account and group totals had to be entered into a VISICALC worksheet because, at that time, no import function existed. The spreadsheet allocated this raw data to both applications and cost centers including an apportionment of a single common database. The spreadsheet also calculated dollar values by applying the percentage of each resource consumed to a monthly cost derived from a combination of historical and replacement cost.

The presentation of summary data was made with graphs and portions of the spreadsheet. The graphs were drawn by hand by adding points to a 12 month master and then xeroxing the master to produce the current YTD graph. The final piece of the monthly report was a management cover letter, which explained trends and summarized events of the month and their impact on computer resources.

System A was relatively simple. It was not elegant, but it accomplished the job of resource accounting with the tools that were available. Appendix A shows the summary spreadsheet and an example of the hand drawn graphs.

System B.

The jobs and macros which accomplish much of the work of System B are shown in Appendix B. Note that the macros need to be changed to reflect the current month. Note also that for the HP3000 jobs, date parameters are normally replaced by a job scheduler. For clarity this replacement has been done for February 1990.

1. Uptime.

In System B uptime is still tracked with a 'manual' log, although it is now kept as a 2020 spreadsheet. Given the reliability of the hardware and of MPE, downtime events are still infrequent. A macro is used to enter data about daily backups (Macro #1). These and other entries are made by scanning the console log which is sent to a printing terminal.

2. Disc Space Used.

Disc consumption is taken from a monthly run of the REPORT command (Job JAYYMMJ2). However, the report is edited with TDP to delete the group subtotals and to format it for input to a 2020 spreadsheet. Because 2020 runs on one of the College's VAX 11/780s the edited file must be transferred with DSCOPY (Job JAYYMMJ3). Once the transfer file is on the VAX, a macro creates a monthly spreadsheet, sorts each category by account, calculates totals, and prints a report (Macro # 3).

3. Disc Percentage Used.

Each day a report is generated based on FREE5, from HP, and LOSTDISC, from the CSL. The FREE5 output is read by a local program which knows the capacity of each disc drive and which reports the largest free area and the percentage of free space in a concise format. The percentage of free space on the last day of the month is used as the monthly figure. The systems at Union each have a 'reserved disc' so the percentage of free space is calculated both with and without this drive.

4. Other Items.

CPU Time, connect time, disc activity, jobs and sessions, and printed pages, are all taken from a monthly run of LOGAUDIT and LOGREPT (Job JAYYMMJ1). These 'unsupported' programs are supplied by HP as part of the TELESUP account. LOGSNAP stores the system log files to a tape. LOGAUDIT reads this tape and creates an intermediate file. LOGREPT then uses the intermediate file to print a report. The report is sent to disc and edited with TDP for input to 2020. Again DSCOPY moves the file to the VAX and another macro loads, sorts, totals, and prints a monthly spreadsheet (Macro # 2).

It is important to note that the 'LOGUTIL' programs are unsupported. There is little documentation and no source code. While they are available at no cost, they can fail at any time. Indeed, a recent upgrade to V-delta-4 caused the count of printed pages to become zero.

System B collects more data and uses a more powerful spreadsheet. 2020 calculates least square trends and draws reasonable looking graphs on a

VT241 graphics terminal (Macro # 4). A screen print to a ThinkJet suffices for internal purposes. These graphs are shown on the last two pages of Appendix B.

Each month the graphs and spreadsheets are printed and reviewed briefly for noticeable changes. Each year the monthly spreadsheets have been consolidated for an annual report. Monthly spreadsheets are the source of basic information which can then be used in a variety of ways. If there is a need to monitor backup time there is data available. If there is a need to report CPU percentage by College Office there is data available as well. The spreadsheet approach has proven useful because it has the flexibility to produce reports which have not been anticipated.

Conclusions.

This paper has shown that a reasonably coherent system of resource accounting can be put together from basic tools which are usually available to HP3000 system managers. A purchased package is not necessary if one uses HP provided commands and utilities, programs from TELESUP or from the CSL, and a spreadsheet package. Nevertheless, these 'make do' systems take some time to create and they don't automatically produce monthly reports. In fairness to third party vendors I must say that resource accounting packages are relatively inexpensive and quite comprehensive. However, if purchasing software is not possible, a resourceful system manager can do pretty well with tools that already exist.

Looking to the future, the next time I am charged with a resource accounting project, the focus will be on discussions with management about what should be measured and what should be reported. The marketplace will then be searched for a package which meets those requirements and the cost compared to the time and effort to put together another 'make do' system. I will also consider using a PC database like DBase or a relational database like ORACLE instead of a spreadsheet.

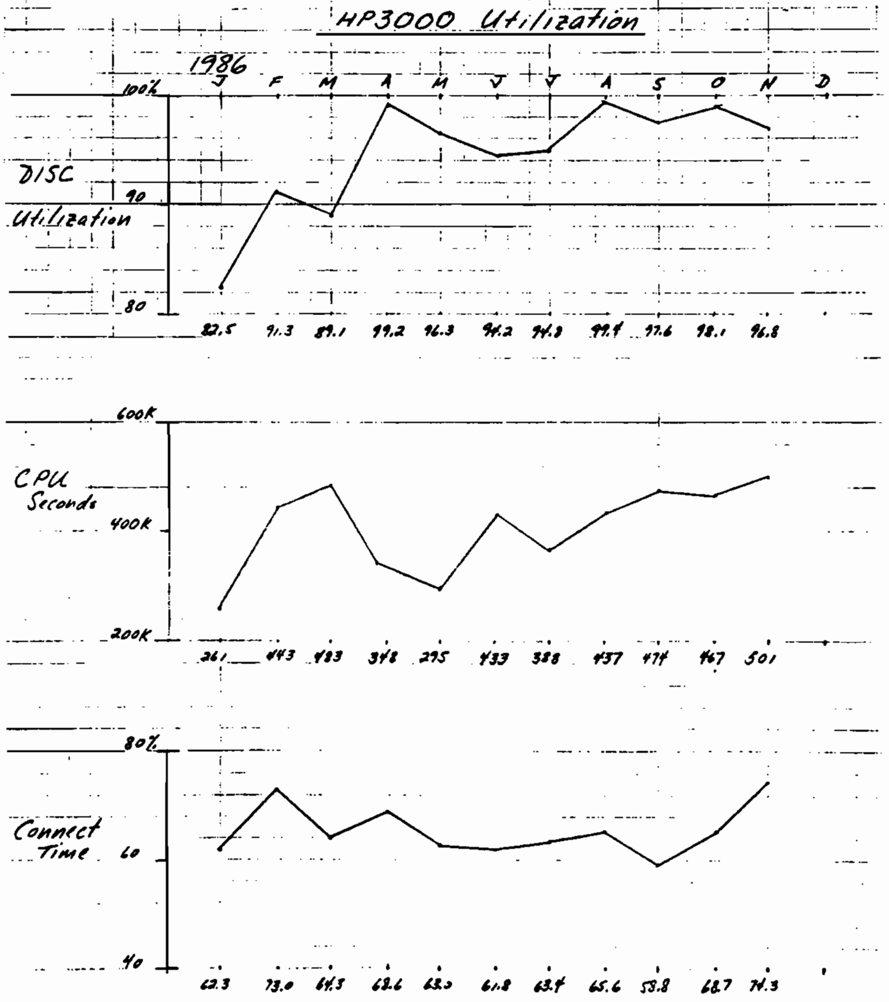
System A grew from a situation where a departmental system had become overloaded by work from others parts of the company. System B grew from a need to show resource consumption to the user community. System B also helped understand the system and its workload. Both systems have aided me as a system manager and have let my superiors manage their computer resources more effectively.

Appendix A.

```
                                              WED, NOV 26, 1986,   3:41 PM  PAGE: 1

                                   HP3000 ESTIMATED COST -- NOVEMBER    1986
                                   ----------------------------------------
EST. COST/MO -- DOLLARS                 1000      3250      1750      6000

CC-APPLICATION                          DISC       CPU   CONNECT     TOTAL
--------------                      -------- -------- --------    ------
ACCOUNTING                   700
    ORDER ENTRY
    OA
    ==========)                           95       163        76       333      5.6%
MARKETING                    703
    ORDER ENTRY
    QUOTES
    OA-SFC
    OA
    ==========)                          274      1072       947      2293     38.2%
SALES                        704
    ORDER ENTRY
    OA
    ===========)                          72       128        61       262      4.4%
MATERIAL CONTROL             715
    ORDER ENTRY
    SHOP FLOOR
    ===========)                         109       259        94       462      7.7%
QUALITY CONTROL              720
    ORDER ENTRY
    OA
    ===========)                          87        63       159       309      5.1%
MED                          726
    OA MCAL
    OA MED
    ===========)                          22       112        29       163      2.7%
AMED                         727
    OA
    ===========)                           1         8         3        11       .2%
MIS SUPPORT                  718
    ORDER ENTRY
    OA
    PROGRAMMING
    HP GENERAL
    HP TUNING
    ===========)                         340      1446       381      2166     36.1%
TOTAL                ====)              1000      3250      1750      6000    100.0%

APPLICATION-CC                          DISC       CPU   CONNECT     TOTAL
--------------                      -------- -------- -------- --------
ORDER ENTRY
    ACCOUNTING               700
    MARKETING                703
    SALES                    704
    SHIPPING                 715
    Q/C                      720
    SUPPORT                  718
    ===========)                         513      1510       958      2981     49.7%
QUOTES
    MARKETING                703
    ===========)                          98       222       112       432      7.2%
SHOP FLOOR
```

```
                                          WED, NOV 26, 1986,  3:41 PM  PAGE: 2
     MAT CTL           715
     ===========>                7         2         2        11        .2%
OA
     ACCOUNTING        700
     MARKETING         703
     SPR               703
     SALES             704
     Q/C               720
     MCAL              726
     MED               726
     AMED              727
     SUPPORT           718
     ===========>              179       394       394       967      16.1%
PROGRAMMING
     MIS               718
     ===========>               48        52        56       156       2.6%
HP3000 SUPPORT
     MIS               718
     ===========>              149      1066       220      1434      23.9%
HP3000 TUNING
     MIS               718
     ===========>                7         4         9        19        .3%
TOTAL           ====>         1000      3250      1750      6000     100.0%
                                                            6000
```

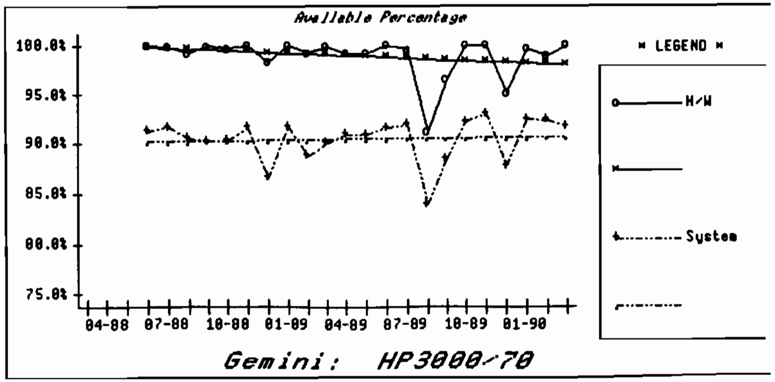
Appendix A.


HP3000 Utilization
1986
J F M A M J J A S O N D
100%
90
80
DISC Utilization
82.5 91.3 89.1 99.2 96.3 94.2 94.8 99.4 97.6 98.1 96.8
600K
400K
200K
CPU Seconds
261 443 483 348 295 433 388 437 474 467 501
80%
60
40
Connect Time
62.3 73.0 64.3 68.6 63.0 61.8 63.4 65.6 58.8 68.7 74.3

<u>Appendix B</u>.

Uptime Worksheet and Graph.

GEM		Console Log: 90-02						13,669	3	
File:		GEM_CONSOLE_9002						13,668,564		1,580
		GEMINI								
Date		Category	Start	Stop	Down	System	Applic	Free Sect	% Free	% wo Res
		FULLBACK	1911	2206		175				
		Wait	2206	2220		14				
		CONDENSE	2220	2452		152				
2.24	SA							4,012	29.4%	20.1%
2.25	SU	PARTBACK	1901	2011		70		4,058	29.7%	20.5%
2.26	MO	PARTBACK	1900	2000		60		4,051	29.6%	20.4%
		MAILMAIN	2000	2057			57			
2.27	TU	PARTBACK	1858	1956		58		4,030	29.5%	20.3%
2.28	WE	PARTBACK	1900	2009		69		4,004	29.3%	20.1%
02-90		SUMMARY			428	2551	252	4,296	31.4%	22.5%
27		DAYS	38880		98.90%	92.34%		4,004	29.3%	20.1%
SAVED		FORMULAS			0	0	0	4,295	31.4%	22.5%



Appendix B.

2020 Macro # 1: Enter Daily Backup Information

```
#right
#input(prompt "Enter Start Time")
#right#;#right #right #right #right #right #right #right
#input(prompt "Enter No. of Files")
#right#;#right #right
#input(prompt "Enter No. of Uncompressed Sectors")
#left#;
#input(prompt "Enter No. of Tape Blocks")
#left#;
#input(prompt "Enter No. of Reels")
#right#;#right #right
#input(prompt "Enter No. of Compressed Sectors")
#right#;#calc
#left #left #left #left #left #left #left #left #left #left #left #left
#input(prompt "Enter Stop Time")
#left#;#left #calc
```

2020 Macro #2: Create Monthly Spreadsheet from LOGAUDIT Data

```
/SRXXX_YYMMLOG#/'DATE:~~02-90~~SYSTEM:~~GEMINI~~~FILE:~~GEM_9002LOG#/
>A7#/
/SIDFGEM_9002LOG.TNS#/
/MDR8...9#/
/MMR7#/2#/
/MDR3#/#end
/MDR#/#home
/TDSC[(A,F),7...80]#/F#/DJ7#/
/TDSC[(A,E),7...80]#/E#/DM7#/
/TDSC[(A,D),7...80]#/D#/DP7#/
/TDSC[(A,G),7...80]#/G#/DS7#/
/TDSC[(A,B),7...80]#/B#/DV7#/
/TDSC[(A,C),7...80]#/C#/DY7#/#calc
/SWGEM_9002LOG#/
```

2020 Macro # 3: Create Monthly Spreadsheet from REPORT Data

```
/SRXXX_YYMMREP#/'DATE:~~02-90~~SYSTEM:~~GEMINI~~~FILE:~~GEM_9002REP#/
>A4#/
/SIDFGEM_9002REP.TNS#/
/MBA4...D4#/
/TDSC[(A,B),5...80]#/B#/DF5#/
/TDSC[(A,C),5...80]#/C#/DI5#/
/TDSC[(A,D),5...80]#/D#/DL5#/#calc
/PPGEN_TTH3#/#/
/SWGEM_9002REP#/
```

2020 Macro # 4: Print Graphs to Terminal Screen

```
#NORMAL
/SRGEM_GRAPHS#/
/GPDTERMINAL#/Q
NG##1#list#;#/PGY#WAIT(30)#/
NG##2#list#;#/PGY#WAIT(30)#/
NG##3#list#;#/PGY#WAIT(30)#/
NG##6#list#;#/PGY#WAIT(30)#/
NG##4#list#;#/PGY#WAIT(30)#/
NG##5#list#;#/PGY#WAIT(30)#/
NG##7#list#;#/PGY#WAIT(30)#/
Q
```

Job JAYYMMJ2: Monthly Job to Create REPORT Output

```
!JOB JAYYMMJ2,MANAGER.SYS,JA;HIPRI;OUTCLASS=LP,8,1
!TELLOP; *-----------------------------------------------------------*
!TELLOP; *              JA9002J2 IS STARTING.
!TELLOP; *-----------------------------------------------------------*
!COMMENT *------------------------------------------------------------*
!COMMENT *  Job to store the results of the report command into a    *
!COMMENT *  binary file and an ascii file.  These files are for      *
!COMMENT *  future use by a resource accounting system.              *
!COMMENT *                                                           *
!COMMENT *  The results of the report command are printed and the    *
!COMMENT *  account totals are set to zero.  The job is intended     *
!COMMENT *  to run monthly so that monthly resource consumption      *
!COMMENT *  reports can be produced.                                 *
!COMMENT *------------------------------------------------------------*
!FILE REP9002B;DEV=DISC;REC=,,,BINARY;NOCCTL
!REPORT @.@,*REP9002B
!SAVE REP9002B
!COMMENT *------------------------------------------------------------*
!FILE REP9002A;DEV=DISC;REC=-80,,F,ASCII;NOCCTL
!REPORT @.@,*REP9002A
!SAVE REP9002A
!COMMENT *------------------------------------------------------------*
!FILE REP9002L;DEV=LP
!REPORT @.@,*REP9002L
!COMMENT *------------------------------------------------------------*
!RESETACCT @
!COMMENT *------------------------------------------------------------*
!COMMENT *  COLLECT FREE SPACE DATA TO INCLUDE WITH REPORTS.         *
!COMMENT *------------------------------------------------------------*
!BUILD UT360S01;REC=67,,F,ASCII;DISC=150;TEMP
!BUILD UT361S01;REC=67,,F,ASCII;DISC=20;TEMP
!FILE FREE5OUT=UT360S01,OLDTEMP
!FILE STDLIST=UT361S01,OLDTEMP
!RUN SHOWOUT/xxx.PROG.CONTROL;STDLIST=*STDLIST
!RUN FREE5.PUB.SYS
!FILE FTN08=$STDLIST;ENV=ELITE.HPENV.SYS;CCTL
!RUN DISCUTIL/MLJ.PROG.CONTROL
Y
!RESET FTN08
!RESET STDLIST
!RESET FREE5OUT
!PURGE UT360S01,TEMP
!PURGE UT361S01,TEMP
!COMMENT *------------------------------------------------------------*
!COMMENT *  USE THE UTILITY LOSTDISC FOR A DIFFERENT FORMAT.         *
!COMMENT *------------------------------------------------------------*
!FILE LOSTDISK=$STDLIST;ENV=ELITE.HPENV.SYS
!RUN LOSTDISC.PUB.TECH;STDLIST=*LOSTDISK
!TELLOP; *-----------------------------------------------------------*
!TELLOP; *              JA9002J2 HAS COMPLETED
!TELLOP; *-----------------------------------------------------------*
!EOJ
```

Appendix B.

Job JAYYMMJ1: Monthly Job to Process System Log Files

```
!JOB JAYYMMJ1,MANAGER.SYS,JA;INPRI=13;OUTCLASS=LP,8
!TELLOP; *-----------------------------------------------------------*
!TELLOP; *  JAYYMMJ1 for 02-90 has started.
!TELLOP; *-----------------------------------------------------------*
!comment *-----------------------------------------------------------*
!comment *  Job to create job accounting report files from LOGSNAP   *
!comment *  tapes.  Files are then to be transferred to a PC for     *
!comment *  use with Lotus 1-2-3.                                     *
!comment *-----------------------------------------------------------*
!comment *  PART 1:                                                   *
!comment *                                                            *
!comment *  Read the LOGSNAP tape and create a job summaries file.   *
!comment *  Job responds to questions as follows:                    *
!comment *                                                            *
!comment *       LP                 ENTER 'LIST' DEVICE>              *
!comment *       s###,e###          ENTER STARTING,STOPPING LOGFILE>  *
!comment *       TAPE               IS INPUT FROM DISC OR TAPE?>      *
!comment *       N x 21             TYPE 0> ... TYPE 18>, TYPE 46,    *
!comment *                          TYPE 47>  (Logging Events).       *
!comment *       Y                  JOB SUMMARIES?>                   *
!comment *       N                  LONGFORM>                         *
!comment *       mm/dd/yy           ENTER STARTING DATE> MM/DD/YY     *
!comment *       mm/dd/yy           ENTER STOPPING DATE> MM/DD/YY     *
!comment *       <cr>               USER.ACCOUNT>                     *
!comment *       <cr>               LDEV>                             *
!comment *       Y                  CREATE A JOB SUMMARIES FILE?>     *
!comment *                                                            *
!comment *-----------------------------------------------------------*
!PURGE JOBS9002
!FILE JOBSUM=JOBS9002;REC=80,8,F,BINARY;DISC=20000
!FILE LIST=$NULL
!TELLOP; *-----------------------------------------------------------*
!TELLOP; *  Please mount tape: ?\CHAR,6,Tape Number\ for LOGAUDIT.
!TELLOP; *-----------------------------------------------------------*
!RUN LOGAUDIT.PUB.TELESUP
LP
0,9999
TAPE
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
Y
N
02/01/90
02/28/90


Y
!TELLOP; *-----------------------------------------------------------*
!TELLOP; *  LOGAUDIT for 02-90 has completed.
!TELLOP; *-----------------------------------------------------------*
!comment *-----------------------------------------------------------*
```

```
!comment *  PART 2:                                                  *
!comment *                                                           *
!comment *  Read the job summaries file and write the report to a    *
!comment *  disc file.  The job answers questions as follows:        *
!comment *                                                           *
!comment *  ENTER 'LIST' DEVICE>                       LP            *
!comment *  ENTER INPUT FILENAME>                      JOBSyymm      *
!comment *  ENTER STARTING DATE> MM/DD/YY              mm/dd/yy      *
!comment *  ENTER STOPPING DATE> MM/DD/YY              mm/dd/yy      *
!comment *  SUMMARIZE BY GROUPS OR USERS?>             GROUPS        *
!comment *  USER/GROUP TO BE EXCLUDED?>                <cr>          *
!comment *  PRINT ALL GROUP SUMMARIES?>                N             *
!comment *  PRINT ALL ACCOUNT SUMMARIES?>              Y             *
!comment *  LONG OR SHORT FORMAT?>                     SHORT         *
!comment *  PRINT SELECT GROUPS/ACCOUNTS?>             N             *
!comment *                                                           *
!comment *-----------------------------------------------------------*
!comment *  Produce a disc file to edit and download.                *
!comment *-----------------------------------------------------------*
!PURGE JA9002
!BUILD JA9002;REC=-134,,F,ASCII;NOCCTL
!FILE LIST=JA9002,OLD;DEV=DISC
!RUN LOGREPT.PUB.TELESUP
LP
JOBS9002
02/01/90
02/28/90
GROUPS

N
Y
SHORT
N
!comment *-----------------------------------------------------------*
!comment *  Produce a printed report.                                *
!comment *-----------------------------------------------------------*
!RESET LIST
!FILE LIST=JA9002R;DEV=LP,8
!RUN LOGREPT.PUB.TELESUP
LP
JOBS9002
02/01/90
02/28/90
GROUPS

N
Y
SHORT
N
!TELLOP *------------------------------------------------------------*
!TELLOP; *  LOGREPT for 02-90 has completed.
!TELLOP *------------------------------------------------------------*
!comment *-----------------------------------------------------------*
!comment *  PART 3:                                                  *
!comment *                                                           *
!comment *  Reformat the job accounting report for use by Lotus.     *
!comment *  Place a leading zero before a decimal point.             *
!comment *-----------------------------------------------------------*
!PURGE JA9002TN
!TDP
T JA9002
DELETEQ 1
DELETEQ 3/4
CHANGEQ 'PM'  TO 'PM"'     IN 2
CHANGEQ 2     TO ',      "' IN 2
CHANGEQ 1/8   TO ''        IN ALL
CHANGEQ 1     TO '"'       IN ALL
CHANGEQ 10    TO '" '      IN 5/LAST
CHANGEQ 29/63 TO ''        IN 5/LAST
CHANGEQ 36/52 TO ''        IN 5/LAST
CHANGEQ ' .'  TO '0.'      IN 5/LAST
ADD 2
"ACCOUNT"   "#"   "#""CONNECT" "CPU"    "DISC" "PRINT"   "TAPE"
```

Appendix B.

```
"   NAME" "JOBS""SESN""HOURS""HOURS" "RECORDS""PAGES""RECORDS"
//
SET RIGHT=80
SET LENGTH=80
KEEP JA9002TN,UNN
EXIT
!TELLOP; *-----------------------------------------------------------*
!TELLOP; *  REFORMT for 02-90 has completed.
!TELLOP; *-----------------------------------------------------------*
!SUBMIT JAYYMMJ3.JA.SYS
!TELLOP; *-----------------------------------------------------------*
!TELLOP; *  JAYYMMJ1 for 02-90 has completed.
!TELLOP; *-----------------------------------------------------------*
!EOJ
```

Appendix B.

Job JAYYMMJ3: Monthly Job to Create Edit and Transfer Files

```
!JOB JAYYMMJ3,MANAGER.SYS,JA;INPRI=13;OUTCLASS=LP,8
!TELLOP; *-----------------------------------------------------*
!TELLOP; *  JAYYMMJ3 for 02-90 has started.
!TELLOP; *-----------------------------------------------------*
!comment *-----------------------------------------------------*
!comment *  PART 2A:                                           *
!comment *                                                     *
!comment *  Reformat the output of the REPORT command for use by *
!comment *  2020 on the VAX.                                   *
!comment *-----------------------------------------------------*
!PURGE V9002REP
!FCOPY FROM=REP9002A;&
:      TO=V9002REP;&
:      NEW;SUBSET="  ",1,EXCLUDE
!TDP
T V9002REP
CHANGEQ 13/21 TO "    DISC " IN 1
CHANGEQ 31/39 TO "     CPU " IN 1
CHANGEQ 49/57 TO " CONNECT " IN 1
CHANGEQ 58/66 TO "" IN ALL
CHANGEQ 40/48 TO "" IN ALL
CHANGEQ 22/30 TO "" IN ALL
KEEP V9002REP,UNN
EXIT
!comment *-----------------------------------------------------*
!comment *  PART 2B:                                           *
!comment *                                                     *
!comment *  Reformat the output of LOGREPT (destined for Lotus) for *
!comment *  use by 2020 on the VAX.                            *
!comment *-----------------------------------------------------*
!PURGE V9002LOG
!FCOPY FROM=JA9002TN;&
:      TO=V9002LOG;NEW
!TDP
T V9002LOG
CHANGEQ 10          TO ''           IN 2/LAST
CHANGEQ 1           TO ''           IN 2/LAST
CHANGEQ 'S""SESN'   TO 'S SESN '    IN 3,LIT
CHANGEQ '"#"  "#"'  TO ' #     #'   IN 2,LIT
CHANGEQ '"'         TO ' '          IN 2/LAST,LIT
KEEP V9002LOG
EXIT
!comment *-----------------------------------------------------*
!comment *  PART 3:                                            *
!comment *                                                     *
!comment *  Transfer both editted files to the VAX for use with *
!comment *  2020.                                              *
!comment *-----------------------------------------------------*
!DSCOPY
V9002REP &
TO "[.JA]GEM_9002REP.TNS":HPVAX[SENNB:xxxxxxxx]
V9002LOG &
TO "[.JA]GEM_9002LOG.TNS":HPVAX[SENNB:xxxxxxxx]
//
!TELLOP; *-----------------------------------------------------*
!TELLOP; *  JAYYMMJ3:  REFORMT for REPORT has completed.
!TELLOP; *  JAYYMMJ3 for 02-90 has completed.
!TELLOP; *-----------------------------------------------------*
!EOJ
```

Appendix B.

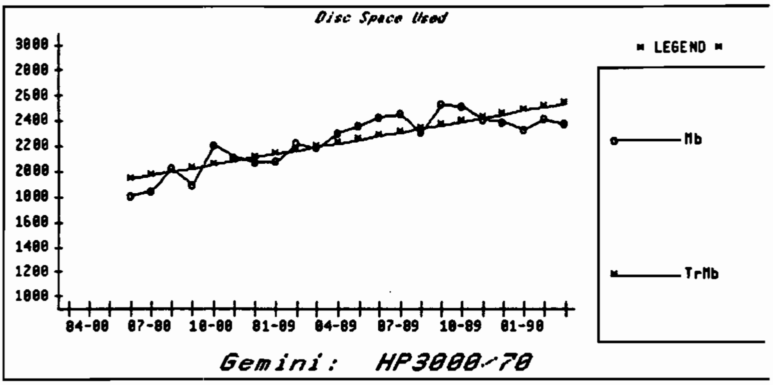
Sample Spreadsheet
Data Generated from the REPORT Command

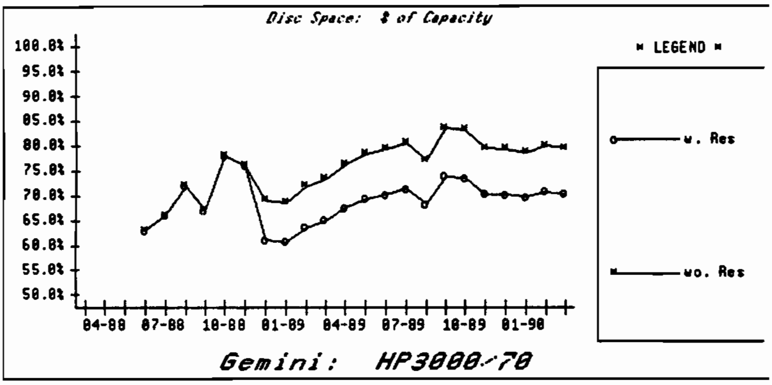
```
DATE:  02-90  SYSTEM:  GEMINI   FILE:  GEM_9002REP

DATE:  02-90  SYSTEM:  GEMINI   FILE:  GEM_9002REP
ACCOUNT      DISC     CPU    CONNECT  DISC SORT            CPU SORT             CONNECT SORT
@          9428538  822055   346720             9428538             822055               346720
              2414

ACAF         44796       88       49  IRIS      4376471  LAMONT     399183  SYS        70807
ATHLETIC        63       56      152  PREX       861083  IRIS        69103  LAMONT     68242
BECKER        9729    36550    42458  BSI        688927  CONTROL     61781  WELLS      44871
BRADMARK     14550        0        0  LAMONT     536841  FINANCE     60783  BECKER     42458
BSI         688927    35155     5902  SYS        522360  BECKER      36550  FINANCE    38015
CAR           4011      119      148  WELLS      330963  BSI         35155  REG        28595
CDR             17        0        0  POWERHS    304118  MJS3000     25322  FINAID     14099
COGNOS      116641       14      178  HPOFFICE   200983  REG         23733  POWERHS    10817
CONTROL      26666    61781      932  FINANCE    176291  WELLS       21601  SEC         8889
FINAID       57080    14125    14099  PLANT      175584  FINAID      14125  BSI         5902
FINANCE     176291    60783    38015  PROD       173940  POWERHS     12712  IRIS        4635
FOOD         25158      154      139  TELAMON    118794  LIFE         9036  STUAFF      2360
HPOFFICE    200983     5433       60  COGNOS     116641  INFOWORK     8178  LIFE        1128
HPSPEP        4479        0        0  SECURITY    88799  SECURITY     7525  CONTROL      932
HPWORD           0        0        0  SUPPORT     69319  PROD         6713  PER          825
INFOWORK     15594     8178      543  OCS         63756  TELESUP      6456  TELAMON      636
IRIS       4376471    69103     4635  REG         62958  HPOFFICE     5433  INFOWORK     543
ITF3000       5122        0        0  FINAID      57080  SEC          4641  OCS          496
LAMONT      536841   399183    68242  MJS3000     51610  VAX          3522  PROD         465
LIB          17265      211      217  ACAF        44796  SYS          3476  STZ          365
LIFE         20434     9036     1128  ROBELLE     42182  TELAMON      2860  PLANT        353
MENUDEV       1857        0        0  TELESUP     40668  STUAFF       1089  LIB          217
MJS3000      51610    25322        0  STUAFF      39455  PREX          729  VAX          179
MUSEUM       14445      469       75  VESOFT      36249  OCS           609  COGNOS       178
OCS          63756      609      496  CONTROL     26666  MUSEUM        469  ATHLETIC     152
PACHNMC       8452        0        0  FOOD        25158  PER           459  CAR          148
PER           3240      459      825  TECH        22782  LIB           211  FOOD         139
PLANT       175584       85      353  LIFE        20434  FOOD          154  MUSEUM        75
POWERHS     304118    12712    10817  LIB         17265  CAR           119  HPOFFICE      60
PREX        861083      729       46  UNION       17054  ACAF           88  ACAF          49
PROD        173940     6713      465  INFOWORK    15594  PLANT          85  PREX          46
REG          62958    23733    28595  BRADMARK    14550  ATHLETIC       56  SIR           38
RJE              0        0        0  MUSEUM      14445  STZ            56  TELESUP        6
ROBELLE      42182        0        0  STZ         10812  SIR            29  TYM            0
SEC            563     4641     8889  BECKER       9729  COGNOS         14  SUPPORT        0
SECURITY     88799     7525        0  STAFF        9593  TOUR            0  TECH           0
```

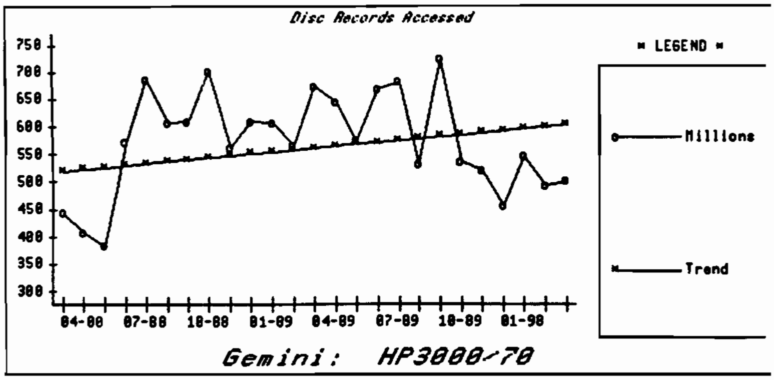
Appendix B.

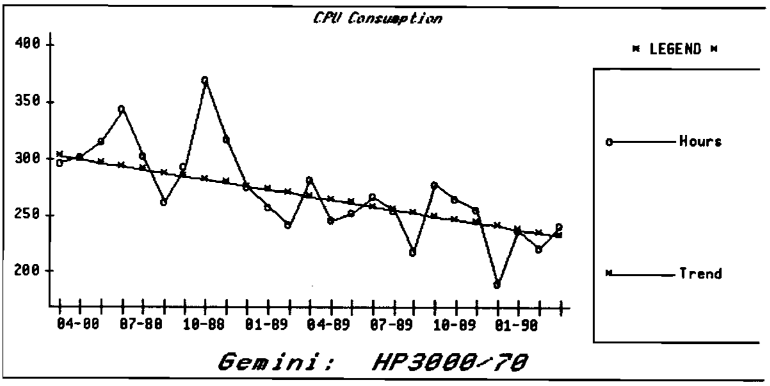
Sample Spreadsheet
Data Generated from the LOGREPT Program

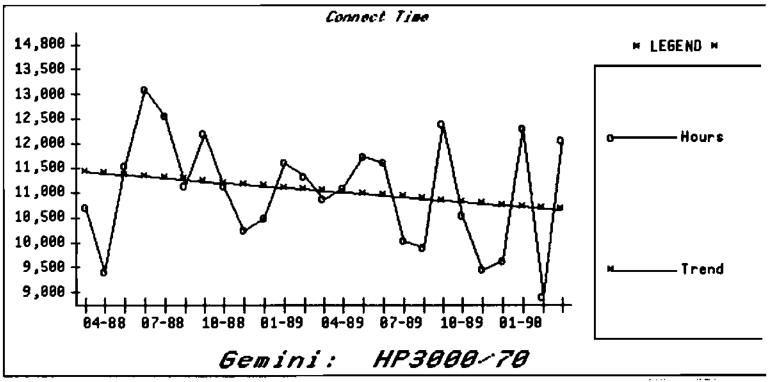
```
DATE:  02-90  SYSTEM:  GEMINI   FILE:  GEM_9002LOG

05 PM"

DISC SORT            CPU SORT          CONNECT SORT       PRINT SORT       JOBS SORT       SESN SORT
           494358538           220.86             8890.94                0           4796            4497

FINANCE  196878400  LAMONT    109.83  LAMONT    1610.68  PROD          0  CONTROL   1649  LAMONT    1322
LAMONT   108360896  IRIS       17.9   SYS        925.07  PREX          0  VAX        840  FINANCE    539
IRIS      68472288  CONTROL    17.16  HPOFFICE   922.63  SEC           0  LAMONT     699  WELLS      495
BECKER    31610360  FINANCE    15.89  WELLS      755.87  REG           0  FINANCE    638  BECKER     459
BSI       17976232  BECKER     10.15  BECKER     734.98  POWERHS       0  WELLS      142  REG        327
MJS3000    9563814  BSI         9.77  FINANCE    710.72  OCS           0  REG        129  FINAID     266
REG        9529900  REG         6.59  INFOWORK   687.03  MUSEUM        0  FINAID     107  BSI        178
FINAID     7968355  WELLS          6  REG        494.73  PLANT         0  BECKER     106  SEC        136
WELLS      6246802  MJS3000     4.52  MJS3000     338.8  PER           0  BSI         59  POWERHS    113
TELESUP    5855659  FINAID      3.92  CONTROL    287.52  TELESUP       0  HPOFFICE    57  IRIS       101
SYS        4423741  POWERHS     3.53  FINAID     253.62  TELAMON       0  IRIS        57  VAX         96
INFOWORK   4410828  LIFE        2.51  POWERHS    193.18  WELLS         0  INFOWORK    55  STUAFF      76
SEC        4193271  SECURITY    2.09  TELESUP    185.55  VAX           0  POWERHS     54  SYS         70
PROD       3442374  PROD        1.86  TELAMON    179.63  SYS           0  TELAMON     41  PER         62
LIFE       3108295  TELESUP     1.58  BSI         150.9  SIR           0  SECURITY    35  TELAMON     52
POWERHS    3028188  INFOWORK    1.42  IRIS       150.87  SECURITY      0  PROD        32  LIFE        44
CONTROL    2847007  HPOFFICE    1.33  SEC        148.15  STZ           0  TELESUP     27  PROD        20
TELAMON    1239238  SEC         1.29  STUAFF      39.55  STUAFF        0  LIFE        23  ACAF        20
VAX        1176851  VAX         0.98  LIFE        28.12  COGNOS        0  MJS3000     13  LIB         17
SECURITY   1156883  SYS         0.83  VAX         17.78  CAR           0  SYS          9  CAR         16
HPOFFICE   1004584  TELAMON     0.56  PROD        15.48  FINAID        0  PREX         6  OCS         13
STUAFF      778489  STUAFF       0.3  PER         13.75  CONTROL       0  FOOD         6  FOOD        11
PER         290028  PREX         0.2  OCS          8.27  BSI           0  STUAFF       4  INFOWORK     9
LIB         238419  OCS         0.17  SECURITY     6.98  ACAF          0  MUSEUM       4  ATHLETIC     9
MUSEUM      134972  PER         0.13  STZ          6.08  $SYSTEM       0  PLANT        4  SIR          7
OCS         122178  MUSEUM      0.13  PLANT        5.95  BECKER        0  STZ          0  HPOFFICE     7
PREX         77486  LIB         0.06  LIB          3.62  ATHLETIC      0  $SYSTEM      0  STZ          7
CAR          46230  FOOD        0.04  COGNOS       2.97  LIB           0  ACAF         0  CONTROL      6
ACAF         39924  CAR         0.03  ATHLETIC     2.53  LAMONT        0  ATHLETIC     0  PLANT        6
FOOD         36941  PLANT       0.02  CAR          2.47  MJS3000       0  PER          0  MUSEUM       5
PLANT        30914  ATHLETIC    0.02  FOOD         2.43  LIFE          0  OCS          0  PREX         5
$SYSTEM      26667  ACAF        0.02  PREX         1.93  IRIS          0  LIB          0  COGNOS       2
ATHLETIC     23273  STZ         0.02  MUSEUM       1.65  FOOD          0  COGNOS       0  TELESUP      1
STZ           7994  SIR         0.01  ACAF         0.82  FINANCE       0  SIR          0  $SYSTEM      0
SIR           7135  $SYSTEM        0  SIR          0.63  INFOWORK      0  CAR          0  MJS3000      0
COGNOS        3922  COGNOS         0  $SYSTEM         0  HPOFFICE      0  SEC          0  SECURITY     0
```

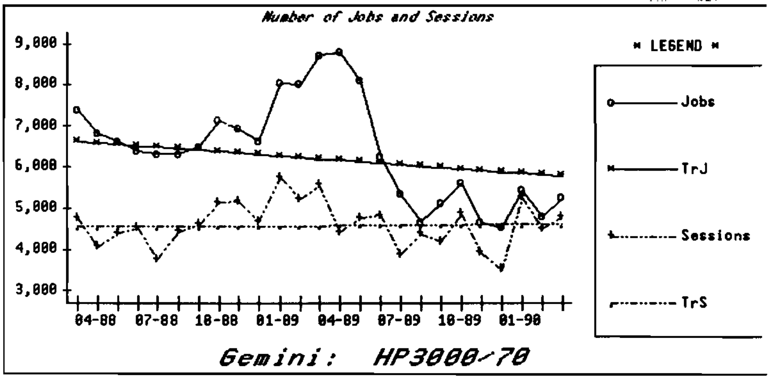

Disc Space Used
3000
2800
2600
2400
2200
2000
1800
1600
1400
1200
1000
04-88
07-88
10-88
01-89
04-89
07-89
10-89
01-90
* LEGEND *
Mb
TrMb
Gemini: HP3000/70


Disc Space: % of Capacity
100.0%
95.0%
90.0%
85.0%
80.0%
75.0%
70.0%
65.0%
60.0%
55.0%
50.0%
04-88
07-88
10-88
01-89
04-89
07-89
10-89
01-90
* LEGEND *
w. Res
wo. Res
Gemini: HP3000/70


Disc Records Accessed
750
700
650
600
550
500
450
400
350
300
04-88
07-88
10-88
01-89
04-89
07-89
10-89
01-90
* LEGEND *
Millions
Trend
Gemini: HP3000/70


CPU Consumption
400
350
300
250
200
04-88
07-88
10-88
01-89
04-89
07-89
10-89
01-90
* LEGEND *
Hours
Trend
Gemini: HP3000/70


Connect Time
14,000
13,500
13,000
12,500
12,000
11,500
11,000
10,500
10,000
9,500
9,000
04-88
07-88
10-88
01-89
04-89
07-89
10-89
01-90
* LEGEND *
Hours
Trend
Gemini: HP3000/70


Number of Jobs and Sessions
9,000
8,000
7,000
6,000
5,000
4,000
3,000
04-88
07-88
10-88
01-89
04-89
07-89
10-89
01-90
* LEGEND *
Jobs
TrJ
Sessions
TrS
Gemini: HP3000/70

Beyond SYSGEN/VOLUTIL: Volume Management on MPE XL

Sandra M. Iwamoto
Hewlett-Packard Company
19111 Pruneridge Avenue
Cupertino, CA 95014

So here you are–in the middle of your worst nightmare. It's midnight Friday morning and you're into your third hour of a RESTORE. You've had to do an INSTALL on your MPE XL system (RELOAD in MPE V/E -ese) because one of your 10 HP7937 disk drives entered the "Twilight Zone" this evening, taking with it the system you need to do your most critical processing for the night.

As you sit there, cursing the fates and wondering how you're going to finish your night's processing once the system is back up, you ask yourself the inevitable question: "how did I get into this, and HOW DO I MAKE SURE IT NEVER HAPPENS AGAIN?"

Volume Management on MPE XL is designed to provide you with the functionality you'll need to address this question. Volume Management allows you to partition your data into separate, individually manageable sets. Only the system's volume set must be available in order for your MPE XL system to continue functioning. Because of this partitioning, the frequency and length of INSTALLs can be reduced, resulting in decreased recovery time.

This paper will provide you with an overview of Volume Management on MPE XL, and give you the information you'll need to most effectively take advantage of its benefits.

VOLUME MANAGEMENT BASICS

Volume Management is part of the basic MPE XL operating system. It views your disk space in terms of the following subsets:

- Volumes
- Volume Sets
- Volume Classes

A *volume* is a single disk pack. Each volume has its own volume label, free space map, file label table, and of course, files. In addition, each volume is a member of a volume set.

A *volume set* is made up of one or more volumes. One of the volumes in each volume set must be designated as the master volume of that volume set, while the others are member volumes. The main difference between the master volume of a volume set and the member volumes is that the master volume contains the root of the directory of accounts, groups, and files associated with that volume set, and it also contains the Volume Set Information Table (VSIT) for that volume set. More on this later.

There are two different types of volume sets:

- MPE XL System Volume Set

- User Volume Sets

There is only one system volume set on each MPE XL system. It is the only required volume set on the system, and it must be mounted in order for the MPE XL system to function properly. It is created automatically when your MPE XL system is initially configured, and LDEV 1 is its master volume.

There can be a maximum of 31 user volume sets mounted on each MPE XL system in addition to the system volume set. User volume sets are optional. They do not need to be mounted in order for the MPE XL system to function. However, if the data on them is needed for any particular application, then they must be mounted in order for that application to run properly.

Volume classes can be used to define logical groups within a volume set, further subsetting the disk space on an MPE XL system. Volumes in a volume set can be grouped into volume classes, and then files which should reside within a particular volume class can be built with that restriction. Volumes may belong to more than one volume class, so volume classes may overlap within a volume set.

The Volume Set Information Table (VSIT), which resides on the master volume of each volume set, contains information describing that volume set. This information includes the name of the volume set, the number and names of the volumes and volume classes in the volume set, and the volumes in each volume class. The information in the VSIT is maintained using the VOLUTIL utility.

NOTE

On MPE V/E systems, volume sets are called domains. The system volume set equivalent is the system domain, and user volume sets are known as private volume sets. The collection of private volume sets on an MPE V/E system is called the private volume domain. Unlike MPE XL, where Volume Management handles both system and user volume sets, MPE V/E uses a special subsystem, the Private Volume Facility, to handle private volume sets.

WHY USE USER VOLUME SETS?

The use of user volume sets lies at the heart of achieving the most effective use of Volume Management on MPE XL. If used correctly, user volume sets will provide the following benefits:

- Improved disk space allocation and management
- Increased data privacy and security
- Reduced downtime

IMPROVED DISK SPACE ALLOCATION AND MANAGEMENT

After a user volume set is defined, you must define which groups and accounts (and therefore files) will reside on that volume set. This is done through the appropriate accounting structure commands; i.e. :NEWACCT, :ALTACCT, :NEWGROUP, :ALTGROUP. By defining which groups and accounts will reside on any given volume set, you are telling the system that all files in the specified groups and accounts are to be built on that volume set, and conversely, that files belonging to other groups and accounts are not to be built (i.e. take up space) on that volume set. This allows you to control and manage your disk space allocation based on an accounting structure relationship, which on many systems also translates into a functional or application-based relationship.

If you want even more control of your disk space allocation within a volume set, you may choose to define volume classes for that volume set. Then, after associating one or more volumes to a specific volume class (remember, volume classes may overlap), you can require that files be restricted to that class. Files built with this restriction will only be allocated space from that volume class. This specification is done using the :BUILD, :FILE, or :RESTORE commands, or the FOPEN and HPFOPEN intrinsics. As an example, some people find this functionality useful in order to ensure that their databases reside on different physical volumes from the database log files when their logfiles are used for rollback recovery. For people who use rollforward recovery, this can be achieved without the use of volume classes; you can just build your logfiles in different groups from the databases, then associate those groups with different volume sets.

And finally, you can use the most granular level of control: you can specify that a file be built on a specific volume. This is also done with the :BUILD, :FILE, or :RESTORE commands, or the FOPEN and HPFOPEN intrinsics.

Note that you can specify that a file be restricted to a particular volume class or to a particular volume even in the system volume set; this functionality is not limited to user volume sets.

INCREASED DATA PRIVACY AND SECURITY

Because you can restrict groups, accounts, and files to particular volume sets, the use of user volume sets provides you with more options for controlling access to the data in those files.

You may, if you choose, decide to only have a single user volume set on a system, and put your most private data on that volume set. Then, you could restrict access to that data by ensuring that only valid accessors are given UV capability, which is the MPE XL user/account capability which allows access to user volume sets. Only users with "UV" capability will be able to access that user volume set; the others will only be able to access the system volume set.

Or, if you want to have more than one user volume set mounted simultaneously on a system, and have sensitive data that you want to protect, you can define a volume set which contains only that private data (the other volume sets would contain "standard security" data). Then, since user volumes need not be mounted at all times, you can logically remove the volume set containing this sensitive data from the system; i.e. make this data logically inaccessible by users. Or, you can make this data physically unavailable by powering down the disk drives, and removing the disk packs if you have removable media. Note that you can choose to use more than one volume set in this manner.

REDUCED DOWNTIME

To discuss this topic, let's return to the scenario outlined at the opening of this paper, and see how it could have been changed with the use of user volume sets.

In our scenario, all of the disk drives were defined as members of the default MPE XL system volume set. This means that user files are mixed together with MPE XL's system files within this single volume set. It also means that if anything happens to this volume set, two areas are impacted: the system (because the system volume set contains the data required for MPE XL to operate) and user data (since it's sharing this volume set with MPE XL). The result is that when the disk drive goes down it takes the whole system with it, and recovering the system means fixing the disk drive, doing an INSTALL, rebuilding the system volume set (defining volumes and volume classes), rebuilding the accounting structure for the entire system, and restoring all system and user files. On most systems, this process takes several hours. And since the trend is to have more disk drives/space on MPE XL systems than there were on MPE V/E systems, the downtime in a case like this could be staggering.

If we had partitioned the data on our MPE XL system into one system volume set to hold MPE XL's files, and say, three user volume sets to hold our user data, the scenario would have looked quite different.

If the failing disk drive had been part of the system volume set, we would have impacted system operation, and the MPE XL portion of the data on our system. In this case, recovery would mean the system would be down while we fixed the disc drive, did an INSTALL, rebuilt the system volume set, rebuilt the system directory, and restored all of MPE XL's files. We would not, however, have to restore the files on the user volume sets, since they are not impacted by a disk drive failing in the system volume set. And, since the vast majority of the data on an MPE XL system is user data, not MPE XL files, this scenario could result in decreased downtime of several hours.

If, on the other hand, the failing disk drive had belonged to one of the user volume sets, we would have impacted just the operations on that volume set; the system could continue to operate successfully, and users of applications which reside on the two unaffected user volume sets could continue to operate successfully. In this case, recovery would mean fixing the drive, rebuilding the user volume set, rebuilding the accounting structure on that volume set, and restoring the files on that volume set. And all of this can be done in parallel with normal system operations for the users of applications on the two unaffected user volume sets. And so, "downtime" in this case would only be applicable to users of the application(s) on the user volume set with the failing disk drive. The other users of the system wouldn't experience any downtime at all!

And finally; partitioning your disk drives into system volumes and user volumes decreases the likelihood of your having to do an INSTALL in the first place. This is due to two factors. The first is outlined in the preceding paragraph. The second is simple statistics. While HP's disk drives are very reliable, they do fail occasionally. And taking even a small failure rate, multiplied by the number of drives in the system volume set, will give you a higher effective failure rate for the system as a whole if all of your drives are in the system volume set than if only a few of them are.

SETTING UP VOLUME SETS

This section will discuss the steps necessary to set up system or user volume sets for use on an MPE XL system. As appropriate, distinctions will be made between doing this on a new system and on an existing system where all disks are members of the system volume set.

There are six steps to setting up system or user volume sets for use:

- Connect and configure disks
- Prepare disk media for initialization
- Create volume set and master volume
- Create additional volumes
- Create new volume classes (if any)
- Set up accounting structure associations

CONNECT AND CONFIGURE DISKS

This step simply means connecting the disk drives to your MPE XL system and configuring the LDEVs and paths using SYSGEN.

NEW SYSTEM. For disks (other than LDEV 1) which will be part of the system volume set, you may wish to use SYSGEN's AVOL command to add these volumes to SYSGEN's volume list. This is not required for LDEV 1, because it is automatically defined as part of the system volume set in SYSGEN. Defining all system volumes in SYSGEN's volume list will save you from having to run VOLUTIL to add these volumes if you ever have to do an INSTALL from your backup SLT tape, since they will already be configured and initialized as system volumes in your SYSGEN configuration. You should not add user volumes to SYSGEN's volume list. Doing so will decrease your flexibility in configuring these volumes.

NOTE

The only purpose for adding system volumes into SYSGEN's volume list is to save some time when doing INSTALLs, since the Volume Management portion of START (done after INSTALL) will initialize all volumes in this list automatically. So once the system is up, you won't have to use VOLUTIL to define these volumes.

Also, note that adding system volumes into SYSGEN's volume list is analogous to adding system volumes into MPE V/E's volume table in SYSDUMP or during a RELOAD.

EXISTING SYSTEM. Before you can begin to use a disk which is currently part of the system volume set as a user volume, you must release it from the system volume set. To do so, you must do an INSTALL on the system. Prior to the INSTALL, you must generate a SYSGEN SLT tape which contains the volume list you wish to define as your system volume set in SYSGEN (see discussion under "New System" above), as well as doing a full backup including your directory. Note that in order to free this volume from the system volume set, it cannot be part of the volume list you've defined on your new SLT tape. Then do an INSTALL with this new SLT.

Regardless of whether or not you are changing your existing disk allocation for the system volume set, if you are adding new disks to the system at this time, you must connect and configure them using SYSGEN. In addition, if one or more of these new volumes will become part of your system volume set, you may choose to add it to SYSGEN's volume list (see discussion under "New System" above).

PREPARE DISK MEDIA FOR INITIALIZATION

To prepare your disk media for initialization by Volume Management, you must get each volume into either a SCRATCH or an UNKNOWN state. To do this, you must use the VOLUTIL utility, and you must be logged on to a user with Create Volumes (CV) capability.

The volume which you are preparing may start from any one of the following volume states. You can check the current state of a volume using either the :DSTAT ALL or VOLUTIL's SHOWSET DSTATUS or SHOWVOL DSTATUS commands.

MASTER	A volume in this state is currently defined as the master volume of some volume set. It may be a system or user volume set. A volume in this state is currently mounted on the system and available for use.
MEMBER	A volume in this state is currently defined as a member volume of some volume set. It may be a system or user volume set. A volume in this state is currently mounted on the system and available for use. In order for a volume to be in this state, the master volume of its volume set must be in the MASTER state.
LONER	A volume in this state is currently unavailable for use. It may be a master volume or member volume of some volume set. Exception: LDEV 1 is never in the LONER state. A volume is in the LONER state when its master volume is not currently mounted or has been made unavailable by the :VSCLOSE command.
SCRATCH	A volume in this state is currently unavailable for use. At this point, it is neither a master nor a member volume. This volume may once have been a master or member volume of some set, but because it has been scratched, the data on the volume has been declared to be of no importance. This volume may be "unscratched" with VOLUTIL's UNSCRATCHVOL command. A volume in SCRATCH state is ready to be initialized by Volume Management.
UNKNOWN	A volume in this state is currently unavailable for use. A volume in

the UNKNOWN state does not have a label that the system can recognize. This volume may be a new volume, it may be a volume from another system, or it may be a volume which has been formatted. A volume in this state is ready to be initialized by Volume Management. Note that a volume which was used as a Private Volume on an MPE V/E system will be mounted in this state.

As indicated above, volumes which are in the SCRATCH or UNKNOWN state are ready for initialization. Volumes which are in the MASTER, MEMBER, or LONER state must be moved to the SCRATCH or UNKNOWN state before they can be initialized.

To move a volume from the MASTER or MEMBER state to the SCRATCH state, you must first dismount it and make it unavailable for use. If this volume is part of the system volume set, you must do an INSTALL in order to free this volume. Note that the SLT you use in your INSTALL must not have this volume defined as part of SYSGEN's volume list, or this volume will not be freed for initialization.

If this MASTER or MEMBER volume is part of a user volume set, you must first dismount that volume set and make it unavailable for use. To do this, you must first ensure that no one is currently using this user volume set. You can use the :VSUSER command to check this. Once you have ensured that there are no accessors of this user volume set, you can issue the :VSCLOSE command to move the volumes in this user volume set from MASTER or MEMBER state to LONER state. Once the volumes are in the LONER state, issue the VOLUTIL SCRATCHVOL command on the desired volumes to move them to the SCRATCH state, making them available for initialization. Note that if you scratch the master volume of a volume set, the volume set is no longer usable.

To move a volume from the LONER state to a SCRATCH state, you need only issue VOLUTIL's SCRATCHVOL command on the volume.

NOTE

It is strongly recommended that you use the :DSTAT ALL and VOLUTIL's SHOWSET DSTATUS or SHOWVOL DSTATUS commands frequently as you are preparing your disk media to ensure that this process goes smoothly.

For MPE V/E -ers: MASTER/MEMBER = MOUNTED, LONER = AVAIL, SCRATCH = AVAIL *SCRATCH*, UNKNOWN = FOREIGN. The VOLUTIL equivalent is VINIT. VOLUTIL's commands differ somewhat from VINIT's, but the functionality is much the same.

CREATE VOLUME SET AND MASTER VOLUME

Defining a volume set and its master volume is done in a single step, much like creating a new account and account manager. Since the system volume set and its master (LDEV 1) are automatically created when the system is INSTALLed, this section will only discuss the creation of user volume sets.

User volume sets are created using VOLUTIL's NEWSET command. You must be logged on to a user with CV capability in order to use this command. Also, the volume which will be defined as the master volume of a new user volume set must be online in either the SCRATCH or UNKNOWN state, since it will be initialized during the execution of this command. Consult the "Volume Management Reference Manual" (P/N 32650-60006), for details on this command.

CREATE ADDITIONAL VOLUMES

Now that you have your system volume set and one or more user volume sets (and their associated master volumes) defined, you can add additional volumes to these volume sets.

As stated in the section on "Connect and Configure Disks", you may choose to define/initialize your system member volumes using SYSGEN instead of VOLUTIL. If you choose not to, then the steps for creating and initializing system member volumes are the same as for user member volumes.

Creating and initializing a new member volume requires the use of VOLUTIL's NEWVOL command and, optionally, the INITVOL command. You must be logged on to a user with CV capability in order to use these commands. Also, the target volume must be in the SCRATCH or UNKNOWN state, and the master volume of the target volume set must be mounted and in the MASTER state. If the master volume is currently in the LONER state, you must issue the :VSOPEN command to move the master volume to the MASTER state.

Before we continue, let me explain the use of VOLUTIL's NEWVOL and INITVOL commands. The NEWVOL command can be used to create and initialize a volume, or, if the volume is not currently online and accessible in the proper state, it will just create the volume. Then, once the disk is online and accessible in the proper state (i.e. SCRATCH or UNKNOWN), you can use the INITVOL command to initialize the volume.

The creation/definition of a volume means that an entry for this volume has been added to the VSIT of the master volume in the target volume set. This entry includes information such as the volume name, any associated volume classes, and permanent and transient space allocations for the volume. You do not need access to the target volume in order to create/define it; you only need access to the master volume of the target volume set (i.e. it must be in the MASTER state).

The initialization of a volume means writing the volume label to that disk, creating the volume's label table, and creating the volume's free space map. It is during this step that you need access to the target volume (i.e. it must be in the SCRATCH or UNKNOWN state).

The most straightforward way to create and initialize a volume then, is to have the master and target volumes online and in the proper states, and to use VOLUTIL's NEWVOL command, specifying the LDEV on which the target volume resides. This will cause the volume to be created and initialized all in one step. Otherwise, you'll have to use both the NEWVOL and INITVOL commands. Consult the "Volume Management Reference Manual" (P/N 32650-60006), for details on these commands.

It is strongly recommended that you use the :DSTAT ALL and VOLUTIL's SHOWSET STRUCT commands frequently throughout this process to ensure that you get the expected results.

When you have successfully used the NEWVOL or NEWVOL/INITVOL combination of commands to create and initialize your volumes, they will be mounted and ready for use.

NOTE

> Volumes can be added to the system volume set or to a user volume set at any time; e.g. during normal processing. But, once volumes have been added to a system or user volume set, they cannot be deleted through VOLUTIL. The only way to delete a system volume is to do an INSTALL excluding that volume from the system volume set. The only way to delete a user volume from its user volume set is to scratch and rebuild that volume set.

CREATE NEW VOLUME CLASSES (IF ANY)

As indicated in the section on "Improved Disk Space Allocation and Management", volume classes can be used to control the allocation of disk space at a more granular level than the one provided by associating groups and accounts with particular volume sets.

Creating a volume class requires the use of VOLUTIL's NEWCLASS command. You must be logged on to a user with CV capability in order to use this command.

First, some basics: if no volume class(es) are are specified for a volume when it's created/initialized, it will belong to the default volume class DISC. If you wish to associate other volume classes to a volume, you must specify each of these volume classes (including DISC, if desired) at volume creation/initialization, or, if you wish to add an existing volume to a volume class, by using VOLUTIL's NEWCLASS or EXPANDCLASS commands. Note that the only way to exclude the default DISC volume class from a volume is to specify the list of associated volume classes (excluding DISC) at volume creation/initialization.

As an example, if you have created a volume set which contains four disks, and you want to use three of these disks for data files, program files, job stream files, and databases, and want to use the fourth disk to hold only your database logfiles (to use in rollback recovery), you would need to define a new class, e.g. LOGDISC. This new class must be defined before you create the fourth, dedicated, volume, since you'll need to add this volume into the new LOGDISC class when the volume is created. And, when you specify which class this fourth volume belongs to, you should specify just the LOGDISC class. This will result in a volume which belongs exclusively to the LOGDISC volume class; it will not belong to the default DISC class. This will prevent other files from being built on this disk.

Consult the "Volume Management Reference Manual" (P/N 32650-60006), for details on the use of the NEWCLASS and EXPANDCLASS commands.

For best results, you should use VOLUTIL's SHOWSET STRUCT and SHOWCLASS commands frequently to check your current status and progress during this process.

NOTE

Volume classes can be added to a volume set, or associated with a specific volume, at any time that the target volume set and/or volume is mounted; e.g. during normal processing. But, once volume classes have been added to a system or user volume set, or associated with a specific volume, they cannot be deleted through VOLUTIL. The only way to delete a volume class from the system volume set, or to change the association of volumes to volume classes in the system volume set, is through SYSGEN (if these volumes are defined in SYSGEN's volume list). In order to ensure that this is done cleanly, you may need to do an INSTALL. The only way to delete a volume class from a user volume set, or to change the association of volumes to volume classes in a user volume set, is to scratch and rebuild that volume set.

SET UP ACCOUNTING STRUCTURE ASSOCIATIONS

At this point, we have defined all of the volume sets on our system, we have allocated our disk volumes into these volume sets, and we have (optionally) allocated our disk volumes into volume classes within these volume sets. This completes the partitioning of our disks into their respective Volume Management subsets. The only remaining step is to allocate our data onto these volume sets. This is done by creating accounting structure-to-volume set associations. This activity is commonly called "homing" groups to volume sets.

Homing groups to volume sets requires the use of some familiar MPE commands, :NEWACCT, :ALTACCT, :NEWGROUP, and :ALTGROUP. The difference is the use of some additional parameters: ;HOMEVS= and ;ONVS=. Let's go through some examples to see how this works.

NEW SYSTEM. For a new system, or one which has just been INSTALLed, you need to define your entire accounting structure from scratch. Whether you enter your commands by hand, or use a job stream which builds your accounts, groups, and users en masse, here's how to associate groups to specific volume sets.

CASE A: Creating a new account homed to the system volume set:

There's no need to show an example for this, since it's the default association for any group and account built on the system.

CASE B: Creating a new account homed (at least in part) to a user volume set:

1. **Create the account entry in the system directory.** Since this account will have some (or all) of its groups homed to a user volume set, you must give this account UV capability, since without it users in this account won't be able to log on or build files in this account (or at least, not into groups which are on a user volume set).

```
:NEWACCT IWAMOTO,SANDRA;CAP=AM,AL,GL,ND,SF,IA,BA,UV
```

2. **Create the account entry in the user volume set directory.** You must create a "twin" account entry in the directory of each user volume set on which you wish

to build files for this account. The user volume set on which you wish to build this account entry is specified with the ;ONVS= parameter.

```
   :NEWACCT IWAMOTO,SANDRA;CAP=AM,AL,GL,ND,SF,IA,BA,UV
                                                    ;ONVS=MYSET
```

3. **Create the group entry in the system directory.** Like the account entry, you must first build the group entry in the system directory. The user volume set (if any) to which you want to "home" this group is specified with the ;HOMEVS= parameter. If you don't specify this parameter, the group will be homed to the system volume set.

```
   :NEWGROUP PVGROUP.IWAMOTO;HOMEVS=MYSET
```

4. **Create the group entry in the user volume set directory.** You must now build the "twin" group entry on the desired user volume set. This volume set name must match the one specified in the ;HOMEVS= parameter in Step 3. Note that a group may only belong to one volume set at any one time, although each group in an account may belong to a different volume set. The user volume set on which you wish to build this group entry is specified with the ;ONVS= parameter.

```
   :NEWGROUP PVGROUP.IWAMOTO;ONVS=MYSET
```

5. **Alter the PUB group's home volume set. (Optional)** The PUB group is built automatically when you create the account entry in the system directory. The PUB group is homed to the system volume set. If you wish to change its home volume set to a user volume set, you must use the following commands:

```
   :ALTGROUP PUB.IWAMOTO;HOMEVS=MYSET (to declare its new home)
   :NEWGROUP PUB.IWAMOTO;ONVS=MYSET  (to build this group entry
                                      in the user volume set
                                      directory)
```

6. **Create any other users for this account.** Users only have entries in the system directory, but any users who will need access to a user volume set (any user volume set) must be given UV capability.

```
   :NEWUSER FRIEND.IWAMOTO;CAP=IA,BA,ND,SF,UV
```

7. **Restore/build files on the system.** This will cause the files to be placed onto their proper home volume sets. Note that files which must be restricted to a particular volume or volume class must be restored or built specifying that volume or volume class restriction.

As you can see in the steps outlined above, the system directory and the directories on the user volume sets are treated as separate entities. Each directory on a volume set, including the system directory, is separately managed. You must issue the appropriate :NEWACCT, :ALTACCT, :NEWGROUP, and :ALTGROUP commands for each volume set you're working with, then match it with the ;HOMEVS= or ;ONVS= parameter, as appropriate. The ;HOMEVS= parameter sets up the home volume set information in the system directory, and the ;ONVS= parameter builds or modifies the specified entry in the specified volume set directory.

It is CRUCIAL that you use the :LISTGROUP, :REPORT, and :REPORT;ONVS= commands frequently as you are building your accounting structure. The :LISTGROUP command will verify the home volume set for each group, the :REPORT command will show you which groups have been built for the specified account(s), and the :REPORT;ONVS= command will show you which groups have been homed to the specified volume set. Issuing these commands at each step of this process will ensure that the group entries are being built as you expect.

NOTE

For MPE V/E -ers: Associating accounting structures to user volume sets on MPE XL is a bit more linear than with private volumes on MPE V/E, but there is no longer any shortcut when building new accounting structures (e.g. :NEWGROUP;VS=MYSET;SPAN).

EXISTING SYSTEM. If you have an existing system and you don't plan to reallocate the disks currently defined in the system volume set to user volume sets, your task (once you've added disks and built your user volume set(s)), is to adjust your accounting structure to take advantage of the system volume set/user volume set disk space partitioning.

Let's take a look at an example of this:

1. **Store all files in the groups and accounts you wish to move.** This is done with a simple :STORE command.

```
:STORE @.@.FINANCE,@.@.MFG,@.@.PERSONEL;*T;SHOW
```

2. **Purge these files from the system volume set.** This is done to free up the disk space on the system volume set. Also, if you don't purge the files, you won't be able to alter the group; you'll get an "IN USE: CAN'T BE PURGED. (CIERR 916)" message when you use the ;HOMEVS= parameter. Note that this is also true when you're moving a group from one user volume set to another.

3. **Add UV capability to the affected accounts and their users.** This is done with the normal :ALTACCT and :ALTUSER commands.

4. **Create entries for these accounts in the appropriate user volume set directory.**

```
:NEWACCT FINANCE,MGR;CAP=AM,AL,GL,ND,SF,IA,BA,UV;ONVS=MYSET
:NEWACCT FINANCE,MGR;CAP=AM,AL,GL,ND,SF,IA,BA,UV;ONVS=LOGSET
:NEWACCT MFG,MGR;CAP=AM,AL,GL,ND,SF,IA,BA,UV;ONVS=MYSET
:NEWACCT PERSONEL,MGR;CAP=AM,AL,GL,ND,SF,IA,BA,UV
                                        ;ONVS=SECURESET
```

5. **Alter the group entries in the system directory.** The existing group entries must have their home volume set altered from the system volume set to the appropriate user volume set.

```
:ALTGROUP PUB.FINANCE;HOMEVS=MYSET
:ALTGROUP DATABASE.FINANCE;HOMEVS=LOGSET
...
```

6. **Create the group entries in the appropriate user volume set directory.** This volume set name must match the one specified in the ;HOMEVS= parameter in Step 5. Note that a group may only belong to one volume set at any one time, although each group in an account may belong to a different volume set. The user volume set on which you wish to build this group entry is specified with the ;ONVS= parameter.

```
:NEWGROUP PUB.FINANCE;ONVS=MYSET
:NEWGROUP DATABASE.FINANCE;ONVS=LOGSET
...
```

7. **Restore the files.** This will cause the files to be placed onto their newly specified home volume sets. Note that files which must be restricted to a particular volume or volume class must be restored or built separately, specifying that volume or volume class restriction.

As indicated in the note in the "New System" discussion, the system directory and each user volume set's directory are managed as separate entities, and it is CRUCIAL that you use the :LISTGROUP, :REPORT, and :REPORT;ONVS= commands frequently as you are altering your accounting structure-to-volume set associations. This will ensure that you end up with the results you expect.

NOTE

The steps outlined above can also be used whenever you wish to change the home volume set of a particular group, even if you wish to change it back to the system volume set. There is, however, an additional step in this process. When you are moving a group from one user volume set to another, you should purge the group entry from its former home volume set's directory using the :PURGEGROUP;ONVS= command.

Once all accounts, groups, and users have been built, and the association of accounts and groups with the volume sets on a system have been completely defined, you can restore your files onto the system. As files are restored, they will automatically be built onto the volume set which houses the group and account to which they belong. The exception to this is files which you wish to restrict to a specific volume or volume class. You must restore or build these files specifying the volume or volume class restriction; otherwise the files will default to the volume set of their group and account.

OPERATIONAL CONSIDERATIONS

So now that you have spent the time to set up your volume sets and have partitioned your data onto these volume sets, how do you work with a system that has user volumes? What are the operational considerations to take into account when dealing with a system with user volumes?

There are seven areas of system management, account management, and operations which are impacted by the use of user volume sets:

- Changing system configuration and/or disk allocation

- Access to user volume sets

- Building files
- Managing accounts, groups, and users
- System backup
- Doing an INSTALL
- Rebuilding a user volume set

CHANGING SYSTEM CONFIGURATION AND/OR DISK ALLOCATION. Changing the system configuration on a system with user volumes, or changing the allocation of disks from one volume set to another, are topics which were covered in the section on "Setting Up User Volumes". Please refer to the appropriate topics in that section for details.

ACCESS TO USER VOLUME SETS. As discussed in the section on "Setting Up User Volumes", a user volume set and the volumes in it are automatically mounted after they have been successfully created and initialized. In addition, any user volume sets whose volumes are connected, configured, and powered on will be automatically accessible at system bootup, just like the system volume set. This means that users with UV capability can log on to groups and accounts on these user volume sets, build, alter, or purge files, etc.

If there is a need to make a user volume set inaccessible, because the volume set contains sensitive data which you wish to "take down", or you wish to scratch and rebuild that volume set, or there's been a hardware failure on one of the disks in that volume set and you need to make the entire volume set unavailable for use, you need to do two things. First, you must make sure no one is currently using that user volume set. You can check this using the :VSUSER command. If there are any users logged onto or accessing that volume set, you must get them to release their use of that volume set. Then, once the volume set has been completely released, you must issue the :VSCLOSE command, specifying that volume set. This will cause the volumes in the user volume set to move to the LONER state, making them inaccessible.

When you need to make this (or any) volume set in the LONER state available for access; i.e. you wish to move the volumes from LONER to MASTER or MEMBER state, you must issue the :VSOPEN command, specifying that volume set.

There are a number of other Volume Management CI commands. Briefly, they are:

VMOUNT	Used to enable and disable the user volumes facility. Whenenver the system is booted, this facility is automatically enabled. If this facility is disabled, user volumes will become inaccessible, even if the volumes are mounted in the MASTER or MEMBER state.
VSRESERVESYS	Used to reserve the specified user volume set on a system-wide basis; i.e. make the user volume set accessible to anyone with UV capability.
VSRELEASESYS	Used to release the specified user volume set on a

	system-wide basis; i.e. globally release the user volume set. This command is used to negate a previous VSRESERVESYS command.
VSRESERVE	Used to reserve the specified user volume set for the user who issues this command. This means that even if the user isn't currently logged onto this volume set, or isn't accessing a file on this volume set, they have "reserved" this volume set, and wish to prevent it from being dismounted.
VSRELEASE	Used to release the specified user volume set for the user who issues this command. This command is used to negate a previous VSRESERVE command.

Consult the "Volume Management Reference Manual" (P/N 32650-60006) or the "MPE XL Commands Reference Manual" (P/N 32650-60002) for details on these commands.

NOTE

For MPE V/E -ers: :VSRESERVESYS = :LMOUNT, :VSRELEASESYS = :LDISMOUNT, :VSRESERVE = :MOUNT, and :VSRELEASE = :DISMOUNT. These MPE V/E commands are still available on MPE XL systems, but the user volume names specified in these commands must be in the MPE V/E format, not the MPE XL format. Note too that there is less operational overhead with MPE XL's user volumes than with MPE V/E's Private Volume Facility. You are not required to issue a :VMOUNT ON,AUTO upon rebooting, nor are you required to issue :LMOUNT and :LDISMOUNT commands upon rebooting or prior to shutting down a system in order to automatically enable or disable private/user volume sets for access. Also, MPE XL doesn't allow the use of the :UP and :DOWN commands for disk drives.

BUILDING FILES. For the most part, once your accounting structure-to-volume set association is set up, building files on a user volume set is exactly the same as on the system volume set. The only external difference arises if you are restricting a file to belong only on a particular volume, or within a particular volume class. Then, you'll need to build that file specifying that volume or volume class restriction. Consult the "MPE XL Commands Reference Manual" (P/N 32650-60002) or the "MPE XL Intrinsics Reference Manual" (P/N 32650-60013) for details on the :BUILD and :FILE commands and the FOPEN and HPFOPEN intrinsics.

As for altering and purging files, there is no difference between files on the system volume set or on user volume sets, nor is there a difference for files with volume or volume class restrictions.

MANAGING ACCOUNTS, GROUPS, AND USERS. If there is one area of system management and account management which is most affected on a day-to-day basis by the use of user volumes, this is it. System managers who are creating, altering, or purging accounts, groups, users, and files need to understand the new and changed commands they must use in order to correctly do their job. Similarly, account managers need to understand the new and changed commands they must use to correctly create, alter, or purge groups, users, and files in their accounts. The changed commands and their use have already been discussed in the section on "Setting Up User Volumes".

To ensure the most trouble-free and successful operation in a user volume environment, system managers and account managers must be trained to understand Volume Management, and the new and altered commands. In addition, they should exercise care whenever making changes to accounts, groups, users, and files under their control. The best way to do this is to repeatedly check the progress of their task by issuing the appropriate :REPORT, :REPORT;ONVS=, and :LISTxxxx commands.

SYSTEM BACKUP. When backing up your system, you do essentially the same thing you do today: cut a SYSGEN SLT tape, and store all your files and your directory. The difference with user volume sets is that now you have more than one directory. You have the system directory, plus one or more user volume set directories. Each one of these must be backed up. To do this, your :STORE command should include the ;ONVS= parameter as well as the ;DIRECTORY parameter. The list of volume sets in the ;ONVS= parameter should include "MPEXL_SYSTEM_VOLUME_SET" for the system directory, followed by the names of the user volume sets on this system. Consult the "MPE XL Commands Reference Manual" (P/N 32650-60002) for details on this command.

DOING AN INSTALL. As mentioned in the section on "Reduced Downtime", the number of times you'll have to do an INSTALL will decrease as a result of the use of user volume sets. In addition, each INSTALL will take less time. If you've configured your system volumes into SYSGEN's volume list, all you'll need to do after you've rebooted from the INSTALL is rebuild (restore) the system directory, and restore the files on the system volume set. If you haven't configured your system volumes using SYSGEN, you'll have to use VOLUTIL to rebuild your system volume set after rebooting and before rebuilding the system directory.

Assuming the data on your user volume sets is intact, which is generally true in situations requiring INSTALLs, you don't need to do anything with the user volume sets after an INSTALL. As long as the disk drives containing the user volume sets are connected, configured, and powered on, the START after your INSTALL will read the information from the VSIT of each master volume and automatically mount all of the user volume sets on your system. If, however, one or more of your user volume sets has sustained some "damage", you'll need to rebuild the damaged volume set(s).

REBUILDING A USER VOLUME SET. Typically, the reason a user volume set would need to be rebuilt is that there was a disk failure on one of the volumes in the set. Rebuilding a user volume set requires the use of VOLUTIL to rebuild the user volume set structure, MPE XL commands to rebuild the accounting structure in that user volume set's directory, and restoring the files onto the volume set. The last two steps can be accomplished by the use of a backup :STORE tape set with the user volume set's directory and files.

RECOMMENDATIONS

In the last two sections on setting up volume sets and considering operational differences in a user volume environment, I covered a lot of information about the nitty gritty of working with user volume sets. In this section, I'll present a list of recommendations from experienced users of user volumes on MPE XL about how to use this feature most effectively. Some of these have already been mentioned.

1. **Use user volumes on any system with 10 or more disk drives.** It can certainly be beneficial on systems with fewer than 10 disks, but it is a MUST on systems with more disk drives.

2. **Put a human-readable label on your disk drives noting the LDEV number and path.** This is not something specific to systems with user volumes. This is a general recommendation for all systems, to help in troubleshooting. When you're having disk problems, the last thing you want or need is to delay the problem resolution while you or your HP CE determine which drive is which.

3. **Minimize the number of volumes in the system volume set.** The most common practice is to put only HP-supplied (i.e. FOS and SUBSYS) files, plus perhaps key third party applications, in the system volume set. (Note that the HPDESK database is an exception because of its size.) In addition, you must ensure that there is enough room for transient space (remember, user volumes don't have any transient space), spoolfiles, and other MPE XL overhead. Based on MPE XL's current size, I would recommend no fewer than 2 HP7937 disks and 1 HP7935 disk. And, you may want to set your transient/permanent space allocation on each system member volume to 100/80, to ensure a comfortable amount of transient space. For an existing system, use DISCFREE to determine your current transient space usage so you can appropriately "size" your system volume set.

4. **Define all system volumes in SYSGEN's volume list.** As indicated in the section on "Connect and Configure Disks", this will save you time in the event of an INSTALL.

5. **For each user volume set, create accounting structure-to-volume set associations based on functional modularity and dependency.** Remember that the one of the strength of user volumes is the ability to affect as few users as possible if one volume set goes down.

6. **Use good judgment when sizing each user volume set.** Each user volume set should contain enough disk space for the files it will hold plus room for growth, plus space for MPE XL overhead such as the user volume set directory, volume label, VSIT, free space maps, label tables, and 96MB for transaction management logfiles.

7. **Separate database log files from their associated databases.** If you use rollforward recovery on a database, build your database log file in a different group from the database, and home that group to a different volume set from the database. If you use rollback recovery, define one volume in the database's volume set as a non-DISC class volume (so database files won't be built there) and build your database log file with a volume class restriction included on that volume.

8. **Minimize the use of volume and volume class restrictions.** These restrictions require more user overhead to manage than just the standard volume set restriction. As much as possible, partition your data at a volume set level to achieve the modularity you desire.

9. **Teach all system managers and account managers about Volume Management.** Volume Management isn't that difficult to understand; it's just that it requires building new habits.

And the best way to ensure that everyone learns the right habits is to give them an understanding of why they're necessary.

10. **Last and most important:** CHECK YOURSELF AT EACH STEP ALONG THE WAY. As I said before, Volume Management isn't that difficult once you understand what you're doing and why. The trick is, if you check yourself before and after each step, it's easier to catch and correct a mistake or an omission; either of which can lead to surprising, sometimes unfortunate, results. Use all the commands at your disposal: :DSTAT ALL, :REPORT, :REPORT;ONVS=, :LISTxxxx, :VSUSER, VOLUTIL's SHOWxxxx commands.

NOTE

For MPE V/E -ers: Volume Management on MPE XL is completely different code from Private Volumes on MPE V/E. Moreover, unlike MPE V/E's Private Volumes, MPE XL's system volume set and user volume sets use the same code; user volume management was not an "add-on" feature. This means that there is no distinguishable performance difference or reliability difference between using system volumes and using user volumes.

AVOIDING 'GOTCHAS'

In this, the final section, I've provided a list of the most common pitfalls people have run into with Volume Management. This list is not restricted to problems relating to user volumes; in fact, the majority of problems is common to both system and user volumes. This list is presented in descending order of frequency.

1. **Cockpit error.** Many headaches can be avoided and much wasted time saved by exercising reasonable care when creating, modifying, or deleting volume set structure or accounting structure information. The solution: check your work at each step.

2. **UNKNOWN volumes.** Some problems which cause system or user volumes to come up UNKNOWN after a reboot are the result of hardware problems with the disk drive(s), cable, HPIB/HPFL or PBUS interface, etc. The solution: fix or replace the defective hardware.

   On the other hand, some problems causing UNKNOWN volumes after reboots are due to software. There are a couple of known problems which can cause either system or user volumes to come up UNKNOWN after a reboot. You need help from your HP Support Representative to resolve problems affecting system volumes. You may, however, be able to avoid problems with your user volumes by using this one extra precaution: after you've defined and initialized a new master volume or member volume in a user volume set, be sure to :VSCLOSE the volume set. This will cause the Volume Management information on each volume to be validated, ensuring that your initialization has completed successfully. Once the :VSCLOSE has completed successfully, you can remount your volume set with a :VSOPEN command.

3. **System and account managers can't use the ;HOMEVS= or ;ONVS= parameters with just UV capability.** The solution: give SM and AM users who require this ability CV capability.

4. **The :BUILD and :FILE commands sometimes don't accept volume and volume class restrictions in the ;DEV= parameter.** This is a problem which appears on some MPE XL releases and not on others. The solution: for volume restrictions, use the LDEV number. For volume class restrictions, define device classes in SYSGEN that coincide with the desired volume classes. For example, if a particular volume belongs to the volume classes DISC and MYCLASS, add the device class MYCLASS to this volume's LDEV in SYSGEN (DISC will already be there). The reason for this is that for disk devices, device classes are interpreted as volume classes. When you specify ;DEV=MYCLASS, the device class MYCLASS will be used to search for volumes with the volume class MYCLASS.

5. **Group entries appear in user volume set directories when the groups are not homed to that volume set.** This occurs when someone has changed the home volume set of a group from one volume set to another; e.g. from one user volume set to another, or from a user volume set to the system volume set, without purging the group entry from its old home volume set. As long as the group entry in the system directory points to the correct new home volume set, having the old group entry around doesn't cause any problems; it may just cause some confusion. The solution: keep your accounting structure clean. (Check your work.)

6. **Wasted disk space in user volume sets.** This can occur when groups or accounts which were homed to user volume sets have been purged from the system directory, but their "twin" entries in the user volume set directories have not been purged, and so their associated disk space has not been released. The solution: be sure to do a :PURGEGROUP;ONVS= and :PURGEACCT;ONVS= as appropriate when deleting groups or accounts from the system.

SUMMARY

Volume Management on MPE XL, especially with its ability to create and manage user volume sets, is a powerful and effective tool to manage disk space on HP3000 900 Series systems. Admittedly, preparing to use user volume sets requires some education, and managing volume sets and accounting structures in a user volume environment does add more operational overhead. But the benefits provided by the use of user volumes sets: improved disk space allocation and management, increased data privacy and security, and reduced downtime in the event of a disk failure, more than outweigh the additional effort. The first time you have the need to INSTALL your 10 (or more) disk system, and realize that you'll only have to restore the files on 3 drives instead of all 10 thus saving hours of downtime, you'll congratulate yourself on your decision to make the move to user volumes.

Bottlenecks: How to Recognize and Resolve Them

Ralph T. Kotoski
Hewlett-Packard Company
24 Inverness Place East
Englewood, Colorado 80112

Several terms should be defined that are important to keep in mind when addressing any performance issues. The first term is "It Depends...". Every utilization factor, every analysis, and every recommendation for resolution will be dependent upon some set of variables and assumptions. These variables and assumptions must be defined as tightly as possible before final analysis and recommendations can be reached. Whenever someone asks if changing some parameter within the environment will have a positive or negative impact upon performance, the response usually begins with "It depends...". Understanding the variables and assumptions before going any further is imperative. The reason for this is any prediction of what is the current bottleneck or what the result may be from an alteration to the environment is absolutely dependent upon a clear understanding of the variables and assumptions in that environment.

The second term to keep in mind when analyzing bottlenecks is that "There is always a tradeoff." No matter what is done to relieve a bottleneck and improve performance, some sort of tradeoff will always have to be made. Tradeoffs come in all shapes and sizes. They can be in the form of money, system management, programmer productivity, response time or throughput for less important activities, etc. Never be fooled into thinking that some sort of enhancement can be made to address a bottleneck without making a tradeoff. If the tradeoff is not obvious, look harder for it. It may or may not be significant, but it will be there. Business decisions will be made based upon the ability to evaluate the bottleneck, its resolution, and the tradeoff to be made.

The term bottleneck has been mentioned several times already. What is a bottleneck? Simply put, a bottleneck is the maximum capacity that can be provided by a single resource. In a computer system, there are many different types of resources, and consequently, just as many different bottlenecks. The most interesting bottlenecks are usually those that keep the performance (how ever we chose to measure that) of a system from being greater. We want to identify the resource that has reached a high enough level of utilization or is saturated such that excessive delays are present in requesting and receiving service from that resource. That is to say, if the load were to be increased to demand more than the capacity of that resource, no additional throughput is possible. In fact, the response time for that resource continues to degrade as queues and waiting times for that resource increase. That resource, then, is the bottleneck to increased performance. Note that when the bottleneck is first reached, it prevents greater throughput. As demand increases, in the form of numbers of users or increased transaction requests, the bottleneck usually causes response times to increase. This is due to the queueing for the bottlenecked resource. The workload has to wait in line to be serviced.

All bottlenecks are not bad. In fact, since every resource on a system has a bottleneck, they can not all be bad. The bad bottlenecks, the ones we are usually interested in, are those impeding greater performance.

Be aware that the bottleneck to increased performance can change. It depends upon the type of load placed upon the system at any given point in time and how intense it is. It is extremely rare to find an environment that is homogeneous 24 hours a day, 7 days a week. Keep this point in mind for later discussion.

Some typical bottlenecks are the CPU, disk throughput, main memory, and operating system resources, sometimes referred to as Locks and Latches.

As a side note, remember that not all the resources and bottlenecks actually reside within the computer system itself. For example, data communications lines are resources and can be bottlenecks to increased throughput. Data entry operators can be viewed in much the same way. The speed at which they can enter data, the number of coffee breaks taken and the number of phone interruptions all contribute to the capacity with which they are able to enter data.

Another thing to keep in mind is that "There is always another bottleneck." Success, believe it or not, is reaching the next bottleneck. Why are these two statements true? First, there is always another bottleneck because computer systems are composed of resources that are finite. There is no such thing as an infinite resource within a computer system. Second, reaching the next bottleneck means successfully overcoming the previous bottleneck. The bottom line here is that more work is being done with the system than was possible before overcoming the previous bottleneck. That is the success...doing more than was possible prior to relieving the previous bottleneck.

Now that the definition of a bottleneck has been explored, it is time to discuss identification of bottlenecks. As mentioned previously, the bottlenecks we are most interested in are those that prevent greater throughput or are impacting response time. When looking for bottlenecks, it is extremely important to look in as many different ways and for as long a period of time as possible. There are several reasons for this. First, consistency is a key factor. If one data point is used to determine where a bottleneck exists, the likelihood of identifying the wrong bottleneck is significant. Remember, there is virtually no such thing as a homogeneous load on a system. The system manager must be assured that the times the during the analysis were in fact typical and most important to the business operation. No unusual conditions, such as lighter than average load or extra processing, must exist. Unusual conditions can, and usually do, lead to the identification of the wrong bottleneck. Consistency over time is a must when attempting to identify bottlenecks.

There is typically not a single indicator of a bottleneck. Some indicators can, and are, shared by several bottlenecks. To be assured of the proper identification of a bottleneck, as many different indicators as possible must be explored. Leave no stone unturned. Remember, a decision regarding large sums of money and people time will be made based upon the correct identification of the bottleneck. If the wrong bottleneck is identified, corrective action may have minimal effect, or worse yet, may make things worse. When in doubt, continue to explore, or ask for assistance from someone who specializes in performance analysis and tuning.

The tools available to the average system manager are varied. One of the most used tools is the telephone. Generally, users will call the system manager when performance degrades. Many times, system managers overlook the importance of the data they receive from their user community. Listening skills are of paramount importance. Sometimes, users may complain and not be able to provide any clues as to where the problem resides. However, there are times when they may be able to point the system manager in the direction where the problem lies.

Talking to users is extremely important to the process of identifying bottlenecks. Understanding what they are trying to accomplish, how they go about doing it, and when problems arise helps the system manager more clearly understand the workload on the system. Remember, no load is homogeneous. A system manager can not afford to make assumptions about the workload on the system. Ultimately, business decisions are made based upon the knowledge of the system manager.

Users may be able to tell the system manager exactly what activities by which users are causing problems. For example, whenever the Accounting Department executes a certain report, the entire system slows down dramatically. How slow is slow? That leads us to the next tool every system manager should have.

The stopwatch, or a watch with a second hand, is an invaluable tool. Users do not usually measure response time quantitatively. They generally have a perception of what response time is. This perception can be influenced by many factors including time of day, pressure to perform, whether or not a customer is on the phone, and whether or not their pay is based upon how much work they are able to process through the system. This is not to discount the users' perception of response time. Just be aware that their perceptions may not be close to reality. As an example, I have talked with users who have complained of response times lasting several minutes. When timed, response time turned out to be between 20 and 30 seconds. However, these users were on the phone with customers, consequently, 20 seconds seemed like several minutes.

This is why it is important to quantitatively measure actual response time. This gives the system manager a baseline from which to work. It gives a yardstick with which to measure success when addressing bottlenecks. Finally, it allows the system manager to gauge how close to reality response time expectations are. If the same measured transactions were executed on an uncontended system, what would response time be? What is the difference between contended and uncontended response time? Look for ways to quantitatively define workloads and performance.

When measuring response time, be sure to note all activity on the system when the measurements were made. This will give an indication of the types of activities on the system and which of them may be contributing to any problems that exist.

Once information from the user community has been gathered, along with measurements of actual response times, it is time to look at the kinds of things the system itself will show indicating bottlenecks. There are several things to look for without having to use a performance analysis tool, though, these tools certainly make the job of analyzing bottlenecks much easier.

The first thing to do is to walk into the computer room and watch the system. There are indicators of activities that can point the system manager in the right direction. First, look at the CPU activity indicator. Different systems have different types of indicators.

In the MPE V/E family of systems, there are basically two types of indicators. On the Series 6X and 70, the Current Instruction Register (CIR) is displayed by 16 LEDs on the front panel. When the CPU is idle or paused waiting on disk IO, three LEDs will be turned on solid. The third and fourth LEDs from the left, and the fifth LED from the right will be turned on. This represents the PAUSE instruction. It means the CPU is idle or paused. If a system appears to be bottlenecked and these three LEDs are lit, then it is a good bet that the bottleneck is not the CPU. When the CPU is busy, this register of LEDs will be flashing in many different configurations. If the CPU is extremely busy, the display will look blurred. This may be an indication of a CPU bottleneck, though it is not a guarantee.

It is not as easy to look for a CPU bottleneck with the Series 4X and 5X systems. They have a system busy indicator LED on their front panels. However, this indicator is lit whenever the CPU is busy or there is some kind of disk IO transfer activity on the system. Consequently, this indicator cannot be used to definitively determine a bottleneck. Of course, if it is consistently dark, this is probably a good indicator that if a bottleneck exists, it is probably somewhere other than the CPU or disk subsystem.

The PA-RISC (or Series 900) systems have their own indicators to display the activity of the CPU. The larger systems, Series 950, 955, 960, all have a LED display panel on the top of the cabinet housing the processor. The second LED display in this panel is used (when the system is up) to give a relative percentage of CPU busy. It alternates between displaying 'F' and a character between '0' and 'A'. The alternation cycle length depends upon whether or not the BXT2 patch has been installed, or if the operating system version is 2.1 or later. On operating systems prior to 2.1 without the BXT2 patch, the cycle is about 2 seconds in length. With the 2.1 version of MPE XL or the BXT2 patch, this cycle is shortened to about 300 milliseconds. The display between '0' and 'A' is a percentage of CPU busy over the last cycle. For example, if the display flashes between '0' and 'F' consistently, it means the CPU is idle or paused. On the other hand, if it flashes between '9' and 'F', it means the CPU was about 90 percent busy during the last interval. If the display flashes between 'A' and 'F' consistently, it means the CPU is 100 percent busy and could be an indicator of a CPU bottleneck.

The smaller PA-RISC systems, Series 925 and 935, have a row of 5 LEDs as an indicator. Each LED represents 20 percent busy. The indicator will alternately flash (at the same interval as discussed above) between all on and the number representing CPU busy. If the LEDs are on solid, this represents 100 percent busy.

After watching the CPU indicator(s), it is time to turn attention towards the disk subsystem. All current HP disk drives have an indicator on the front which show whether disk IO is in progress. If these indicators are flashing on and off furiously, this can be an indicator of high disk IO rates and could indicate a disk or memory bottleneck. You will also want to see if the indicators for all the disk drives are flashing on and off with the same frequency and intensity. If only one or two of the drives show any activity during a busy period, this probably means IO balancing is in order. The goal is to spread disk IO as evenly as possible among all the drives. Using just one or two drives can create a disk bottleneck.

On older disk drives, you can feel the drives for vibration as the heads seek back and forth. Frequent and violent head movement is another indicator that a disk bottleneck may exist. Sometimes, just watching the drives is enough. Are they doing a dance across the floor?

Other things to look at in the computer room are data communications lines, MUXes, modems, and tape drive activity. Is a job or session reading from or writing to tape? If so, how fast is the tape moving? This can indicate how busy the system is. Some data communications equipment have indicators that show traffic over the line(s). Look to see how busy these lines appear to be.

Note any activity in the computer room that can show some evidence of how busy the system is. Maybe printer activity is an indicator in your shop.

While physical observation is a key in the analysis of system activity, other avenues are also available to help the system manager in the identification of bottlenecks. Several MPE commands can be of assistance. The first is the :SHOWQ command. This command basically displays processes on the system that are using the CPU, short waited, or long waited. All processes that are long waited, usually waiting on a terminal read, paused via the PAUSE intrinsic or waiting for a :REPLY at the console, are shown along the left hand side of the display. The processes most interesting are those along the right hand side of the display. They are generally using the CPU or waiting to use the CPU. The important thing to notice here is the length of this list. Look at how many of the processes that are critical to the business operation are in this list. If a batch job, for example, that is not important, is in this list, do not count it. On MPE V/E systems, if this list of important processes is longer than 5 to 7, this can be a good indicator of a CPU bottleneck. A threshold for PA-RISC systems is more difficult to identify. It depends upon how CPU intensive each of the processes are and may require further analysis with an additional tool like GLANCE/XL. Do not be surprised to see this list 10 to 15 deep and still have acceptable response time.

Another time honored MPE command is the :SHOWJOB command. After gaining experience with a system, this command can give you the feel of how much work is being done on the system by knowing how many, and which, users are on the system. Please note, this is not by any means a true measure of work. However, you may be able to gauge relative loads with this command. Be sure to verify over time that any assumptions made using the :SHOWJOB command remain valid.

On MPE V/E systems, the :SHOWCACHE command is very helpful in analyzing disk IO activity. The number of cache domains assigned for each drive can give an estimate of IO balance. Note that this is for a relatively short period of time and should be monitored regularly. The percentage of IOs eliminated is an excellent indicator of how well Disc Caching is helping. Ranges of 35 to 70 percent eliminated are typical. Look at how much memory is being used for cache domains. If the average amount of memory allocated per drive is less than 1 Megabyte, additional memory may be helpful. Of course, on a system with a large number of drives, this rule of thumb may not always hold true. Disc Caching should be turned off and then back on periodically (maybe daily) to reset the internal counters. When the counters overflow, a negative number of IOs eliminated will begin to be reported. This does not mean that more IOs are being generated than without Disc Caching. It simply means that some of the counters overflowed and need to be reset.

DBUTIL can be used to help discover locking issues within a database. The SHOW db LOCKS command within DBUTIL can give an indication of whether or not there is a locking problem. The important thing to look for is long lines of processes waiting on one or more other processes to release their locks. If this is a consistent problem, database locking strategies should be reevaluated and implemented differently. This can be one of the most pervasive bottlenecks, and yet, be one of the easiest to address.

Data Base Control Block (DBCB) contention problems are much more difficult to ascertain. It is virtually impossible without the aid of a tool like OPT, GLANCE, or a similar third party tool. Processes that experience a DBCB type of bottleneck will be displayed as being IMPEDEd frequently. IMPEDE is a fairly generic wait state within the operating system and is used for many other things. However, if processes are very database intensive and spend a significant amount of time IMPEDEd, it likely that a DBCB contention problem exists or a locking problem exists.

To this point, main memory as a bottlenecked resource has not been discussed in detail. A memory shortage is difficult to identify without a performance analysis tool. With such a tool, on MPE V/E systems, look for a large number of disk IOs performed by the Memory Manager (or Segment Manager). In most cases, an IO rate of greater than 5 per second is a strong indicator of a shortage of memory. Some of the performance tools will also indicate the amount of CPU resources used by the Memory Manager. A good threshold here is, again, elusive and any numbers reported should be correlated with Memory Manager IO. Generally, if more than 5 to 10 percent of the CPU is spent managing memory, a memory shortage may be indicated.

If identifying a memory shortage on a MPE V/E system is difficult, it is even more elusive on the PA-RISC systems. At the time of this writing, Hewlett-Packard recommends configuring between 0.5 and 1.0 Megabytes of memory for each active user on the system. Experience has shown that strictly Compatibility Mode (CM) applications require closer to 1.0Mb per active user. Since all IO in MPE XL is performed by the Memory Manager, thresholds are difficult to define. However, due to the advanced design of PA-RISC, any consistently high disk IO rate may indicate more memory would be useful. PA-RISC is designed to eliminate between 95 and 99 percent of all physical disk IO.

In either case, MPE V/E or MPE XL, there is a simpler method for determining whether a memory bottleneck exists. Most vendors will allow you to try adding more memory to see if performance improves. They will usually do this with the agreement that the memory will be purchased if it does help. That means there is generally little risk for a system manager in terms of spending money for additional hardware before it is known whether or not the hardware will help.

As we can see, there are a number of different ways to go about identifying bottlenecks. The intent here is to only discuss methodology outside of specific performance analysis tools. However, in a dynamic environment, better bottleneck analysis, and consequently, better business decisions can be made with the assistance of a performance analysis tool. These tools can usually pay for themselves in a relatively short period of time and are a wise investment.

Once the bottleneck(s) have been identified and verified, the task becomes one of formulating methods to address the bottleneck(s). Generally, this can be done in one of two ways. First, is to increase the amount of the resource that is bottlenecked. Upgrading the processor in the case of a CPU bottleneck is an example of increasing the amount of CPU resource. The tradeoff in these cases is almost always money. The second method is to reduce the demand for the bottlenecked resource via load management.

Increasing the amount of a resource, such as adding more channels or disk drives, upgrading the CPU, or adding memory, is generally a relatively easy process. The significant tradeoff, as mentioned, is money. The company must pay a vendor for the addition of a particular resource. When upgrading the CPU, another tradeoff can be in the time of the staff to plan and administer the upgrade. This can include training, operating system updates, and migration activities. Keep in mind this method of addressing a bottlenecked resource is widening the bottleneck, not eliminating it. With the bottleneck relieved, more throughput and better response times should be possible.

Load management techniques are usually the most underutilized methods of addressing bottlenecked resources. Yet, many times, the tradeoffs are quite cost effective. Load management attempts to spread workload demands more evenly throughout a processing period. The goal is to reduce the amount of demand for bottlenecked resources during peak or critical processing periods by scheduling certain activities to execute during low utilization periods.

If load management is not practiced, it is virtually impossible to predict whether or not a system will have sufficient resources or capacity to handle critical, peak processing periods. The important point to remember is to plan for, prioritize, and schedule as many activities on the system as possible. In virtually all environments, patterns of work can be identified. Even so called ad-hoc reporting requests can sometimes be planned for and scheduled.

Begin by identifying daily activities. Look at and prioritize all processing that is necessary on a daily basis. Do the same thing for weekly and monthly activities. Patterns do emerge among the workloads. Even last minute requests can, and frequently do, exhibit patterns.

Once the patterns have been identified, the workloads can be prioritized. It is highly recommended that the prioritization process be accomplished with as much input as possible from management within the company. It not only serves to create a common ground from which all concerned can work from, but also avoids any charges of favoritism or bias. Load management is most effective when everyone concerned agrees with and supports it.

Once the prioritization process has been completed, schedule as many activities as possible. It is important to move as much of the lower priority workloads into periods of time when system utilization is low. These are typically lunch times, evenings, and weekends. For example, reporting done every Monday morning may be able to be executed over the prior weekend.

Batch reporting is usually an activity that should be closely scrutinized. Many times, significant gains can be made by scheduling batch work during low utilization times. The question to be asked is when do the reports actually need to be completed?

On-line, interactive users sometimes perform activities that last for extended periods of time and demand large amounts of resources. These activities should be evaluated and, if appropriate, be moved into batch jobs and scheduled.

The process of logging onto the system and starting an application can be intensive. It is important to minimize the number of logons while maintaining proper security. Automatic logons via the :STARTSESS command can be initiated prior to peak processing times. If security is an issue, design it into the application. This way, an unattended terminal is not a window into the system unless the appropriate passwords are supplied. The automatic logon process can significantly reduce demand for bottlenecked resources during busy times.

What are the tradeoffs involved with load management? They will depend upon each environment, but there are some common tradeoffs. The most significant is that it takes effort to manage the workloads on a system. Scheduling and administering load management is an ongoing effort. The benefit is off-loading demand during peak processing periods and a more complete, optimal utilization of system resources.

Another tradeoff associated with load management is that some business activities, such as data collection processes, may need to be altered in terms of the timing of when they occur. Change is not usually reacted to positively. In these cases, the benefits of more timely reporting, better response times for on-line users should be espoused.

Many other methods exist to address certain bottlenecks. This paper will discuss a few of them that can be utilized on either MPE V/E or MPE XL systems. Many times, these methods are overlooked.

One method, frequently ignored, is regular database maintenance. A poorly maintained database can cause a disk bottleneck. In a dynamic environment, databases that are modified frequently should be DBUNLOADed and DBLOADed regularly. This accomplishes several things. First, the data will once again be physically oriented more closely to the logical design (for example, by Sales Order Number). Entries on the same primary chain will be placed in the same block(s). The number of disk IOs can be reduced due to TurboIMAGE buffering, Disc Caching (MPE V/E) and the Storage Manager (MPE XL) being able to more optimally anticipate data needs and eliminate unnecessary disk IO.

Secondly, the DBUNLOAD/DBLOAD process will reset chains within the database. In Detail sets, the Delete Chain usually spans a multitude of blocks, causing TurboIMAGE to place data in blocks that may not be logically optimal. Master sets benefit from DBUNLOADing/DBLOADing also. Keys are rehashed, and even if no resizing is performed, better block placement can occur.

Reducing synonym chains is extremely important and advantageous. Many tools exist to display the state of synonym chains in a database. Use them to identify when a dataset may need to be resized. Generally, a Master set that is approaching 60 to 70 percent capacity will benefit from a resizing effort. Disk space is very cheap now, and should not be a hindrance to making Master sets large enough to eliminate most synonyms. By the way, that is one of the tradeoffs, additional disk space.

If a database is too large to DBUNLOAD/DBLOAD in a timely fashion, then break the process down. Unload and load one or two datasets at a time. Many tools exist that allow single dataset unloading and loading. Several of them will unload and load an entire database faster than DBUNLOAD and DBLOAD.

Do not ignore database maintenance activities. Over time, in a dynamic environment, performance of an application can, and usually does, degrade without proper and regular maintenance activities.

AUTODEFER is another seldom used method of improving batch, database modification throughput. AUTODEFER basically allows TurboIMAGE to execute modifications as fast as possible. The tradeoff (in this case, risk) is that a system interruption during AUTODEFER processing will leave the database in an inconsistent state. Consequently, prior to enabling AUTODEFER for a database, that database must be DBSTOREd (another tradeoff). The benefit is usually very significant. Gains of 2 to 20 times non-AUTODEFER performance are quite usual (actual gains depend upon application and database design, database state, and any contention). NEVER use AUTODEFER for interactive processing since there is no way of recovering transactions that may be lost in the case of a system interruption.

A final tuning recommendation is to keep the amount of historical data stored to a minimum. Large amounts of historical data can cause inefficiencies. If historical data must be stored on-line, separate it out into one or more databases. For example, keep three months of data in the production database. Archive the next six to nine months in a recent history database. Any older data can be further archived to an ancient history database. In most cases, 90 to 95 percent of all transactions occur against current data. Archival of older data significantly improves access time for most transactions. The tradeoff is, of course, the time and effort it takes to administer the archival procedure. Generally, the gain in performance for current activities is significant enough to justify the additional effort of archiving.

Bottleneck detection and resolution can be tricky and is sometimes referred to as an art. The primary point to remember is that identifying the correct bottleneck is of the utmost importance. In order to do this effectively, information regarding business environment, user activity, and system activity must be gathered from every available source. Leave no stone unturned during the investigative phase. Once the primary bottleneck has been identified, verify its existence in as many different ways as possible. Make sure the correct bottleneck has been identified. A business decision regarding money and people time is going to be based upon this process.

Once the bottleneck has been identified and verified, begin the process of formulating methods to widen it. Evaluate all tradeoffs that will have to be made in order to address the bottleneck. The quality of the business decision made will depend upon the effectiveness of this process.

Finally, implement the solutions. Evaluate the success of addressing the bottleneck. Begin the process again. Start looking for the next bottleneck...remember, success is reaching the next bottleneck.

Analysis-Design, CASE, and OOP
A "State of the Art" Assessment

Robert R. Mattson
9545 Delphi Road S.W.
Olympia, WA 98502
(206) 736-2831
(206) 352-5038

Abstract

Each day brings new hardware and software "answers" to solve all our problems! This talk/paper focuses on the state of the art of analysis, design and programming in relation to Structured-System Engineering, CASE and OOP (Object-Oriented Programming). It will look at the fundamental problems we face in developing and maintaining quality systems and what role these two "answers" might play. The question that will be addressed is... whether these two solutions should be considered magic, myths or maybes?

Preface

Analysis-Design, CASE and OOP...what do these words mean? How much attention should the system professional give these terms today? Do they represent significant issues that the already overloaded information system professional should be addressing? How do they relate to the real problems we face with achieving quality systems? How does one cut through all the hype that surrounds them? After many years of experience in using and trying to use the tools and techniques, I hope to present some helpful and interesting information and insights. Maybe there are even a few "answers!" It is hoped that this information and these insights will be of help for others facing similar questions.

Our Challenges and "Answers"

There is an underlying fundamental challenge here. The information systems business is overwhelming these days. This feeling is "in general" and "in particular." In general, this problem translates into a blur of all the new technologies, techniques and challenges which seem to be increasing exponentially while our human absorbing/learning capacity is increasing fractionally at best. This means we must be selective. It is impossible to focus on it all!

In particular, the last ten years has seen a multitude of "answers" to the challenges we face. A little brainstorming could come up with a list a mile long. Here are just a few.

Structured Analysis	RISC
Unix	Structured Design
Structured Programming	Data Modeling
Work Stations	C, Ada, Prolog
Transact, Smalltalk, etc	SQL
Expert Systems	Relational Databases
Fourth Generation Languages	Information Modeling
End User Computing	PC's
Spreadsheets	Desktop Publishing
SAA	CASE
And More and More	

How many more can you come up with to add to this list? It seems that we are being bombarded by <u>easy</u> "answers." Now, we can add another one to the list. This is OOP (Object-Oriented Programming) and its associated object-oriented analysis (OOA) and design (OOD).

There are entire books, multi-day conferences and classes on CASE and OOP. It may seem a little audacious to attempt it, but my goal here is to condense this mass of current information, outline the critical issues and assess their significance.

Before directly addressing CASE and OOP let's explore the general area of system analysis-design. We need to do this because many of the "state of the art" conclusions about system analysis-design directly affect any assessment of CASE and OOP.

System Analysis-Design

What is system analysis-design? How do we recognize an excellent analysis-design as opposed to a mediocre one? What is the best way to assure an excellent system design today? Those are very hard questions on which to put concrete answers. Nevertheless, let's explore some key ideas.

The purpose of doing analysis-design and programming is a successful system!

This seems obvious doesn't it? Surely, this most fundamental axiom is agreed to by all professionals.

This "obvious" point needs to be emphasized here because it would seem that it gets lost many times in all the hype related to analysis-design methodologies, CASE and OOP. The tools and techniques seem to take on a life and purpose of their own. We need to have this goal clearly in mind in order to interpret many of the claims from the "gurus" and the vendors of methods and products.

A successful system is one that meets the customer's desires for usefulness, initial cost, time of completion and total system life cycle cost.

This is a single sentence definition of what constitutes a successful system. Attempting this is fraught with potential objections. Nevertheless, it is important because a definition of what defines success is another building block idea needed when assessing other tools and techniques.

This is a "fundamentalists" definition. It specifies no required "method" or "tool." The customer does not care how the system is developed so long as it meets the success criteria outlined above. Note, also, the emphasis on the importance of cost of the system. This means both initially and over its entire life cycle.

Analysis and design need to be viewed in as integrated activities.

We need to see analysis and design as very closely coupled activities. One can find many examples and ideas within the professional literature and culture which separate these activities. It is incomprehensible how this separation came about. Many of the problems associated with poor designs can be traced to the conceptual separation of these two activities. It is unequivocally the case that analysis and design are inseparable activities if our goal is a quality system design.

The nature of software systems makes it difficult to achieve our definition of success.

Here we need to pause and state something for the record. Achieving successful software systems is not easy! Even achieving a moderately successful system takes a great deal of effort and good degree of skill. In spite of this, we are constantly fighting against a form of mass brain-washing. This is in the form of numerous advertisements for various hardware and software companies and products that imply that we aren't doing things the "easy" way. "Buy this" or "do this" and all those problems one is having with systems will go away. We find ourselves dealing with people, users and management, who's knowledge seems only to be this mass brain-washing. We find ourselves feeling many times like we should be doing things faster, better and cheaper. Is it that all these intelligent and hard working people in this field have just been taking it easy? This is not the case. After years of doing and observing in this field, I believe that we need to acknowledge the intrinsic difficulties of creating successful software systems. We need to do this within the profession at least. We need to do this in order to keep our attitude positive and our self esteem up.

Software systems have many characteristics that are not clearly understood. One such characteristic is the fact that we are dealing with people and their thought processes. The possible ways that people can think about something is large. When one combines multiple thought patterns with the concrete problems in any software system the implications for complexity and difficulties are evident. Another characteristic has to do with change. If an organization and users change then systems must also or they become less successful. Since organizations and users are in constant state of change... we are developing something which must actually be changing itself. This might be called the "moving target" syndrome. These are but two examples of aspects to software systems which we only barely have come to understand or acknowledge. We need to exert far more energy in understanding the many characteristics that are part of the "nature" of software systems.

Maybe by acknowledging and affirming the real and complex challenges we face in developing software systems we will be better armed in evaluating any proposed tool or technique solutions.

Successful system analysis-design is more than "popularized" analysis-design methods address.

or

Successful top quality analysis-design is not simply a function of any method's diagrams and documentation!

We spent a little time laying out some ideas related to systems. We have an agreed upon the criteria for measuring the success of a system. We have acknowledged that we are dealing with something difficult. We have said that there are many complex aspects to this work which are less than fully understood. Is it any wonder that this profession's track record is a mixed one. We try very hard to achieve our customer's system expectations. What do we find? We find that ours and our customer's expectations are very hard to achieve many times. One or more of the criteria is not met, in spite of our best efforts. This less than successful situation and our desire to achieve the best leads to a wish and even need to look for an "answer." In fact, human nature being as it is, we would especially like it if the answer was quick and easy!

In response to our desires the world seems to develop "answers." One answer offered for our desire and needs is that of "software engineering." The concept says we just need to apply "engineering" discipline to the system development process and we won't have these disappointing results. The challenge is that there are many vague definitions as to what "engineering" is in this context.

Notwithstanding the true lack of meaning to the term engineering, the term connotes a formal, standardized, clear, step by step process. Each step of the process will follow specific rules and be replicatable. This concept has a nice feeling about it. It implies a cleanliness and precision that is attractive to people who have "analytical" tendencies.

This enticing concept has given rise to a number of popularized (notice I didn't say popular or commonly used) "methodologies" that have claimed to define "software engineering." The Yourdon/Demarco and the, Warnier/Orr "methodologies" are two of the better known. A great deal of time and cost has been spent in learning and trying to implement these techniques. In fact these techniques have been around for well over ten years now. A recent estimate I heard indicates that less than 5% of the shops use these techniques. Why aren't these techniques more pervasive? Here are some reasons for this situation.

First, we need to focus on what a quality system design is because a design is the primary product that any analysis-design method or tools ultimately should be trying to produce. In studying the current concepts and "methods" related to system analysis-design I am struck by their narrowness of problem domain and consequently solutions. Let me elaborate. Most of the software engineering techniques would seem to indicate that <u>the</u> problem in system design consists only of figuring out what the database design should be and/or what "modules" need to be developed.

They pay only fleeting lip service to other issues. For example, problems such as human interface design and understanding the <u>real</u> system problem seem to at best get passing mention. To often it would appear that these methodological advocates are like little kids with a hammer...to which the whole world looks like a nail.

This is not to say there aren't differences in the quality of database designs. Or that there aren't better ways to design system modules. Yet, are these the most important and toughest challenges normally faced by designers in achieving quality systems? I doubt it. This raises the deeper issue. Namely, what are <u>all</u> the factors one needs to consider when doing a system design. Are they merely the model of the underlying data structures and process flows? My belief is that all the popularized techniques are of some benefit in some problem domains and some situations. But they are much too narrow in their conception of what is required in order to develop a quality software system.

In fact, watch what is happening to most of the popularized techniques. You'll see them trying to change to meet many of the real deficiencies people have found in trying to use them. The fact they are attempting to change is positive. Still, remember that they were and are sold as solutions and answers with <u>no</u> deficiencies. Further as previously noted.. few people continue to use the techniques today who have tried them in the past. Isn't this one of the best indicators of the true value of these "solutions."

Further, I have seen little <u>unquestionable evidence</u> that any of these techniques has delivered on the promises of better, faster <u>and</u> cheaper. I'm sure their <u>promoters</u> might disagree with this assessment. Still, what mainly exists is a great deal of <u>testimonial "evidence"</u> from the people who make their living selling the techniques, themselves or products! The bottom line is that they promise much more than they really deliver.

This leads to an alternative hypothesis. Our goal of achieving successful systems will not in the near future lend itself easily to any simplistic conceptions of the current "structured " or "engineering" methodologies. Rather, the elements that make for successful systems are many. And, they are more complex than anything being addressed in the current "engineering methods" and their "tools." In other words the process will remain a complex, "messy" and artistic process.

<u>Software Engineering Requires "System Blueprints."</u>

A fundamental deficiency brought out by a review of the system analysis-design arena is that we as system developers haven't spent much time defining what constitutes the deliverable of design specifications. Think about what design specifications mean when describing a building or other physical object. The blueprints of architectural or engineering design can be sent across the country to be built in two different towns. The building or object built in two separate locations would be identical in almost every feature. What is the analog in systems? How many people design software-systems in a way that would meet this test? Clearly, if we are to use the word engineering, don't we need to develop a more comprehensive and universal set of design specification standards? This set of system "blueprints" would be a precursor to developing any software engineering methods and the tools (i.e. CASE) to produce them.

Top quality system designs come from top quality system designers!

and

System designers/architects are not just "experienced" programmers!

We've been discussing the methods of analysis-design. Let's turn our attention to the issue of people. Think about the designers of physical things like houses or cars. Architects are not just experienced carpenters. And automotive engineers are not just experienced mechanics. They are trained in multiple disciplines all of which are required to accomplish the design of a physical objects or buildings. It is clear that systems are a much more difficult thing to design than most physical object. Given this fact, why do we seem to believe that the only required training for system designers is at best a few programming classes and a few years as a programmer?

Given this, prevalent belief about required qualifications, is it any wonder that the world finds our designs are less than successful many times? We will continue to have a great deal of difficulty in this area until we recognize the difference between the skills, capabilities and training required of our programmers (carpenters, mechanics) and our system designers (architects, engineers). In fact, I believe, we need to develop a concept called the "system architect" to replace the old meaningless term system analyst. And in fact I have seen this term used recently by Mitch Kapor, founder of Lotus Development and by Bill Gates of Microsoft.

Further the system architect must be conceived of as both the analyst and the designer. We need to stop perpetuating the myth that excellent designs can be done by designers who are passed some "model" of the system on pieces of paper.

Additionally, in my experience, the number of excellent system designers-architects is some very small fraction of all system people. We need to figure out how to identify these individuals. This is a tough task but essential if we are to meet our goals. People are still the key. Given a choice between an mediocre system architect with "all" the methodologies and an excellent one with "no" methodologies, choose the latter!

Conclusions about system analysis-design.

What conclusions can we draw from our discussions about system analysis-design. First, we have a great deal more to learn about what software systems "are" and what software engineering them might entail. Second, the "solutions" that have been popularized in the last ten years have only limited utility. Third, quality system analysis-design is very much a people oriented issue. Lastly, we will need to keep our minds open to the complexity of what we do and therefore the complexity of its solutions.

Case - Computer Assisted Software Engineering

We've reviewed some key ideas about system analysis-design. This gives us the basis to now evaluate CASE.

What is CASE - Computer Assisted Software Engineering?

The name CASE would appear to be a self defining acronym. In other words, if a computer tool assists in software engineering... then it must be a CASE tool. But as we've seen, the term software engineering itself has many meanings... mostly revolving around a few popularized methodologies. And in fact, there are <u>many</u> different products out there that all claim to be CASE tools. Significantly, many differences in scope, function and methods exist. This fact clearly indicates that the definition of what constitutes CASE is pretty <u>broad and loose.</u>

The fact that no definitive definition for CASE exists should put us on guard. Just because someone says a product is a CASE tool <u>means nothing in particular</u>! Armed with such an understanding we can now address the concept of "CASE" in a more realistic light.

CASE Tool Features

Let's list some of the issues and functions for which the CASE tools try to provide "computer assistance." Note that the great majority of CASE tools address just a subset of this list. Here then is a composite list of areas addressed by CASE tools.

1) Computer Assisted Diagramming Tool(s) - for one or more of the following analysis methods:
   - Entity-Relationship Diagramming
   - Data Modeling
   - Data Flow Diagramming - Yourdon/Demarco,
     - Gane/Sarson, etc
   - Real Time Process Diagramming
   - State Transition Diagramming
   - Warnier-Orr Diagramming
   - Flow Charting

2) Assisted "Method" Verification

3) Database Analysis-Design Support

4) Computer Assisted Diagramming tools for Design "methods".

5) Data Dictionary - Repository for information about the system.

6) Report and Screen Format Layout

7) Module Code Generator

8) Reverse Engineering of Existing Systems

9) System User Documentation

Warning! Once again it is important to note that no product I've used or reviewed provides the functionality in all of these or even most of these areas <u>in the way you would really like</u>. CASE vendors appear to be in large part marketers. They sling the buzz words associated with our real needs, desires and problems... with easy abandon. Unfortunately, the implied promises appear much faster and easier than delivered real results.

The allure of CASE.

The major reason for the visibility of CASE extends back to our discussion of analysis, design and programming "solutions." We are looking for an answer, maybe even an "easy" answer to our very real challenges. CASE is alluring because of what it alludes to or promises... assurances of correctness from the 'engineering' and ease from the 'computer aided.'

CASE tools are build on a foundation open to questions.

All the CASE tools I've reviewed rest squarely on only a few analysis-design "methods" which have been advertised as "solutions" in the last few years. We discussed these previously when looking at system design. These methods are the "software engineering" for which CASE tools are attempting to provide computer assistance. Here is THE question... if the underlying "methods" are really only a very small part of what will someday be called software engineering then can these tools be anything but partial solutions? The bottom line is that the CASE tools I've seen are too simplistic in their conception of what constitutes system analysis-design. Consequently, they may be useful for dealing with <u>certain aspects</u> of <u>certain problems</u> in <u>certain situations</u>.

Additionally, the CASE marketing\concept consciously or unconsciously reinforces the belief that software engineering can be a "one, two, three" process which can be automated. We are a long ways from the day that any methodological model or tool... assisted, automated or not, will result in universal push button quality design solutions.

Who will benefit from CASE tools.

Not withstanding the limited uses of the methods that CASE technologies support; those people who are already actively using the "methods" that a particular CASE tool focuses on will stand to benefit most from using the tool. This benefit will come from increasing the efficiency and effectiveness with which they apply the method. CASE tools will not by themselves suddenly make systems people knowledgeable in any of the "methods." In fact, my experience is that CASE will just much more rapidly pinpoint the lack of knowledge and practical skill level in a user.

CASE will benefit those who can <u>afford</u> to utilize one of the "methods." My experience is that these are large shops in business and government. They seem to be able to <u>afford</u> to spend great amounts of time jumping through the methodological hoops. It is not clear that small shops will be able, in the near term, to suffer the added overhead and cost of any of the "methods." This is true whether done manually or supported by CASE tools. Rather these shops should be focusing more on using what Tom Lister calls BASE - Brain Assisted Software Engineering.

Case tools need to handle the entire System Life Cycle (SLC).

This idea was touched upon with other focuses earlier. Here it is important to highlight the limitations of CASE tools because they do not deal with the entire system life cycle. If CASE tools and the methods they support are to be valuable then they need to be an integrated part of the entire system life cycle. Some of the more expensive CASE tools attempt to deal with the entire SLC. One should approach these with a Missourian attitude of believing it when <u>you use it</u>. The bottomline is that most of the CASE tools focus on only a part of the system development phase of the system life cycle. This significantly limits their value.

Summary of CASE

CASE tools imply and promise more than they can deliver. They are built upon a shaky system development methodological foundation. That is not to say that they won't be helpful in doing certain system development tasks, either faster or better. They will not however make a "software engineering" miracle happen any more than a word processor writes a excellent novel. With this knowledge we can sensibly make decisions about evaluating, obtaining and, at this stage, experimenting with CASE technology.

Object-oriented programming.

If the buzz word for the 70's was "structured" and the buzz word for the 80's "relational" or "software engineering" then that for the early 90's is "object-oriented." We'll see "object-oriented" everything... languages, databases, analysis-design methodologies, etc. What is this latest craze all about? Will it be the answer any more than structured, relational or software engineering has been in the past? Let's take a look.

What are the fundamentals of "object-oriented?"

Here are the key concepts underlying the object-oriented model and strategy.

1) "Objects classes" - which resemble the "entity" concept in data modeling. (Example person, part, etc)

2) "Objects" - containing <u>both</u> data and the methods to access and process the data.

3) "Encapsulation" - data processing implementation and data structures of objects are hidden from "outside" world.

4) "Inheritance" - Object classes can be "derived" from other object classes. Inheriting their data attributes and methods.

5) "Polymorphism" - a method defined for a parent object class can be redefined to meet the needs of the child class.

6) "Reusability" - fundamentally a concept along classic lines of the reuse of data structures and methods.

The above are the general principles most often associated with object oriented approach. They are in some ways an evolutionary rather than a revolutionary step. This approach emphasizes and combines strategies used in the past along with new ideas and the languages to support them. So if you are caught by the thought...

we've done things like this or that before... you're partially correct.

<u>The definition of object-oriented is not absolutely defined.</u>

It is important to understand the uncertainties associated with the definition of object-oriented as of this date. As with CASE, there are numerous products and pundits all claiming to have "it" or know what "it" is! This state of affairs sounds very similar to that of CASE and analysis-design methods. This similarity should raise the caution flag when looking at any object-oriented approach.

This definitional variety is very evident when one looks at all the different languages that claim to be object-oriented. There are similarities between them and yet there are significant differences.

Anyone looking at using the object-oriented languages is advised to understand the differences and their implications.

<u>Object-oriented programming will require new methods of analysis-design.</u>

If one tries to use the object-oriented programming languages that exist using the analysis-design mindset of procedural languages... problem rapidly arise. It is generally recognized that new methods of analysis-design will be required to be developed, learned and applied before we will see many of the benefits of object-oriented programming itself.

Given this requirement, my review of the latest thinking in the area of object oriented analysis-design does not indicate a quick solution to the problem. There are numerous people trying to come up with how you do this object-oriented analysis-design. And you will see a spate of books on the subject within the next year. In spite of this, you will find that the techniques are less than the solutions we will need. This state of affairs will diminish this approaches usefulness. Further, it will also limit our testing and assessment of the validity of the object-oriented model in the near future.

THe Object-oriented environment will require higher level of competence and skills.

The fact that a higher level of competence and skill will be required for the object oriented enviornment (analysis-design, programming, database, etc) contradicts some of what you may be hearing. Many proponents refer to the "fact" that object-oriented programming will be much more productive than traditional means. Productive however may not mean easier. It certainly is the case that the design of the objects and their methods is more critical in the object-oriented arena than in more traditional ones. It would also appear that the skills required to design and develop systems within this environment are greater, at least, for some tasks.

The best information available would indicate that as long as six months may be required to train a person to think and program within the object-oriented paradigm. Since we are all experienced with switching to new languages much more rapidly than that.. it is a definite indicator of the difficulties, of possibly a new nature, in implementing this new approach. The true meaning of these learning difficulties has yet to be revealed.

The CASE tools in existence will all claim to support object-oriented analysis-design.

We have already indicated that there is still much uncertainty about what object-oriented programming and analysis-design is exactly. It can be taken as pure marketing hype therefore if you see any CASE tool claiming to be supporting object-oriented in the next year.

No object-oriented language exists currently in the HP3000/MPE environment.

Object-oriented languages will appear for the HP3000, but currently none exist, at least for the MPE operating system. The first language we are likely to see is C++ since this is a language which is translated by a pre-compiler into C.

Some features of the object-oriented model can be approximated in current languages. The sub-program in the COBOL environment allows for achieving some features such as method and data hiding. Pascal allows for one to inherit data structures from another data structure. Message files are a way of passing messages between processes. Yet, languages which directly support the object-oriented concepts would definitely make implementing this approach much easier.

Performance will be a significant issue for some applications.

It is the case that there is a computer performance price to pay for using object-oriented approaches and languages. Whether it is significant will have to do with application design tradeoffs, particular languages and hardware.

The argument that is brought forth to counter this downside is that we will make it up in more reliable, flexible and less expensive systems to build and maintain. This will not be an acceptable tradeoff for all applications.

Conclusions about object-oriented programming.

I have indicated some uncertainty about the nature of the object-oriented approach. This is definitely the state of the art of object-oriented... uncertainty.

Our professional task is to weather the hype in the short term while applying what is truly valuable. The hype and promises will get very fast, thick and wild in the near future. One is forewarned that these are mostly marketing hype and not "bankable" useful techniques or tools.

At the same time, the object-oriented approach will prove to be a useful new model. It will have benefits in certain problem domains. Early indications are that it will not be the answer in every problem domain!

What is needed are some concrete examples, within the problem domains of common HP3000 applications, of successful use of any object-oriented model and its tools. Pioneers, westward OO (Object-Oriented)!

The "State of the art assessment" of the "State of the art."

We've covered a lot of ground in looking at analysis-design, CASE and OOP. I've stated my conclusions in each area and won't repeat those particular ones here. There are however some overall conclusions that I believe can be drawn. First, we, in software systems, are involved with a challenging, multifaceted and moving problem. Second, the "nature" of the problem is far from understood. Third, it may be that software systems have a "nature" that doesn't fit into any known conceptual models, for example "engineering." Fourth, in the foreseeable future, achieving quality systems will be dependent mainly on the brains of the people doing them. And, finally, there are no easy answers! In spite of this fact there are days I'd give a years pay for one!

The Author
Rob Mattson is currently Manager of Information Systems at WIDCO, a large open pit coal mine in the state of Washington. He has been working in the systems field for 15 plus years, the last 12 in the HP3000 environment. His background in addition to many systems related roles includes a degree in Psychology, work as a Certified Public Accountant and as a management consultant/trainer. His current professional areas of interest include general systems theory, project management and exploring the concept of system quality. When not immersed in systems he enjoys his family, sailing, sea kayaking and golf.
A Similar Paper was Presented Originally at Interex Computer Management Symposium, Las Vegas, Nevada March 1990.

COBOL & VIEW Today: Integration Of COBOL 85 And HiLi

By

Rick Hoover

CIV Software
P.O. Box 704
Draper, Utah 84020
(801)-571-5674

Introduction:

I have been working as an applications programmer on HP3000 systems since 1978. During that time I have seen many changes to the system. Most of the changes over the years have been helpful to us programmers. We've seen many enhancements to the MPE operating system. 4GL packages have helped increase (hopefully) programmer productivity (at least that's what the software companies tout!). Faster processors now come in smaller boxes, we access the HP via PC's, etc. The list goes on and on. But...the changes in COBOL and VIEW have not seemed to keep up. Let me give a short historical lesson in VIEW:

In the beginning (pre 1979) HP beget DEL. The user community saw that this was, well, not good, but ok for the price.

HP heard the cries from below and said 'Hummm'. That 'Hummm' begat VIEW. The users saw this new product (free also) and thought 'this is pretty good after all. But...in KSAM files? Look at all the intrinsics! And the compile time for one minor change. WOW!'

Then the gurus at HP saw the writings and said 'Hummm'. This 'Hummm' begat V/3000 and later V/PLUS. The users saw this and said 'Ohhhh look...no more KSAM files! No more re-compiling all forms for one minor change!'. They were happier. But...the users still muttered 'Look at all the intrinsics! They are still with us!'.

And thus came HiLi.

The users, especially new users, could see that this was good. There were only a couple of intrinsics to learn. But...how do they work? Now that COBOL 85 is out, what do I do with these?

This paper will try to show users that are still a bit confused about V/PLUS and HiLi, how HiLi is effectively used in an application. This paper will also explore some of the new commands in COBOL 85 that will be used to ease the programming labor of prior versions of COBOL.

Why Use HiLi?

This is probably the most important question that a user could ask. Let me give you some pluses and minuses about HiLi over V/PLUS intrinsics:

Intrinsics:	There are 29 V/PLUS intrinsics to learn and 11 HiLi intrinsics. Out of the 29 V/PLUS intrinsics, 22 will be used in most applications. Out of the 11 HiLi intrinsics, 8 will probably be used in most applications. However, you may not mix V/PLUS intrinsics and HiLi intrinsics in an application.
Parameters:	V/PLUS uses the V-COMMUNICATIONS-AREA. All information about V/PLUS and how it should work is contained in 1 parameter area. HiLi uses many different parameter areas for the intrinsic calls. Virtually every HiLi intrinsic call passes GLOBALPAK and RETURNPAK. In addition, most of the HiLi intrinsics use at least one more parameter.
Errors:	The program should check the return code after each call. That means many possible error returns from V/PLUS as opposed to HiLi because of the number of intrinsics called. The only field that needs to be checked in HiLi is RETURNSTATUS in the RETURNPAK after the call.
Program Flow:	The program flow for HiLi is very straight forward. After opening the forms file and terminal, the program needs only loop through the HPDSEND - HPDREAD intrinsics. If RETURNSTATUS comes back with a value other than 0 after the HPDREAD, then call the HPDPROMPT error. The following flow chart shows the standard logic construct in a HiLi program:

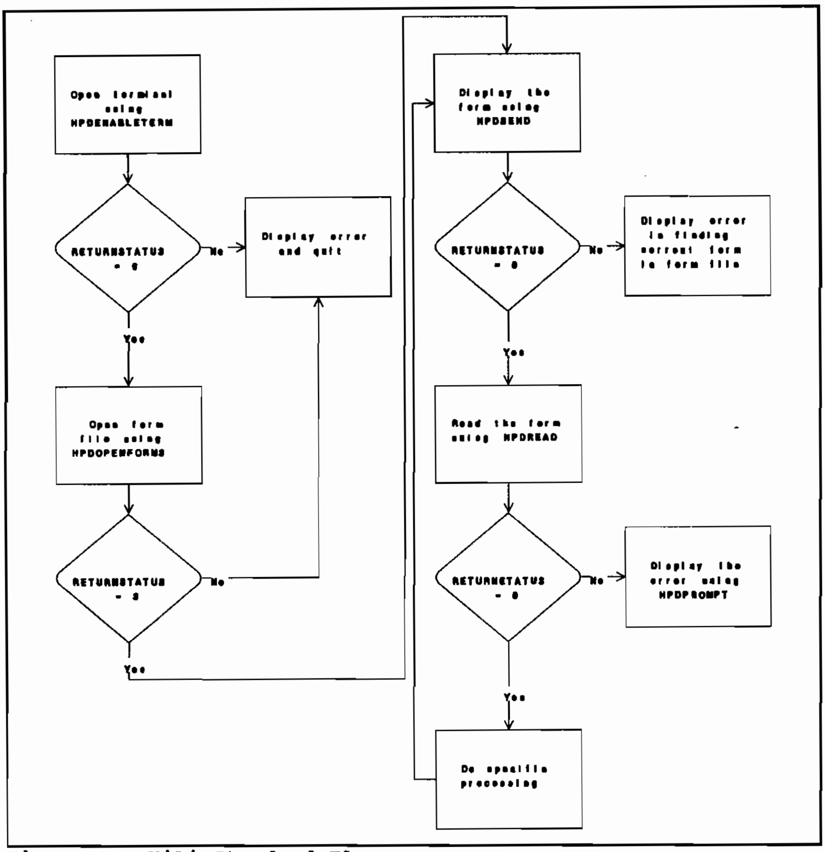

Open terminal using HPDENABLETERM
RETURNSTATUS = 0
No
Yes
Display error and quit
Open form file using HPDOPENFORMS
RETURNSTATUS = 0
No
Yes
Display the form using HPDSEND
RETURNSTATUS = 0
No
Yes
Display error in finding correct form in form file
Read the form using HPDREAD
RETURNSTATUS = 0
No
Yes
Display the error using HPDPROMPT
Do specific processing

Figure 1 - HiLi Standard Flow

There is a standard flow for the actual data entry. The first few steps only occur if you need to open the terminal and forms file. If they have already been opened (from a driver program for example), then those steps are not necessary. Later, I will show how COBOL 85 creates an in-line PERFORM loop that will make this process extremely easy to code and follow.

V/PLUS Control	This is where the designer will have to decide. Because V/PLUS intrinsics allow the programmer total control over how the forms function, using V/PLUS intrinsics may be more helpful. HiLi intrinsics do a lot of the processing for the programmer. In fact, you may come to think of the HiLi intrinsics as macros to V/PLUS intrinsics.
Mind Set	This is probably the toughest point to make. A novice programmer on an HP3000 may be leery of all the V/PLUS intrinsics and find the macro HiLi intrinsics a lot easier to understand. Most programs will send, read and prompt from V/PLUS screens. Many programmers who have used V/PLUS intrinsics for years have found HiLi tough to use. There's a mind set of knowing how to perform a routine. No matter how difficult the procedure, someone will come up and show an easier way. Sure, you may have lost some control over what you have been doing, but it is more straightforward. The most problems of misunderstanding about HiLi intrinsics have come from 'old-timers' used to V/PLUS intrinsics.

Setting Up The Parameters.

The parameters used in HiLi are pretty straight forward. The first 2 parameters needed by HiLi are GLOBALPAK and RETURNPAK.

GLOBALPAK:

```
     01  GLOBALPAK.
         05  EXPECTEDVUF                     PIC  X(08).
         05  CALLPROTOCOL                    PIC S9(08) COMP.
         05  COMAREALENGTH                   PIC S9(08) COMP.
         05  COMAREA                         PIC X(300).
```

There are 4 fields in the GLOBALPAK parameter. These need to be set at the beginning of the main program and then left alone for the remainder of the process. The first field is defined as an 8 byte character array containing the revision level of the intrinsics. There are two ways to set up the EXPECTEDVUF field. The first way is to force the value 'A.00.10' into the field through a move statement as in:

```
MOVE "A.00.10" TO EXPECTEDVUF OF GLOBALPAK.
```

The second way (and the way that I prefer) is to retrieve the current revision level from the system at run time. This insures that 6 months or a year from now, the program will still get the most current information. To install this routine, use the following code:

```
01  VUF-VERSION                     PIC X(14).

CALL "HP32424" USING VUF-VERSION
MOVE VUF-VERSION(8:7) TO EXPECTEDVUF OF GLOBALPAK
```

The infamous HP labs have said there will be functionality later in the revision level. Certain HiLi intrinsics will inspect the revision level and be able to perform routines (whatever that may be) depending on the revision. By using the above code, the program will always be using the proper revision level.

CALLPROTOCOL is simply a four byte integer that tells HiLi which language is being used. To initialize itemcnt this field:

```
MOVE ZEROS TO CALLPROTOCOL OF GLOBALPAK
```

There is another way to initialize this field, but I will wait for a while to explain how.

The next field tells HiLi how large the communication area will be. This field needs to be set to 300.

```
MOVE 300 TO COMAREALENGTH OF GLOBALPAK
```

The final field in GLOBALPAK needs to be set to binary zeros.

```
MOVE LOW-VALUES TO COMAREA OF GLOBALPAK
```

So, that means that in the main program, your code should look something like this:

```
**** BEGIN GLOBALPAK INITIALIZATION

     CALL "HP32424" USING VUF-VERSION
     MOVE VUF-VERSION(8:7) TO EXPECTEDVUF OF GLOBALPAK
     MOVE ZEROS TO CALLPROTOCOL OF GLOBALPAK
     MOVE 300 TO COMAREALENGTH OF GLOBALPAK
     MOVE LOW-VALUES TO COMAREA OF GLOBALPAK
```

A couple of notes about GLOBALPAK:

- The application should initialize GLOBALPAK before the first HiLi intrinsic call and remain untouched for the remainder of the application.

- The COMAREA needs to be initialized to a minimum of 300 bytes. If you are going to use BASIC or any specialized data entry systems, this area will need to be larger. Refer to the Forms Management manual for the correct size.

- The CALLPROTOCOL parameter tells HiLi the language that you will be using. Once this parameter is set, HiLi will expect all calls in that language's format.

- The version can be forced to 'A.00.00'. However, there are certain functions that will not be available to this version. You either force 'A.00.10' or use the above routine to initialize the version. This way, the current functions (discussed later) will be available to you.

RETURNPAK

```
    01  RETURNPAK.
        05  RETURNSTATUS                PIC S9(08) COMP.
        05  SUBLAYERSTATUS              PIC S9(08) COMP.
        05  RETURNMSGLEN                PIC S9(08) COMP.
        05  RETURNMESSAGE               PIC X(256).
        05  LASTITEMTYPE                PIC S9(08) COMP.
        05  LASTITEMNUMBER              PIC S9(08) COMP.
        05  LASTITEMNAME                PIC  X(32).
        05  NUMBERDATAERR               PIC S9(08) COMP.
        05  NUMBERCHGFLD                PIC S9(08) COMP.
```

This parameter is easier to work with because you do not have to make any changes. HiLi uses this parameter to pass back all information about the specific HiLi call. The fields available to you are:

RETURNSTATUS	This field tells the program if the HiLi call was successful or not. If the value of RETURNSTATUS is zero, then the call was successful. If the value is less than zero, then there is a programming or system problem. If the value is greater than zero, then there was an operator or data problem. You should always check this value after a HiLi call.
SUBLAYERSTATUS	This field may be asked for by HP to explain RETURNSTATUS. It is for diagnostic purposes only. You will be able to find any problem by using RETURNSTATUS.
RETURNMSGLEN	This field will contain the length in bytes of the message returned in RETURNMESSAGE.
RETURNMESSAGE	This is a 256 byte field that will contain the error message that was generated from a non-zero error. Remember that the V/PLUS window is only 79 characters long. To use this message: `MOVE RETURNMESSAGE(1:79) OF RETURNPAK TO V-MESSAGE`
LASTITEMTYPE	This field will contain either a 0 or 1. It is used by the HPDREAD intrinsic. 0 shows that the read intrinsic was terminated by either the ENTER key or a function key. 1 shows that the read intrinsic was terminated by a field.
LASTITEMNUM	This field contains the function key number or field name that completed the read. For example, if the value is 0, then the ENTER key was pressed.

LASTITEMNAME	This field contains a 32 byte ASCII name that tells the name of the key or field that completed the read call. For example, this field would contain $ENTER is the ENTER key was pressed, $PFK_1 if the f1 function key was pressed, etc.
NUMBERDATAERR	This field contains the number of data errors found during data initialization, editing or reformatting.
NUMBERCHGFLD	This field contains the number of fields in which the operator entered data.

That takes care of the standard parameters that HiLi expects. We will now go into COBOL 85 and discuss how a driver program is built with some of the new functionality that both HiLi and COBOL 85 allows. First, let me give you a couple of notes about how this little program will function:

- All V/PLUS screens have no editing built in. All editing takes place in the COBOL program. There is nothing inherently wrong with using V/PLUS edits, but they will slow down the operation of the program since they run much slower than a compiled COBOL program will run. Also, there are routines in COBOL 85 that replace the functions of many of the V/PLUS edits.

- The main program is simply a driver program that displays a main menu. The operator selects a specific screen to use and presses the ENTER key.

- The function keys perform this way:
  f1 - Not used
  f2 - not used
  f3 - print the screen
  f4 - not used
  f5 - not used
  f6 - not used
  f7 - return to the main menu (not used in main menu)
  f8 - exit from system from anywhere

- All screens have the current date in the upper left hand corner.

- All maintenance screens use 'A' for add, 'C' for change, 'D' for delete and 'I' for inquiry

- To change or delete existing data, first inquire then change or delete.

The COBOL program code that will be referred to is listed in appendix A. You may first notice there is something missing in SENDPAK and TERMPAK. There is no field name on the third line in SENDPAK and the second line in TERMPAK. COBOL '85 does not need the term FILLER anymore.

Now let's look at the PROCEDURE DIVISION. The first line will initialize the IMAGE parameters. All alphanumeric fields will be set to spaces and all numeric fields will be set to zeros. This will take the place of many MOVE statements. The only problem that occurs is if there are any REDEFINE's in the record layout. REDEFINE's need to be initialized separately. The code will then initialize the various HiLi fields with the forms file name, form name, terminal name message length and default data buffer length.

The first HiLi call is HP32424. This call returns the HiLi version that is now on the system. The version code is in the last 6 positions of the returned version number. This field is necessary to pass to the GLOBALPAK parameter. There are two ways to get this number. The first way is the old standard COBOL way. You simply set up the VERSION parameter as two fields.

```
     01  VERSIONVUF.
         05                          PIC X(07).
         05  VERSION-NUMBER          PIC X(07).
```

The COBOL '85 way to get the parameter is to use a reference move. The reference move allows the program to access a substring inside of a string. The first number indicates the starting position of the substring. The second number states the number of characters to be moved.

```
     MOVE VERSIONVUF(8:7) TO EXPECTEDVUF OF GLOBALPAK
```

The next HiLi intrinsic to be called is HPDENABLETERM. This intrinsic will open the terminal for access. The intrinsic uses one other parameter that defines the type of terminal. This field is a compound parameter. The first part of the parameter is an 88 byte character array that should be set to HPTERM. The second part of the parameter tells if the terminal is a touch screen or not. This field is normally set to 0. There is one other parameter that I have chosen to ignore. This parameter defines a specialized data entry device which will not be discussed in this paper.

The next call is to HPDOPENFORMS after the HPDENABLETERM intrinsic is called. Notice that I am checking the field RETURNSTATUS in the program to insure that the terminal was opened. If the HPDENABLETERM or HPDOPENFORMS intrinsic calls fail, then I display an error message and insure that the program will terminate. This brings up another COBOL 85 function. COBOL 85 now supports nesting IF...ELSE statements terminated within other IF...ELSE statements. The IF...ELSE also introduces the CONTINUE statement which allows the programmer to use true statements for program logic. The second check for RETURNSTATUS uses CONTINUE instead of having an inequality. The IF...ELSE is terminated by END-IF. Each IF requires a matching END-IF.

The next paragraph will do an IN-LINE PERFORM. This IN-LINE PERFORM allows the program to stay inside the body instead of jumping out of the area to another section of the program code. There are two ways to write an IN-LINE PERFORM. I have used the TEST BEFORE type of PERFORM. This means that the test of the field LASTKEYPRESSED will be done before going into the PERFORM'ed statements. If the value of LASTKEYPRESSED was already 8, then the PERFORM'ed code will never be executed. If the opposite is needed, then the PERFORM statement would be:

```
PERFORM WITH TEST AFTER UNTIL LASTKEYPRESSED = 8
```

IN-LINE PERFORM's may also be nested. Each PERFORM is completed with an END-PERFORM statement.

The final point from COBOL 85 in this program is the EVALUATE statement. Finally COBOL has a CASE statement construct! In fact, the COBOL 85 EVALUATE has more power than many other languages. You can EVALUATE multiple fields. Let's look at the EVALUATE that I used. The first line states that the EVALUATE should take a look at the value of the field LASTKEYPRESSED. It then checks against all given values. The WHEN clause states that WHEN the EVALUATE'd field is a given value, do something. In this case, WHEN the value of LASTKEYPRESSED is 0 (ENTER key), PERFORM the paragraph READ-SCREEN. WHEN the value is 3 (f3), PERFORM the paragraph PRINT-SCREEN. WHEN the value is 8 (f8) just CONTINUE (do not take any action). Finally, if any other value is in LASTKEYPRESSED, an error message will be sent to the system informing the user of the key in error (because of the WHEN OTHER clause).

There are three PERFORM'ed paragraphs in the program. The first paragraph is the standard flow of the HiLi code after the terminal and forms file have been opened. The loop is HPDSEND, HPDREAD and, if errors occurred during editing, HPDPROMPT. Let's discuss each of these intrinsics:

HPDSEND calls the following V/PLUS intrinsics;

VCHANGEFIELD
VGETNEXTFORM
VINITFORM
VPLACECURSOR
VPUTBUFFER
VPUTFIELD
VPUTTYPE
VPUTWINDOW
VSETKEYLABEL
VSHOWFORM

This call will retrieve the form from the open forms file and display the form on the screen. By now you should be aware of the first two parameters, GLOBALPAK and RETURNPAK. The next parameter is SENDPAK. This parameter contains three parameters. They are:

DONTENABLEINPUT	This parameter should normally be set to 0 (ignore). If this parameter is set to 1, it will keep the terminal locked. Use this only if you are performing multiple calls to HPDSEND in a row (displaying a header screen then appending a detail screen for example).
WINDOWENH	This parameter allows the program to change the enhancement of the message window. If this field is left blank, the default V/PLUS forms file window enhancement will be used. You may use any combination of 'I' (inverse video), 'H' (half bright video), 'U' (underlined) and 'B' (blinking) or 'NONE' as you would if you were in FORMSPEC.
BYPASSFEATURE	This parameter tells V/PLUS what NOT to do. Zero means to ignore this parameter. 1 tells V/PLUS NOT to initialize the fields with forms specifications. 2 tells V/PLUS NOT to force the operators display area to be refreshed. 3 tells V/PLUS to perform functions 2 and 3.

FORMPAK is the next parameter HPDSEND expects. There are 7 parameters in FORMPAK. They are:

FORMNAME	This is a 32 byte character array that contains the name of the form in the forms file to be displayed. This parameter may be used to refresh the screen. Let's assume that f3 has been defined as the refresh screen key (handy if the user is on a dial up line). When the f3 function key is pressed, the program would call HPDSEND with $REFRESH in this parameter. The screen would be reset and re-initialized and have the data redisplayed.
FORMPOSITION	This field is a throwback to the V/PLUS forms design ideas of appended forms and freeze and appended forms. The possible values for FORMPOSITION are: 0 - clear screen and home form 1 - overlay at beginning of last form 2 - append at end of last form 3 - freeze form and append new form
CHGLISTTYPE	This last section of FORMPAK gets a bit confusing if you haven't used V/PLUS very often. When you create a form, you assign certain attributes to each field. You tell V/PLUS if the field is optional, required, display only. You tell V/PLUS if the field is CHAR, DIG, NUMx, MDY. You tell V/PLUS if the field is half bright inverse, blinking, underlined or none. These are set in place. If, however, you wish to change any of these parameters on the screen (NOT in the forms file), then you may make temporary changes through your program. This is a good practice if, for example, you are using one screen to display employee information. During ADD mode, you want all fields typed in. During change mode, you allow the user to change any field but the employee number. You may change the field to display only with the next parameters. This parameter tells FORMPAK that you will either (0) not change any fields, (1) change fields by the field number assigned by V/PLUS or (2) change fields by the field name (assigned by you). I recommend that you

use the name (2) instead of the number. You will normally not change the name of a form field. There is the chance that someone will renumber the form at some time. This will then wreck havoc with your display.

LISTCOUNT	This field contains the number of fields that you wish to change during this call to HPDSEND.

The next three fields are input as an array.

FIELDIDENT	This field is either a four byte integer or a 32 byte character array depending on how you defined CHGLISTTYPE. This field contains the field number or name of the field to be changed.
CHGTYPE	This numeric field tells HPDSEND the type of change you wish to make to the field identified in FIELDIDENT. The value for CHGTYPE is:

```
    -1 - do not change attributes
     1 - change enhancements
     2 - change field type
     3 - change data type
     4 - change enhancements
     5 - change field type
     6 - change data type
     7 - change field error enhancement
     8 - change field error enhancement
```

Note that values 1, 2, 3 and 8 return the original values back to the program. So, after you call HPDSEND with your changes, the values you sent will be replaced with the original values in the V/PLUS screen.

CHGSPEC	This is an 8 byte character array that will contain the values you wish to change. The possible values are: If CHGTYPE = 1 or 4 You may type in either 'NONE', or any combination of 'H', 'I', 'U' or 'B'. If CHGTYPE = 2 or 5 The values 'O', 'D', 'P', or 'R' are allowed. If CHGTYPE = 3 or 6 The values CHAR, DIG, IMPn, NUM(n), DMY, MDY and YMD are allowed. If CHGTYPE = 7 or 8 The value in this field is ignored.
CURSORPOSITION	This field is a four byte integer or 32 character array. It contains the field number or name that the cursor should be in when the form is displayed. The cursor normally defaults to the first input field on the form.
MESSAGELEN	This field defines the message length to be passed to the message window. This should normally be set to 79.
MESSAGELINE	This field contains the actual message that will be displayed in the message window. This field is defined by HiLi as 256 positions. Only the first 79 are displayed.
DATADESCRIPT	This field contains the number 1-6. This field will be described later in the section on transferring data.
DATABUF	This field contains the data passed from the screen. The easiest (although possibly not the best) way is to define a default data buffer to receive the data. This field may be set to X(1920), the largest possible area that any single V/PLUS form handles.

LABELDESCRIPT	This parameter contains information which will temporarily update the function key labels for this form. The labels will, by default, come from the forms file if this area is not used. The LABELDESCRIPT field is made up of three parameters. The first subparameter tells the number of labels to be changed. The second parameter, which occurs 8 times to coincide with the function keys, contains two fields. The first occurring field is the LABELIDENT. This field contains the label number to be changed. The second field, LABELENH is not used.
LABELBUF	This final parameter contains the new values to be passed to the labels. The parameter occurs 8 times. The main idea to know is that each field is a 16 character byte array. The programmer defines these to line up properly on the screen.

HPDREAD calls the following V/PLUS intrinsics:

```
VFIELDEDITS
VFINISHFORM
VGETBUFFER
VGETFIELD
VGETTYPE
VREADFIELDS
```

HPDREAD passes 8 parameters. The first two parameters are GLOBALPAK and RETURNPAK. The next parameter is READPAK which contains four subparameters.

READTIME	This parameter allows a read timeout. If the field is set to 0, no timeout occurs. The field will force a timeout if the field is set to any value other than zero. The value is interpreted in seconds.
ENABLEREFORMAT	This parameter will reformat the data according to the form before returning the data to the application.
DOREREAD	No, it is not what you may think it is. This field actually doesn't perform any major function. I will explain later how to perform an AUTOREAD. This field should be set to zero.

BYPASSFEATURE	This field should contain the EXPECTEDVUF value.

The next parameter, READITEMS, contains five parameters. This parameter may be substituted for a dummy variable with no change in performance.

ITEMCNT	This field tells HiLi how many active termination item entries there will be.
ITEMENTRY	A table defined in the program that contains termination item types.
ITEMTYPE	An integer that shows the termination item type.
ITEMIDENT	An integer that identifies the key number.
READOPT	An integer that tells how to return data.

Use this parameter to perform an AUTOREAD. An AUTOREAD is defined as the ability to move data from the screen to the data buffer without pressing the ENTER key. The only time that HP transfers data is when the ENTER key is pressed. Data will not be passed when a function key is passed. This may be overridden if the application performs an AUTOREAD. To set up the working storage section of the program, follow the example below:

```
01  READITEMS.
    05  ITEMCOUNT           PIC S9(09) COMP.
    05  ITEMENTRY OCCURS 8 TIMES.
        10  ITEMTYPE        PIC S9(09) COMP.
        10  ITEMIDENT       PIC S9(09) COMP.
        10  READOPT         PIC S9(09) COMP.
```

Set ITEMCOUNT to the number of function keys that will allow data to be returned. Set each ITEMTYPE used to zero which means that the termination will be a read. Set each occurrence of ITEMIDENT to the value of the function key used to pass data. Finally, set each occurrence of READOPT to 3 to return data and any errors from the HPDREAD call.

The next three fields, CURSORPOSITION, DATADESCRIPT and DATABUF have already been discussed. The last parameter, FIELDLIST, will return the fields, by name or number that are in error. I have never used this buffer so I will not discuss it in this paper.

The next HiLi call is HPDPROMPT. This intrinsic will prompt the user with any errors found. The first two parameters, GLOBALPAK and RETURNPAK have been discussed earlier. The next parameter is PROMPTPAK. PROMPTPAK contains 5 subparameters.

REPAINTDATA	This parameter will force a rewriting of data to the device if set to 1. Setting this parameter to zero does not rewrite data.
WINDOWENH	This field will change the message window enhancement field. This parameter has been defined before.
RESETHILITED	Setting this field to zero will not reset flagged fields highlighted due to errors. Setting the field to 1 will reset the field.
DONTENABLEINPUT	This parameter will lock the terminal from operator input, if needed.
BYPASSFEATURE	This field is not used.

The next parameters, CURSORPOSITION, MESSAGE, FIELDLIST, LABELDESCRIPT and LABELBUF have been discussed.

The next HiLi intrinsic call will print the forms to the line printer. This parameter is HPDPRINTFORMS. The first two parameters are GLOBALPAK and RETURNPAK. The next parameter is FORMCONTROL. FORMCONTROL contains four subparameters.

FORMNAME	This field contains the name of the form being processed.
UNDERLINECONTROL	This integer field controls underlining of fields. Setting this field to 0 will not underline print fields. Setting this field to 1 will underline fields.
PAGECONTROL	This integer field will page eject or line feed before a form is printed. Setting the field to 0 will page eject before printing the form. Setting the field to 1 will line feed before printing the form.
ENABLEREFORMAT	This parameter will, if set to 1, reformat data according to the form before printing. No reformatting will take place if the field is set to zero.

The next parameter is FILLCONTROL. This parameter fills or replaces blanks within fields when printing the form. There are three subparameters in FILLCONTROL.

FILLDESC	This parameter will (0) not fill, (1) fill trailing blanks, (2) fill leading blanks or (3) fill trailing and leading blanks.
ENTRYCOUNT	This integer specifies the number of fields to be filled.
FIELDIDTYPE	This parameter tells HiLi the format of the field types (name or number).
ENTRYDESCRIPT	This table tells the application the fields that are to be filled.
FILLCHARACTER	This field contains the actual fill character. The first byte of the four byte field contains the fill character. The last three positions are reserved.

FILLCONTROL is not a necessary parameter. A dummy variable may be passed for this parameter.

DEVICENUM	The next parameter is DEVICENUM. This parameter contains the file number of the list file. If zero is passed to DEVICENUM, the system will print to the device class LP. A note about DEVICENUM. If you wish to close the print form during processing, use the following code:

```
CALL "HPDPRINTFORMS" .....
CALL INTRINSIC "FCLOSE" USING DEVICENUM, \0\, \0\.
MOVE 0 TO DEVICENUM
```

The last two intrinsics will 'shut down' the forms file and terminal. The first intrinsic called is HPDCLOSEFORMS. The first two parameters are GLOBALPAK and RETURNPAK. The last parameter passed is FORMSFILE. This parameter contains the name of the form file.

The last intrinsic called is HPDDISABLETERM. All parameters used in HPDENABLETERM will be used in HPDDISABLETERM.

There are a few other HiLi intrinsics that are available. The intrinsics HPDGETENV, HPDSETENV and HPDGETDESIGN are used to retrieve and set options during execution.

The last piece of information that HiLi offers is data transfer. There are 6 ways to transfer data from V/PLUS screens to the data buffer. Each way offers a specialized way of accessing data. I will briefly discuss each. Each transfer method uses the DATADESCRIPT parameter.

Method A -	This is the simplest method. If DATADESCRIPT is set to 0, all data will be transferred as type CHAR to the receiving data buffer. If DATADESCRIPT is set to -1, no data will be transferred.
Method B -	This method will transfer a subsrange of data from the first byte to the byte selected by the application. Set DATADESCRIPT to 10 and set the BUFLEN to the number of bytes to be moved.
Method C -	This method will move specific fields on the screen, by field number (DATADESCRIPT = 20) or by field name (DATADESCRIPT = 30). It will return to the exact named fields in the application data buffer.
Method D -	This method is the same as method C except the application data buffer need not match the form. The data will be moved by field number (40) or name (50). The data may only come from 1 form buffer.
Method E -	This method is the same as method D except the data may come from multiple buffers. The data will be moved by number (60) or name (70).
Method F -	This method uses the ARB defined by the form in FORMSPEC. DATADESCRIPT is set to 1000 for MPE formatting rules or 1100 to convert real numbers into IEEE format.

This concludes the discussion of the HiLi intrinsics. Now I would like to complete this paper by discussing a few of the many enhancements available in COBOL 85.

This section of the paper will discuss some of the enhancements, by section.

ENVIRONMENT DIVISION.

The SPECIAL-NAMES section now allows SYMBOLIC CHARACTERS and CLASS. For example:

```
SYMBOLIC CHARACTERS CARRIAGE-RETURN IS 14 BELL IS 8.
CLASS A-VALID-GRADE IS "A", "B", "C", "D", "F".

PROCEDURE DIVISION.

     DISPLAY BELL "JOB BEGINNING" CARRIAGE-RETURN.
     IF GRADE-CODE IS NOT A-VALID-GRADE PERFORM ERROR-RTN.
```

PROCEDURE DIVISION.

The PROCEDURE DIVISION allows the user to check for upper and lower case.

```
IF STRING1 IS ALPHABETIC-HIGHER THEN PERFORM CAPITAL-LETTER.

WORKING-STORAGE SECTION.

01  FIELD-1.
    05  FIELD-A           PIC X(10).
    05  FIELD-N           PIC S9(09) COMP.

PROCEDURE DIVISION.
     INITIALIZE FIELD-1 REPLACING NUMERIC BY 999999999
                        REPLACING ALPHANUMERIC BY "COBOL85".
```

The procedure division allows edit masks to contain MOVEable data.

```
WORKING-STORAGE SECTION.
01  PRINT-A               PIC $ZZZ,ZZZ.99CR.
01  HOLD-A                PIC S9(06)V99.

PROCEDURE DIVISION.
     MOVE -76543.21 TO PRINT-A
     MOVE PRINT-A TO HOLD-A.
```

The INSPECT statement has been enhanced to convert data more efficiently.

```
PROCEDURE DIVISION.
     INSPECT FIELD-A CONVERTING "ABCD" TO "WXYZ"
```

Is the same as

```
PROCEDURE DIVISION.
     INSPECT FIELD-A REPLACING
          ALL "A" BY "W"
          ALL "B" BY "X"
          ALL "C" BY "Y"
          ALL "D" BY "Z".
```

To change upper case to lower case is easy.

```
PROCEDURE DIVISION.
     INSPECT FIELD-A CONVERTING
     "ABCDEFGHIJKLMNOPQRSTUVWXYZ" TO
     "abcdefghijklmnopqrstuvwxyz".
```

Tables and arrays may be initialized in the WORKING-STORAGE section.

```
WORKING-STORAGE SECTION.
     01  A.
         05  B   PIC X(10) OCCURS 10 TIMES VALUE SPACES.
```

There is now NO ADVANCING available for the DISPLAY statement. This allows user prompts without the line feed.

```
DISPLAY "TYPE IN YOUR NAME " NO ADVANCING.
```

The DAY-OF-WEEK is now available from COBOL 85.

```
ACCEPT TODAYS-DAY FROM DAY-OF-WEEK.
```

COBOL 85 now allows IF...END-IF constructs that allow the logic to continue through nested IF's. NEXT SENTENCE is not necessary because CONTINUE is now available.

```
IF A > B
     IF C > D
          IF E > F
               PERFORM A-PARA
               PERFORM B-PARA
               IF G > H
                    PERFORM C-PARA
                    IF I > J
                         CONTINUE
                    ELSE
                         PERFORM D-PARA
               END-IF
               PERFORM E-PARA
          END-IF
     END-IF
END-IF
```

In-line PERFORM's are allowed. This has been discussed before.

The one other COBOL 85 addition that will be discussed in this paper is END-READ. This verb allows a more concise coding structure with MPE and KSAM files.

```
READ MPE-FILE
     AT END PERFORM CLOSE-PARAGRAPH
     NOT AT END PERFORM READ-DATA
END-READ
```

Instead of

```
READ MPE-FILE
     AT END MOVE "Y" TO END-OF-FILE.

IF END-OF-FILE NOT = "Y"
     PERFORM READ-DATA
ELSE
     PERFORM CLOSE-FILE.
```

Well, that's about all that I will go into in this paper. I hope that the topics that have been discussed will help your applications. There are many other new verbs available in COBOL 85 that will streamline coding practices. To see examples of some of the items discussed, please review the program listing that follow this page.

```
$control list, map, verbs

 identification division.

 program-id. BATCH00.

 data division.

 working-storage section.

*******
*******              TurboIMAGE Parameters
*******

 01  turboimage-parameters.
     05  image-base-name            pic  x(08).
     05  image-password             pic  x(16).
     05  image-status.
         10  image-condition        pic s9(04) comp.
         10  image-length           pic s9(04) comp.
         10  image-record-number    pic s9(09) comp.
         10  image-back-point       pic s9(09) comp.
         10  image-forward-point    pic s9(09) comp.
     05  image-mode                 pic s9(04) comp.
     05  image-unused-parm          pic  x(02).
     05  lastkeypressed             pic s9(08) comp.

*******
*******              Hi-Li Parameters
*******

 01  hili-parameters.
     05  cursorposition             pic s9(09) comp.

     05  datadescript.
         10  descripttype           pic s9(08) comp.
         10  buflen                 pic s9(08) comp.
         10  rtnbuflen              pic s9(08) comp.

     05  devicenum                  pic s9(09) comp.

     05  fillcontrol.
         10  filldescription        pic s9(09) comp.
         10  entrycount             pic s9(09) comp.
         10  fieldidtype            pic s9(09) comp.

     05  formcontrol.
         10  formnamecontrol        pic  x(32).
         10  underlinecontrol       pic s9(09) comp.
         10  pagecontrol            pic s9(09) comp.
         10  enablereformat         pic s9(09) comp.

     05  formpak.
         10  formnamepak            pic  x(32).
         10  formposition           pic s9(08) comp.
         10  listtype               pic s9(08) comp.
         10  listcount              pic s9(08) comp.
         10  chgentry occurs 1 times.
             15  fieldid            pic  x(32).
             15  changetype         pic s9(08) comp.
             15  changespec         pic  x(08).

     05  formsfile                  pic  x(88).

     05  globalpak.
         10  expectedvuf            pic  x(08).
         10  callprotocol           pic s9(08) comp.
         10  comarealen             pic s9(08) comp.
         10  comarea                pic x(300).

     05  hili-data-buffer           pic x(1920).

     05  hili-unused-parm           pic s9(08) comp.

     05  msgforwindow.
         10  msglen                 pic s9(08) comp.
         10  msgbuf                 pic x(256).

     05  promptpak.
         10  repaintdata            pic s9(08) comp.
         10  windowenh              pic  x(08).
         10  resethilited           pic s9(08) comp.

     05  readpak.
         10  readtime               pic s9(08) comp.
         10  enablereformat         pic s9(08) comp.
         10  doreread               pic s9(08) comp.

     05  returnpak.
         10  returnstatus           pic s9(08) comp.
         10  sublayerstatus         pic s9(08) comp.
         10  returnmsglen           pic s9(08) comp.
         10  returnmsg              pic x(256).
         10  lastitemtype           pic s9(08) comp.
         10  lastitemnum            pic s9(08) comp.
         10  lastitemname           pic  x(32).
         10  numdataerrs            pic s9(08) comp.
         10  numchgfields           pic s9(08) comp.

     05  sendpak.
         10  dontenableinput        pic s9(08) comp.
         10  windowenh              pic  x(08).
         10                         pic s9(08) comp.

     05  termpak.
         10  termname               pic  x(88).
         10                         pic s9(08) comp.

     05  versionvuf                 pic  x(14).

*******
*******         Define Forms Buffer
*******

 01  BATCH00.
     05  todays-date                pic  x(08).
     05  selection                  pic  9(01).

*******
*******            Define any necessary work fields
*******

 01  quit-num                       pic s9(04) comp.

 procedure division.

 main-process.

     initialize turboimage-parameters
     move "  BATCH" to image-base-name

     initialize hili-parameters
     initialize BATCH00
     move 300 to comarealen
     move "BATCHFRM" to formsfile
     move "HPTERM" to termname
     move 79 to msglen
     move 1920 to buflen
     move CURRENT-DATE to TODAYS-DATE of BATCH00
```

```
    move "WRITER" to image-password
    move 1 to image-mode

    call "DBOPEN" using image-base-name,
                        image-password,
                        image-mode,
                        image-status

    if image-condition not = 0
       display "Error in DBOPEN...condition="
                               image-condition
       move 999 to quit-num
       call intrinsic "QUIT" using quit-num.

    move "BATCH00" to formnamepak
    move "BATCH00" to formnamecontrol

    call "HP32424" using versionvuf

    move versionvuf (8:6) to expectedvuf

    call "HPDENABLETERM" using globalpak,
                               returnpak,
                               termpak,
                               hili-unused-parm

    if returnstatus = 0
       call "HPDOPENFORMS" using globalpak,
                                 returnpak,
                                 formsfile
       if returnstatus = 0
          continue
       else
          display
              "HPDOPENFORMS failed...condition="
               returnstatus
       end-if
    else
       display "HPDENABLETERM failed...condition="
               returnstatus
    end-if.

    move
    "Please type in the selection and press ENTER"
    to msgarea


    if returnstatus = 0 and image-condition = 0
       perform until lastkeypressed =  8
          move BATCH00 to hili-data-buffer

          perform hili-standard-process
          move hili-data-buffer to BATCH00
          move lastitemnum to lastkeypressed
          evaluate lastkeypressed
             when 0 perform read-screen
             when 3 perform hpdprint-paragraph
             when 8 continue
          end-evaluate
       end-perform
    end-if

    call "HPDCLOSEFORMS" using globalpak,
                               returnpak,
                               formsfile
    call "HPDDISABLETERM" using globalpak,
                                returnpak,
                                termpak,
                                hili-unused-parm

    call "DBCLOSE" using image-base-name,
                         image-unused-parm,
                         image-mode,
                         image-status

    stop run.

hili-standard-process.

    call "HPDSEND" using globalpak,
                         returnpak,
                         sendpak,
                         formpak,
                         hili-unused-parm,
                         msgforwindow,
                         datadescript,
                         Hili-Data-Buffer,
                         hili-unused-parm,
                         hili-unused-parm

    call "HPDREAD" using globalpak,
                         returnpak,
                         readpak,
                         hili-unused-parm,
                         hili-unused-parm,
                         datadescript,
                         Hili-Data-Buffer,
                         hili-unused-parm

    if numdataerrs > 0
       call "HPDPROMPT" using globalpak,
                              returnpak,
                              promptpak,
                              hili-unused-parm,
                              msgforwindow,
                              hili-unused-parm,
                              hili-unused-parm,
                              hili-unused-parm.

read-screen.

    evaluate SELECTION
       when 1 CALL "BATCH01" using
                             turboimage-parameters
                                   hili-parameters
       when 2 CALL "BATCH02" using
                             turboimage-parameters
                                   hili-parameters
       when 3 CALL "BATCH03" using
                             turboimage-parameters
                                   hili-parameters

       when 4 CALL "BATCH04" using
                             turboimage-parameters
                                   hili-parameters
       when 5 CALL "BATCH05" using
                             turboimage-parameters
                                   hili-parameters
       when 6 CALL "BATCH06" using
                             turboimage-parameters
                                   hili-parameters
       when other move
           "The selection you chose was in error"
            to msgarea
    end-evaluate.

hpdprint-paragraph.
```

```
     call "HPDPRINTFORM" using globalpak
                               returnpak
                               formcontrol
                               fillcontrol
                               devicenum
                               datadescript
                               Nili-Data-Buffer.
```

Complex Data Structures, Omnidex and Relational Retrieval

Kimball G. Everingham
SRI International
333 Ravenswood Avenue
Menlo Park, California 94025
(415) 859-2131

THE DILEMMA:

Over the past couple of decades we have been presented with evermore advanced database management systems and indexing structures. Flat, circular and indexed files; hierarchical, network and relational databases; B-tree, hashed and inverted list indexes have all been provided to us as solutions for our information management needs. Each new development is made in response to the failings of the alternatives currently available.

Unfortunately we rarely find multiple technologies available within one application development solution. When you use KSAM you get B-tree indexing--period. Switch to Image and you get hashed indexing--period. Yet our information management needs demand a combination of these technologies. B-tree, hashed and inverted list indexes all have their advantages and disadvantages over each other for particular kinds of information management problems. Because of this lack of full information management solutions, we are often limited in our ability to apply existing technology to the problems at hand.

The information we strive to manage creates the demand for the kind of structures and indexes we should use, rather than fitting nicely into one structure or indexing solution. For example, relational databases have become popular both at academic and business levels in response to the inflexibility of hierarchical and network databases. While the flexibility of relational databases may, in fact, be essential to some applications, they don't usually give you a choice of indexing options. This leaves you once again needing some capability that doesn't happen to be included in your relational database management system. This is the kind of problem Hewlett-Packard is addressing by providing relational retrieval intrinsics to Image, their network database--someday.

Image, while being a powerful network database, has some significant limitations in its ability to physically model logical entities and relationships--only hashed keys are available and most fields in masters can't be indexed. This is the problem Dynamic Information Systems Corporation (DISC) has been addressing. They have been looking at Image's deficiencies and providing solutions to some of them. The purpose of this paper is to examine the actual indexing techniques and data structures that need to be used by applications developers, and to examine the ability of Image with IMSAM and Omnidex to support them.

INDEXING TECHNIQUES:

It always comes as a surprise to people not familiar with computers that there could be lots of different kinds of indexes, each with all kinds of reasons for and against its use in particular circumstances. Currently the three most important kinds of indexes available are hashed, B-tree and inverted list indexes. Hashed keys provide rapid access to fully specified keys, B-trees provide generic and sequential access to keys, and inverted lists are essential for text management but use a lot of space.

The initial intention of DISC was to enhance Image by making B-tree and inverted list indexing available within Image. They have succeeded in doing this. IMSAM provides the B-tree indexing and Omnidex provides the inverted list indexing. With these new indexing capabilities, Image can compete against any database management system in the retrieval arena.

DATA STRUCTURES:

Retrieval isn't the only thing a database has to do, however. The ability to model and manage all necessary data structures is also essential. One of the supposed advantages of relational databases is that all potential data structures can be represented within its structure. This is certainly an improvement over hierarchical and network databases that cannot represent some recursive and multi-leveled kinds of structures without a lot of coaxing.

Because any data structure can be represented in the relational model, relational databases are supposed to make data structure issues obsolete. This is not so for two reasons. First, the fundamental data structure issue is one of analysis and design not implementation. The one-to-manyness or many-to-manyness of the relationships between entities must be correctly established. Regardless of the kind of database you end up using, if you establish the relationships incorrectly, you will have to rebuild your datasets or relational tables, wrecking havoc on your applications. Second, just because you can represent a data structure doesn't mean you can manage it very well. It is easy to represent a many-to-many recursive relationship in a relational database (or in Image for that matter), but using or maintaining the data is a function of the languages you are using to manage the data. SQL is just as bad it dealing with these complicated data structures as traditional 3gl languages and newer 4gl languages are.

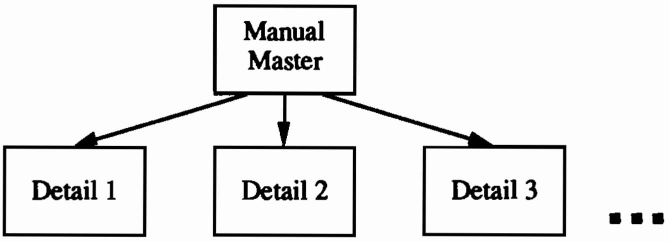
Relational isn't the whole answer to your needs with complex data structures. You need to be able to retrieve, update and report on this information not just store it. For the balance of this paper I will describe the various data structures that applications developers must use and the extent to which Omnidex can help.

ENTITY DATA STRUCTURES:

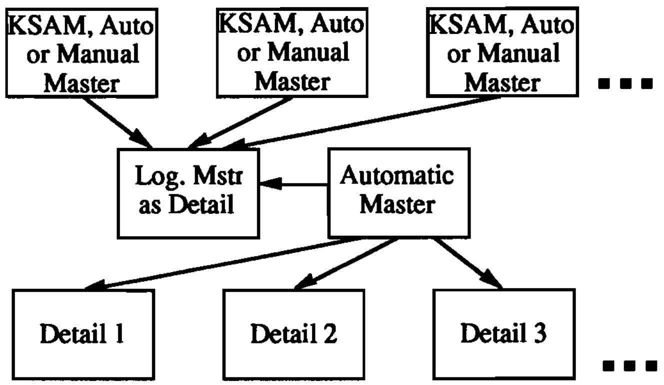
Single entity data structures are easy. As long as you resolutely ignore an entity's relationships to other entities, you will do just fine. Entities can be any class of thing -- even imaginary things, or actions and events treated as things. Companies, cities, parts, orders, potential customers, likely circumstances, instances of movement (shipping) and instances of transfer (transactions) can all be modeled as entities. The basic question is, 'Can it be described as an instance of a class of things?'

Entity attributes are all descriptive of an entity in either a one-to-one or one-to-many relationship. Attributes, per se, can never be any more complicated than that. One-to-one attributes will generally be physically implemented in the same record as the entity itself. One-to-many attributes will tend to be physically implemented in associated records that allow an unlimited number of occurrences. In relational technology, breaking out multiply occurring attributes is known as first normal form.

A generic physical implementation of this structure can be done in Image as follows:



This seems to be the kind of structure the designers of Image had in mind. As all of us know, however, this structure has severe retrieval limitations; namely the manual master can only have one search item field and that one hashed! As application developers began using Image and confronting the limited access of Manual Masters and the restrictions of hashed keys, the following generic structure became common:


KSAM, Auto or Manual Master
KSAM, Auto or Manual Master
KSAM, Auto or Manual Master
■ ■ ■
Log. Mstr as Detail
Automatic Master
Detail 1
Detail 2
Detail 3
■ ■ ■

This allowed a logical master to have multiple search item fields, but you lost the guaranteed referential integrity that a manual master provides. Also, if you used KSAM to provide generic and/or sequential retrieval for your logical master, you had to maintain it programmatically (Yuck!).

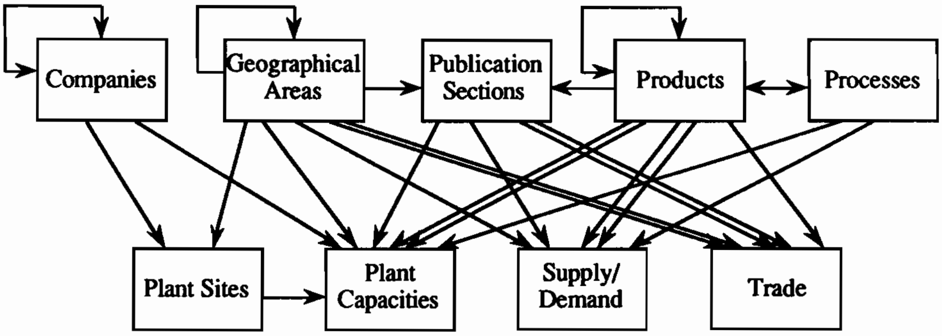
IMSAM, the first product available from DISC, was originally designed to address this particular problem. It is essentially a self-maintaining binary-tree index assigned to particular Image fields. Additionally, Omnidex provides inverted list indexing capability on any field in an Image dataset. The Omnidex inverted list index can be made to behave very much like a hashed key, so now, with IMSAM and Omnidex, the structure in Image of a single entity can be simplified back to its most generic form.

In Omnidex jargon, this group of an Image master (manual or automatic) and its associated details that describe one entity are referred to as a domain. The concept of a domain is fundamental to designing physical structures using Omnidex. The most important thing about a domain is that Omnidex indexing is done across entire domains so that for Omnidex purposes you are dealing with one physical index for each logical entity regardless of the number of detail sets it takes to describe it.

RELATIONSHIP DATA STRUCTURES:

Logical relationship modeling is considerably more complex than entity modeling. Nasty things like many-to-many and recursive relationships only exist for relationships--not for entity attributes. And if relationship modeling is complex, the actual physical implementation is ofttimes daunting in the extreme. At least, entities exist to relationships as single things. Therefore, all the complexity described above as entity structure can be represented in a single symbol with very little loss of meaning, thus vastly simplifying things.

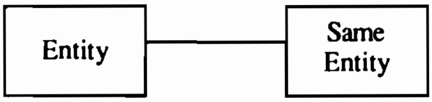
Even with this simplification things can get messy. The following generalized entity-relationship model represents the data I am currently dealing with. In it are embedded most of the kinds of relationships I am covering. Establishing the correct nature of these relationships is part of the job of the systems analyst. It is not easy. Typically users are fond of thinking of actual many-to-many relationships as one-to-many relationships with exceptions. This can be very misleading.



Obviously the point of relationship modeling is to model the logical relationships that actually exist in the data for which you are providing a structure. From the point of view of information there is no distinction between simple and complex relationships. There are just the relationships that exist. In practical terms, however, there are a couple of basic relationships that have traditionally been supported by the tools available to us--database management systems, third and fourth generation languages; and a few relationships that have always been trouble--the infamous bill of materials problem, for example.

SIMPLE RELATIONSHIPS:

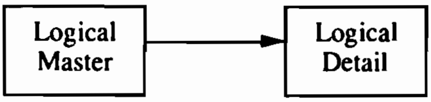
One-to-one relationships:



A one-to-one relationship between entities is a tautological sort of thing. It means that both entities are really the same thing. There are some practical reasons, however, for physically implementing multiple datasets for a single entity. Application enhancements where the original database is untouchable for some reason, and archival sets are two examples.

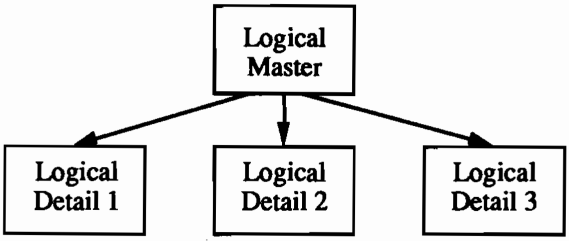
One of the uses of Omnidex id files is to make retrieval from these kinds of structures very easy. Essentially retrievals can be done on two or more physical domains that are logically the same entity, then treated as if they were one.

One-to-many relationships:

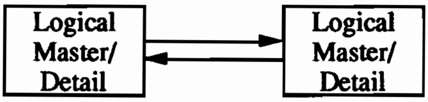


One-to-many relationships are the bread-and-butter of entity relationships. They are often called master to detail relationships, but logically it doesn't make any difference how they are physically implemented. All serious database management systems support these relationships. Fourth generation screen languages are particularly apt to support this relationship to the exclusion of others.

Since a logical master can have any number of one-to-many relationships with logical details, frequently the structure of logical masters and their logical details ends up looking like a generic entity. This is a higher level of generalization, though, as each box can represent multiple datasets.



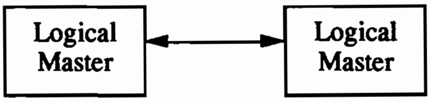
Occasionally you might even see the following unusual combination.



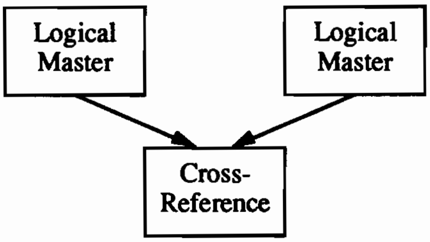
With the new multifind capability of Omnidex, you can retrieve any number of logical master records, then retrieve all their associated logical details regardless of the physical implementation of these sets. In my opinion, this is a giant step for Omnidex and Image. This begins to make available broad relational retrieval capabilities. In an application containing two or more related entities, you get the unlimited ability to move back and forth between them.

COMPLEX RELATIONSHIPS:

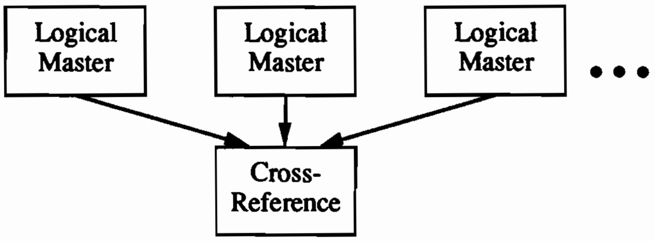
Many-to-many relationships:



Many-to-many relationships quickly break down to multiple one-to-many relationships through cross-reference files.



Cross-reference files are not limited to binary relationships, but can relate any number of logical masters.



The cross-reference files themselves end up getting implemented in Image in a couple of different ways. If the purpose of the file is solely to provide cross-reference, they end up as detail sets. Often, however, a logical cross-reference file is its own logical master with its own logical details. In an Omnidex enhanced system, these may become Omnidex domains of their own.

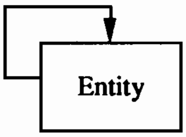
Regardless of the level of importance of the cross-reference file, one characteristic is always present and must be dealt with. Cross-reference records are always created after the creation of all its related logical masters. This usually means that any screen designed to maintain cross-reference files must perform retrieval routines in the middle of creating an entry. Both IMSAM and Omnidex are useful in providing this kind of on-the-fly lookup just because of the general advantage of generic retrieval over fully specified retrievals.

Beyond that simple lookup capability, however, Omnidex is weak in dealing with these kinds of relationships. A cross-reference file can only be assigned to one Omnidex domain regardless of the number of domains of which it is logically a part. There are three ways around this:

- Pick one domain and live with it.
- Make duplicate cross-reference files and put them in different domains.
- Make the cross-reference file a domain of its own and use multifind.

Each of these solutions makes sense in difference situations. I would hope that DISC will see the opportunity here and make it possible to put these kinds of files in multiple domains or perhaps allow individual fields in a single dataset to be in different domains.

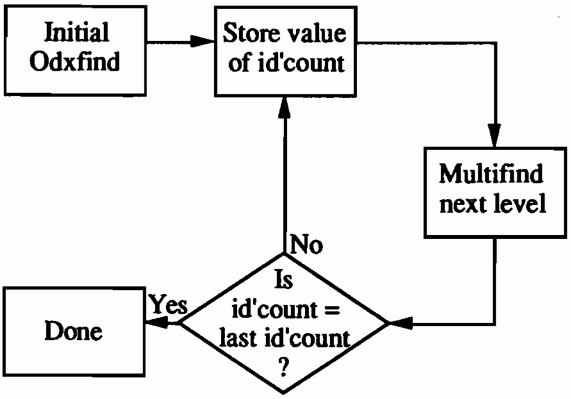
One-to-many recursive relationships:



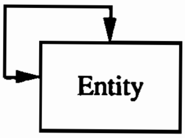
The one-to-many recursive relationship is a simple hierarchy. It can be physically implemented in one file with a unique key and a field for the unique key of that record's logical master. Accounting structures, organization charts, geo-political regions and simple bill of materials hierarchies can be kept this way. Just because this structure can store your information, however, doesn't mean it is going to be easy to manage and retrieve. People always want to know not the records directly 'owned' by another record; but all the records ultimately 'owned' by a record regardless of the number of levels down those records can get.

This kind of structure results in recursive programming. This is a 'do...while' loop containing a call to itself. Techniques to do this kind of work in 4gls tend to be arcane, undocumented and unsupported. It is my understanding that even in 3gls this is a nightmare -- especially in Cobol.

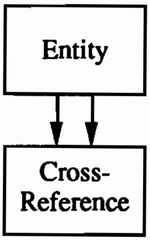
With recent enhancements to Omnidex, called multifind and id files, along with some sneaky indexing, this kind of retrieval can be done in a single 'do...while' loop. The sneaky indexing is accomplished by copying the key of the dataset to a separate field and grouping it with the foreign key representing the logical master. Then the Omnidex retrieval can be done as follows:



Many-to-many recursive relationships:



Like one-to-many recursive relationships, this structure is self-referent, but like many-to-many structures it breaks up into a series of one-to-many relationships.



This is the full bill of materials problem. Not only are you modeling parts as parts of parts, but any part can be part of more than one part. I currently maintain data on companies. Companies can own any number of companies and can in turn be owned by any number of companies. The questions, 'Who are all the ultimate owners of a company?' and 'What are all the companies owned by this multinational worldwide?' are standard. Not only do my users want to see these answered in reports, they want them on their screens.

The ability to retrieve entire hierarchies using Omnidex multifind and id files applies here. However, the cross-reference file is logically in two domains (the owning company and the subsidiary company); and Omnidex will only allow this file to be in one domain or the other. If you want to be able to retrieve both hierarchies (ascending and descending), you are back with the three options available for cross-reference files. Often a duplicate file is an excellent choice here. It is programmatically possible to retrieve the second hierarchy from this Omnidex domain in a different manner, but the solution is not programmer friendly.

CONCLUSION:

The ongoing development of IMSAM and Omnidex retrieval capabilities has made for some remarkable enhancements to the ability of Image to handle complex data structures. It is very powerful in dealing with single entities and complements Image's deficiencies in this area. While useful in dealing with many-to-many relationships, the limitation of one domain per dataset is frustrating. Finally, it has recently become very powerful in dealing with both one-to-many relationships between entities and recursive relationships complementing the deficiencies of Image and most 3gls and 4gls. An IMSAM and Omnidex enhanced Image database is a robust full service database management system.

Create Your Own On-Line Help System

Allen R. Burns
Rutgers - The State University of New Jersey
311 North Fifth Street
Camden, New Jersey 08102
(609)-757-6065

Most HP users are aware that HP has an on-line help system that contains documentation on MPE. Many users are also aware that they may create their own help system. Hopefully this paper will provide you with the knowledge and incentive to create an on-line HELP system for your site. There are many benefits and the time to create the system is a small price to pay compared to the time you will be saving. This may be done for the MPE "Classic" as well as the HPPA MPE XL machines.

I will give examples of the help system we developed and discuss the steps necessary to activate the help file. All of this is available in the System Manager manual in the section titled "MPE Message System". This paper will also address some points to consider as you create your own help file.

As a University, we use our HP 3000 Series 68 strictly for academic purposes. We have 2 terminal labs, 3 microcomputer labs, and over 20 dialup lines servicing users to the HP 3000. The average number of sessions on our system exceeds 60. The users we serve are students, faculty and their support staff. Some of them are very knowledgeable as to the use of the HP, however, our average user would not be considered an expert and many fall into the categories of "novice" or "occasional user". Each semester we experience a new group of users not familiar with the HP 3000. We found ourselves answering the same questions and decided that we would create our own documentation on these most often asked questions.

We began this documentation process by writing short documents of one to three pages to distribute to our users. We have written an extensive amount of in-house documentation over the years. These documents are placed in magazine racks in the various labs and some of greater length may be obtained for the cost of copying them. My interest in creating a customized on-line help system was sparked after reading the October 1985 issue of the INTERACT magazine. It contained an article written by Philip R. Yantis on "The MPE HELP Subsystem", which I would recommend to anyone desiring more information on this topic. Even though the basic concepts are found in the "MPE Message System" section of the System Manager Manual, it wasn't until after I read Mr. Yantis' article that I saw a way to allow our users immediate access to all of the HP 3000 documents that we had written by creating our own on-line help system. The addition of this on-line help system has aided our users tremendously over the last 5 years and I want to encourage others to create similar on-line help systems for their sites. This utility is readily available and free to every site since it comes with the FOS tape on both the "Classic" 3000 and the new HPPA MPE/XL machines.

Establish the ascii file:

The first step in designing a custom on-line help system is creating an ascii file with the information you want to have on line. Although we did this in the line editor, you may use your favorite editor. The file contains command entries used by the MAKECAT program in addition to the information you want to include in the help system. I had a student create the ascii file originally as an independent study project and have used other students to keep the file updated. Since most of the short technical documents we publish are created on PC word-processors, we upload these files to the HP 3000 and "JOIN" them into the master help file. A plain vanilla ascii file created by your PC wordprocessor package works best.

A good place to start is to list all or part of the CICAT.PUB.SYS file to your screen or line printer. This file contains the help messages for HP's on-line HELP, which may be modified or customized should you desire. A "LISTF,2 on the CICAT.PUB.SYS file on our system shows the following:

```
FILENAME  CODE ----------LOGICAL RECORD-----------  ----SPACE----
                SIZE  TYP        EOF       LIMIT R/B  SECTORS #X MX
CICAT            80B  FA       19337       19337  16     6065 29 29
```

The file is an 80 byte, fixed length record, ascii file. A slight disadvantage to customizing this file is each time the system is updated, you will have to update this file. But, if your changes are not significant, you may decide to do this. We have a significant amount of customized documentation that we use and have chosen to create and maintain a separate help system.

Example 1 is taken from the CICAT.PUB.SYS file which I will use to illustrate the commands found in the file. The first part is the very beginning of the file followed by an actual example of the documentation for an MPE command. The final lines show the end of the CICAT file. Bold printed text is used by the MAKECAT program to establish pointers in the help catalog file and will be discussed later.

(Example 1)

```
                         ***(Beginning of CICAT)***
<< LINES .001/.009 ARE RESERVED FOR SYSTEMS INTEGRATION>>
\ENTRY=HELPMENU,SESSIONS,JOBS,PROGRAMS,FILES,MANAGE,OPERATOR,
\CONTINUE,SPOOLER,UTILITY
Information is available on the following classes of commands:
  Running Sessions
  Running Jobs
  Managing Files
  Running Subsystems and Programs
  System Management, Status, and Accounting
  Operator Control
  Spooler Control
  Utility Functions

For more information, enter a KEYWORD.  You can also enter any
command name as a keyword,  Enter 'help' for information on
help.  Enter 'exit' to leave help.
\ITEM=SESSIONS
Running Sessions.  Following are the commands used:
   COMMAND ( ) LOG ON
   ABORT
    ...
    ...
                  ***(Example of a complete command)***

\ENTRY=ABORTIO,PARMS,OPERATION,EXAMPLE
    :ABORTIO

Aborts all pending I/O requests for a device.

SYNTAX

    :ABORTIO ldn

\ITEM=PARMS
PARAMETERS

ldn    The logical device number of the device for which I/O is
       being aborted.

\ITEM=OPERATION
OPERATION

This command causes all pending I/O operations on the specified ldn
to be aborted.  If no queued I/O......
             ..........
JOB/DATA accepting devices always have outstanding read requests
pending due to the auto-recognition ......
             ..........
             ..........
```

```
\ITEM=EXAMPLE
EXAMPLE

To abort all pending output requests for logical device  20, enter:

     :ABORTIO 20
     11:16/3/LDEV#20 NO I/O PENDING

To completely clear spooled device 5 it is necessary to abort all
pending I/O operations as shown below:
      :STOPSPOOL 5
      11:20/31/SP#5/STOPPED
          .......
      :REFUSE 5
      :ABORTIO 5
      :ABORTIO 5
      11:21/40/LDEV#5 NO I/O PENDING
        ...
        ...
```

*****(End of CICAT)*****

```
\ITEM=EXAMPLE

EXAMPLE
To create a multi-line welcome message directly from the standard
input device, enter:

        :WELCOME
        #HEWLETT-PACKARD DEMONSTRATION CENTER
        #WELCOME TO THE HP3000 COMPUTER SYSTEM.
        #NOTE: FILES WILL BE STORED EACH DAY BETWEEN 6AM AND 7AM.
        #return
To copy the message from a file WELMESG.PUB.SYS, enter:
        :WELCOME WELMESG.PUB.SYS
\ALL
```

*****(End of Example 1)*****

The **\ENTRY=** and **\ITEM=** are catalog commands embedded in the text to build the command catalog. These are the most often used catalog commands found in the ascii text file. The **\ENTRY=** command is used to identify the major headings in the help file and the **\ITEM=** command is used to further breakdown the major entry into subentries or categories. Terms following the **\ENTRY=** commands may be used as KEYWORDS to locate and display the major topics in the help information. The format of the \ENTRY= command is:

```
          \ENTRY=topic,comment1,comment2,...
```

(Example 2)

```
        \ENTRY=ABORTIO,PARMS,OPERATION,EXAMPLE

      ABORTIO is the key topic, which has to be alphanumeric
              with no spaces or special characters. Any
              character(s) following the last valid character
              is/are treated as comments. The
              PARMS,OPERATION,EXAMPLE are considered comments
              and are used to list in this case the ITEM entries
              for this command. If needed, a \CONTINUE may be
              used on the next line to continue comments.
```

*****(End of Example 2)*****

The **\ITEM** subtopics are included to present additional text by the help facility. One can get to these subtopics by typing the "ENTRY" topic followed by the "ITEM" topic at the prompt and hitting the return key. Hitting just the return key will scroll you through the "ITEM" topics in order they are listed. We limit the size of the information displayed in our help system to one screen length to prevent information from scrolling off the terminal screen. We then have the user type a letter or hit the return key to display the next page of help. This is helpful as most of our terminals do not have page memory.

Items may be further broken down by a **\SUBITEM**=subitem topic command. This follows the same rule as an **\ITEM** command. It is also listed with the **\ITEM** command line and is accessed in the fashion as an "ITEM" topic. The ascii file **MUST** end with the **\ALL** command to indicate to the MAKECAT program the end of the help file.

Other commands that you may include are **\STOPHELP**, **\STARTHELP** and **\SUBSET**. When the **\SUBSET** command is placed at the beginning of the file, all text between the **\STOPHELP** and **\STARTHELP** commands will not be included in the actual on-line help file created by the MAKECAT program. The file is 80 bytes in length, but nothing beyond column 72 is displayed and it is suggested that you do not place anything on columns 73 to 80 by Philip Yantis in his article as it sometimes causes problems for the MAKECAT program when text is found in this area.

All of this is documented somewhat in the "MPE MESSAGE" section of the System Manager's manual. However, I feel that I would have some difficulty in following the discussion given by the manual. Hopefully the following (Example 3) from our help file will further illustrate the way the file is constructed. [Note: \ITEM=A displays screen 1 and \ITEM=B displays screen 2]

(Example 3)

```
                    ***(Beginning of RHELPIN)***

\ENTRY=RHELP
**** Welcome to Rutgers Camden "HELP" system***

You may type a specific keyword or A for more information.
\ITEM=A
Information on the following topics is available on line
for the users of the Rutgers Camden HP 3000 computer.
The help is designed for the beginner or those forgetting
simple but often used commands.  For further information
the HP help system should be consulted.
To use our "HELP" system you would type:

     :RHELP      (to get into the "Help" subsystem)
or type:
     :RHELP parm1
         or
     :RHELP parm1,parm2

To use the HP 3000 Help System you would have to EXIT
this help system and type:
     :help parm1,parm2

To EXIT either system type E or EXIT at the prompt >

Parm1 and parm2 would be keywords that may be available in this
help system or in HP's help system.  If you get a response
that indicates that there is no help, you may try the advanced
help system of HP or vice versa.
*****For more information type B followed by the RETURN key.*****
\ITEM=B
This help file contains information in the following topics.
To obtain information on any topic type in the key word listed.

FORTRAN FORTPREP FORTGO COBOL ....... etc............... BASIC
MAIL  PLOTTER  ARCHIVING ............ etc.............FTNPREP
FTNGO FTP KERMIT TELNET ............. etc...............HPDESK
         ....
         .... etc....
         ....
*********************************************************
**** RHELP was designed by Nancy Little - July 1986 ****
**** RHELP was updated by Steven Newman - Sept. 1989 ***
*********************************************************
         ....
         ....
         ....
```

```
\ENTRY=FORTRAN
To compile a Fortran program the following command would
be issued. (Fortran 66 Compiler)
        :FORTRAN (textfile) (,uslfile)....
         ....
         ....
**** To CONTINUE hit the A key followed by the RETURN key. ****
\ITEM=A
You would get a listing.....
         ....
         ....
\SUBITEM=OUTPUT
To obtain a HARDCOPY of your compile listing and output...
         ....
         ....
```

{The body of the file continues with "\ENTRY=" commands for all the topics that you want to provide help for on your system. We have help commands for commands such as FTNGO in our system though they are covered by HP's system. We reduced the information given by HP and added examples that go along with the instructions the students see in class. We have tried to display only one screen of information (23 lines) per \ENTRY or \ITEM topic to allow the user to read the entire page before it scrolls from the screen.}

*****(End of RHELP)*****

```
\ENTRY=HPDESK
```

{Remember the last command in your file will be the **\ALL** command. If this is not found by MAKECAT the program will abort.}

```
\ALL
```

*****(End of Example 3)*****

Establish the HELP file:
Now that the ascii file exists, the MAKECAT program is executed to create the HELP catalog file. Thus, I will refer to 2 files, the help file created by you in the editor and the help catalog file created by MAKECAT. The help catalog file is the file actually accessed by the user of the help system. Since MAKECAT may be entered from different entry points, my illustration will specify the "help" entry point on the :RUN command.

Assume the editor file has an MPE name of RHELPIN and the name of the help catalog file we will create is RHELPCAT. The following commands would be used:

```
        :FILE INPUT=RHELPIN.group.acct
        :FILE HELPCAT=RHELPCAT.group.acct
        :RUN MAKECAT.PUB.SYS,HELP
```

This will create a help catalog file RHELPCAT on the specified group and account. MAKECAT has a formal input file name of INPUT and a formal output name of HELPCAT. By issuing the two file equations, you have redirected the formal file designators to your own file names. Make sure that the file RHELPCAT.group.account does not exist prior to executing the RUN command as this will result in an error. MAKECAT always builds a new RHELPCAT file upon execution.

Establish a UDC:

The last step is to make this newly created RHELPCAT file available to your users. This is done by creating a UDC. Our UDC is "RHELP". The command has the following format and is included in our system wide UDC.

```
            *****
            RHELP PARM1= " " PARM2= " "
            FILE CICAT.PUB.SYS=RHELPCAT.group.acct
            HELP !PARM1 !PARM2
            RESET CICAT.PUB.SYS
```

The FILE command changes the file equation from HP's help file CICAT to the file you have created. The RESET command returns the HP system help file CICAT back as the input file for HP's HELP. The two help systems exist independent of each other. Both function the same way but, are accessed by different commands. RHELP accesses the RHELPCAT file that is also on PUB.SYS at our site. Both may be accessed in either immediate mode or subsystem mode. Entering either HELP or RHELP without any parameters places one into the subsystem mode. Entering HELP or RHELP with parameters will give information on the specified parameter(s) and help is terminated. Entering in the subsystem mode allows you to scroll through the help files by entering the \ENTRY, \ITEM, \SUBITEM topics or just pressing the RETURN key.

HELP is an MPE command and there is an explanation of its use in immediate and subsytem mode in the MPE Commands manual. The MPE commands in HP's HELP system all have the following keywords:

```
              PARMS
              OPERATION
              EXAMPLE
              ALL
```

Should you want to obtain HELP on the MPE command SHOWDEV in the immediate mode, you could type any of the following at the (:) prompt.

```
:HELP SHOWDEV
:HELP SHOWDEV,PARMS
:HELP SHOWDEV,OPERATION
:HELP SHOWDEV,EXAMPLE
:HELP SHOWDEV,ALL
```

Showdev is the command or major heading and would appear in the file as \ENTRY=SHOWDEV with \ITEM topics of PARMS, OPERATION,EXAMPLE. ALL will display all documentation on the command.

HELP followed by a UDC command will give you information on the UDC command, unless the "NO HELP" option has been placed into the UDC header line. If you have further information on a command that also happens to be a UDC, the user will have to enter either help system in the subsystem mode to access that information.

We have information available on many topics that would be of interest to all our users in the RHELPCAT file. We could maintain additional HELP files for certain accounts and those help files could be secured to prevent others from reading them by the MPE security features. Should we want to provide certain information just for the faculty we could create a help file just for their account. These different help files could be set for individual accounts or users.

What have we found to be useful in our help system? Information regarding the standard compiler commands and input and output JCL needed for these compilers has been useful. We have condensed the technical documentation and added examples of the JCL needed for the entry level courses. We have on-line information regarding our dialup facilities and settings. Short documents on FTP and Telnet were recently added. Kermit, a popular free pc to mainframe communications program is documented on our help system. We also have information regarding the use of our terminal servers. HPDESK and electronic mail are illustrated in short documents as to how they are to be used at our site. There is no shortage of topics. As the information in these documents needs to be updated, it is quite easy to add or alter the contents of the file. The help system can be updated without affecting the users.

Topics that one may want to include in an on-line help system would be those used by occasional users of the HP 3000. Directions on weekly, monthly or yearly reports could be summarized and placed on the help system. We have departments that keep mailing lists on the HP 3000. The steps necessary to add addresses or create the output lists could be maintained in the help system. Any standard policies on computer use at your site could be posted on the help system. The help UDC could be activated at logon to supply users with the choice of activities available to them on their "logon" with a short summary of the activity they would be doing. Any question you have been asked more than 3 times deserves a place on your on-line help system. New users and occasional users could be directed to this information source from your support group.

I feel that both our users and staff have benefited from our help system. I was forever being asked the protocol needed to call into our system as well as the phone numbers. I now refer the students and faculty to the RHELP facility and tell them to type in "DIALUP". All they need to know is now available for them to copy down or output to printers we have attached to our terminals. We even use the RHELP to display documents for the CSL programs. All of this is available to Classic and HPPA MPE/XL users at no cost other than the time to create the help file. I hope that you will take advantage of this "free" feature and start an on-line help system for your site.

Database therapy: a practitioner's advice for the 90's

F. Alfredo Rego

Adager

Sun Valley, Idaho
83353-0030
U.S.A.

Telephone +1 (208) 726-9100

In 1980, I wrote an article called *Database Therapy: A Practitioner's Experiences*. Now, a decade later, I would like to build on that *Practitioner's* foundation, expanding its applicability to issues that are our concern in the nineties.

We know what we want: our databases should remain *correct* and *available* despite failures. This requires a lot of work. Keeping our guard up means several things. First of all, we should try to *avoid* problems. Unfortunately, we may fail despite our best efforts. If so, we should *locate* and *fix* the errors and we should *remove* the causes.

Throughout the decades, the fundamental ideas and methodologies have remained the same but the buzzwords may have varied slightly.

Buzzword review

This essay applies to database systems and operating systems in general even though it uses Hewlett-Packard's IMAGE and MPE as examples. To save ink, "IMAGE" encompasses IMAGE/3000, TurboIMAGE, TurboIMAGE/XL, and whatever new fancy names Hewlett-Packard may dream up for future versions of its award-winning database management system. By the same token, "MPE" encompasses the MPE and MPE/XL operating systems for the HP3000 family of computers.

Database entries

A database models the dynamic behavior of entities and their relationships by means of *entries*. An entry consists of a key (which uniquely identifies the entity or relationship) and a collection of attributes (which give quality and color to the entity or relationship).

A *dataset* is a homogeneous collection of entries. There are different kinds of datasets, each optimized for a specific access technique.

To make an IMAGE database functional, we access its entries in a variety of ways, ranging from *serial scans* of entire datasets (just as we were forced to do with batch-only machines) to *hashing and chaining* (quite convenient for online applications). Hashing and chaining are techniques based on direct access to specific addresses so that we may jump directly into the entry or entries which interest us without having to wade through millions of irrelevant entries.

Groups of related entries

Typically, we are interested in groups of entries ("one entry by itself" is just a special case of a "group"). The definition of "group" is quite dynamic and context-sensitive, even within the same database. For instance, at the end of the year we may want to analyze all the transactions of a client, whereas at dunning time we may be only interested in the client's unpaid invoices.

To be able to deal with a group of entries, we face two challenges: first, we must *define* the group; second, we must *find* the members (if any) of the group. The definition of a group is not a technological problem but a managerial one. The finding of the group members is the technological problem that databases are supposed to solve.

Ways of finding groups of related entries

We can approach the "finding" challenge in several ways. One method is to sequentially scan the whole dataset applying a "filter" that will find (and separate) the entries of interest, according to our group definition. Another way is to use indexing methods which, hopefully, will provide faster access (at a price, unfortunately: complexity).

Database Indexing

Indexing methods can be addictive! Some purists state that, without exception, pointers should never be used in database technology, since pointers decrease the immunity one has from the effects of logical structural changes. However, almost all **practical** database implementations (including those developed under IBM's DB2) use many B-trees and/or hash tables for indexing, thereby implicitly incorporating pointers. To enforce the use of indices, many DB2 installations prohibit the use of ad-hoc queries ("dynamic SQLs") since they would require serial scans with horrendous performance implications. This is certainly familiar to IMAGE users, who prefer "hard-wired" accesses (by means of IMAGE paths or by means of other indexing methods widely used in IMAGE shops).

Data and metadata

Whether intimately intertwined or physically separated, an indexed database has two parts: the data ("your information") and the indexing metadata. Both parts are treasures to be protected. A well-designed database management system does NOT allow standard users to access, directly, the indexing information (the metadata). IMAGE, for instance, requires privileged mode (enforced by MPE) for access to such delicate matter. By the same token, a well-designed applications system does not allow users to access, directly, your information. Your application, for instance, should have user interfaces that check all transactions against validation tables.

To be ON or not to be ON: that is the BIT question!

When a bit stands by itself, without any relationships whatsoever with other bits, this "on/off" question has no *correct* answer. But when a bit is part of a redundant conglomerate, the question's chance of having a correct answer increases (as a function of the number of redundant "colleagues" in the conglomerate). For example, consider "parity bits". With one parity bit, our only hope is to detect a one-bit mischief; with more parity bits, we can hope to also point our fingers to the offending bit and (if we are really smart and if we want to pay the price in terms of extra bits) to correct it!

IMAGE databases have built-in redundancies (involving critical fields, chain pointers, chain counts, entry counts and "presence" bits). It is mathematically very simple to detect inconsistencies among these redundant IMAGE elements. The hardware (any flavor of HP3000 computer) that supports IMAGE databases has all kinds of error-detecting and error-correcting components (such as memory, discs, tape drives, and so on).

Systems programmers use checksumming techniques to increase the likelihood of catching unauthorized changes to vital data structures. There is no reason why *applications* programmers cannot incorporate analogous checksumming methods that would use an extra field per entry and a few milliseconds of additional arithmetic per database access.

Another technique (championed by Jerry Johnson) is *frequency analysis*. We simply scan a whole dataset and sort out all the different values of the field that interests us, with a count of the number of times that each value appears in the dataset. Usually, values that appear only once merit review, since they might be typographical errors. For instance, if your enterprise does business all over the world and "country" is the field that interests you, it is possible that you may have 154 occurrences of "Guatemala" but 1 case of "Guatamala", pointing to an obvious mistake.

Keeping your guard up

IMAGE is extremely *reliable* (since it has few bugs of its own) and *resilient* (since it shows "compassion" toward the faults of its underlying hardware and operating system). IMAGE, by design, contains spare resources (for instance, the redundancy between pointers and search fields). Nevertheless, IMAGE structures do fail. Such failures may go undetected or they may show up on a user's screen or report. Depending on how your application programs deal (or fail to deal) with database problems, you may end up with a nicely-logged set of diagnostics that pinpoint specific chains or you may find yourself with aborted jobs that say nothing at all. In any case, you must always be on your toes!

Prevention

A bit of prevention can save many megabytes worth of reloading. As an example of an elementary precaution, consider the electrical power that makes our computers tick. Since my early days in Guatemala, back in the seventies, I always have insisted on a healthy standard for my HP3000 computers: uninterruptible power supplies (UPSs) backed up by diesel generators. I am dismayed to see how many big companies (with vast resources) choose not to invest in these elementary measures. I am also very pleased to see that some smart organizations (notably Hewlett-Packard's response centers) have opted for outstanding batteries of UPSs and powerful auxiliary generators.

Fault tolerant technologies typically carry performance penalties. This applies to fault tolerance toward lousy electrical power as well as to fault tolerance toward lousy systems or applications programming (not to mention lousy hardware). We must decide the relative worth of the various options (for instance, consider the extremes of never backing up at all versus backing up every five minutes).

Semantic checking

Even though the structure of your database may be perfect, you may find yourself with a worthless bunch of bits if the database's information does not reflect the reality of your enterprise.

In *A Practitioner's Experiences*, I said:

> "People can (and do) misuse software. They can use QUERY to find all the entries that meet certain criteria and then delete them. They can accomplish the same (good? bad?) thing with a ten-line BASIC program or with the equivalent 100-line structured COBOL program. It does not matter how they do it. If those entries are supposed to be there (according to you), then you should take some preventive action. For instance, use IMAGE logging and Bob Green's DBAUDIT to find out who did what, when, to which entries...
>
> Innocent-looking application programs, as you well know, are one of the worst threats to the consistency of a database. Your best strategy is to build a set of application programs that spend their lives checking the application-dependent consistency of your database."

Backup and (hopefully) recovery

Backup is one thing; recovery is another thing altogether. I quote from my *Practitioner's Experiences* essay:

> "People can store things on the wrong set of tapes, clobbering whatever data was on the tapes! And people can restore from the wrong set of tapes, clobbering whatever data was on disc! A well-organized tape-library system is a good investment, especially if it is itself computerized (after all, you are trying, precisely, to protect yourself from sloppy operators...)
>
> People can physically keep your backup tapes in a hostile environment, thereby rendering them useless. I recommend professional handling of your off-site tape storage. And I recommend that you periodically check the validity of the tapes kept both on-site and off-site. What would happen if you had to restore some file from some tape that was physically impossible to read [or that did not quite contain what it was supposed to have, due to a bug in the backup software]?
>
> People can fail to backup the system at all. Whether they do it intentionally or innocently is irrelevant: the catastrophic consequences are the same. How do you know that the tapes that are supposed to contain your full sysdumps really contain them? Backing things up is a chore. How do you know that your people are not taking shortcuts? I know a user who has an HP3000 machine dedicated to just a single purpose: a reload from the sysdump tapes produced by other computers. If any reload

has any difficulties whatsoever, he takes immediate action to correct the problem while it is *important* but not *urgent*. Most people wait until a problem is *important*, *urgent*, and *impossible* to solve within the given time/resource constraints."

What if you lowered your guard and you now face database damage?

Broken chains are painful. Why should we add insult to injury using primitive methods to detect and fix broken chains? When we are forced to embark on a fixing mission, we are naturally nervous. Particularly if we are not *gurus*. Even bit pushers should feel uncomfortable dealing with cryptic listings in octal (or hexadecimal), since a minor human mistake may have disastrous consequences. All of us have experienced, at one time or another, the frustration that unfriendly systems generate. We all deserve better human interfaces. There is no reason to compound an already tense situation with temperamental database therapy software!

The first step is to detect the errors and to diagnose the faults. Then, we must devise a strategy to recover the database. One approach is to restore the whole database from the latest backup (assuming that the fault happened <u>after</u> such backup) and then use DBRECOV to apply all logged transactions. If this is unfeasible (due to the unbearably long times involved, or to the lack of backup, or to the lack of log files), then we must treat the database in an OnLine manner. Regardless of our choice of methodology, our objective is to put the database back into a stable state. Ideally, we should also find the culprit and remove it so we don't go through the whole thing over and over again.

One of the toughest challenges facing the database therapist is the correction of discrepancies between critical fields and their associated pointer structures. Fortunately, redundancy comes to your help (if the pointers are broken but the SearchField is not, you can reconstruct the pointers; if the SearchField is broken but the pointers are fine, you can reconstruct the SearchField -- a little trickier in the case of master datasets, though). If the pointers are broken <u>and</u> the SearchField is also broken, then a combination of "amputation" and "transplanting" may be necessary.

Typically, diagnostics will identify faulty entries by entry numbers and/or by the values of the offending search fields. In either case, we must eventually define <u>some</u> search field value that will guide us in trying to find the ChainHead. The fact that we find (or fail to find) the ChainHead does not really tell us much if we have not *exhaustively* tested its master dataset beforehand. For instance, even when we find the ChainHead, the involved synonym chain may have been broken in the past and there might be other "floating" duplicate master ChainHeads with the same search field value. Not finding the ChainHead is an even surer symptom of a faulty master dataset. The conclusion is that a complete certification of the involved master datasets is a pre-requisite for doing any kind of chain therapy.

When starting with an entry identified by its entry number, we should not assume that the "key" value of the entry at that location is valid.

When starting with a search field value, we must remember that decimal data formats (IMAGE types "Z" and "P") present a special challenge, since non-negative values may be *signed* or *unsigned*. We must try both, treating them as *separate* requests.

After any fixing, a serial scan of the involved datasets is necessary, to pick up any pieces that may have been left scattered around. This applies to entries as well as to the global dataset HighWater mark, FreeEntry count and detail FreeEntry list.

The post-fixing dataset certification should make sure that there are neither duplicate master ChainHeads nor orphan detail entries.

For checkup (auditing) and treatment (fixing), the technique of zooming is very convenient, since we can select the level of involvement (and the amount of time involved in the process): All or some of the paths in a dataset? All or some of the chains in a path? Consistency between the global counts and pointers of a dataset and the actual entries in the dataset (detail FreeEntry lists, HighWater marks, FreeEntry counts)? Consistency between the root-file tables and the actual datasets as they exist on disc?

A special case of zooming is "windowing" (or "paging", from an MPE viewpoint). IMAGE entries are grouped into MPE blocks. Sometimes, entire MPE blocks are shifted (typically by one byte) as a consequence of hardware or software failures. At other times, entire MPE blocks are over-written by the spooler or other software. If we just look at IMAGE entries, we will get some bizarre diagnostics. If we look at them from the perspective of MPE blocks, we can usually see a more reasonable pattern. Under extreme cases of clobbered blocks, the only reasonable thing to do is to "amputate" (zero out) the offending blocks, perhaps "transplanting" information substitutes from some backup medium and "bridging" the pointers to rebuild the involved chains.

Beware of simplistic solutions

I saw a case where, due to an unlucky coincidence (involving two master chainheads with different search field values pointing to the <u>same</u> detail entry), two chains got hopelessly intermingled. In this case, the search field values were correct but the pointers were incorrect. One (incorrect) way of "fixing" the problem relied on the simplistic approach of changing the "offending" search field values (which were just fine) to match a given chainhead's value. The correct way is to disentangle the pointers to reconnect all the appropriate chain members.

This whole problem could have been eliminated by plugging one of the few holes in IMAGE (the lack of checking for strict correspondence between the search field values for <u>all</u> members of a given chain). If you are serious about improving the quality of IMAGE, please write a note to your Hewlett-Packard software support engineer stating this enhancement request.

Subtleties

The most powerful methods can also be the most dangerous, in the wrong hands. Strong medicines (such as privileged DEBUG or DISKEDIT) need prescription and administration, as opposed to "over the counter" solutions.

Conclusion: "success" and "failure"

There are two possible outcomes once we finish our database therapy operation: "success" and "failure". We should not become too complacent when we think we were successful, since there are still some tough questions (*how* did we get into the mess?, *why* did it happen at all?, *what* can we do to avoid an encore?) And even when we face an apparent failure, we should keep things in perspective and learn as much as possible in the process of going down fighting for our database's life.

Gratitude

I would like to thank the Adager colleagues who helped me refine the ideas and the presentation: Jerry Johnson, Patrick Mullen, Ken Paul, Leslie Keffer de Rego, Fred White and Rene Woc.

DESIGNING USER INQUIRY SCREENS FOR KEYED ACCESS

PAUL EDWARDS
BRADMARK COMPUTER SYSTEMS, INC.
1506 ESTATES WAY
CARROLLTON TX 75006
(214) 242-6660

DESIGNING USER INQUIRY SCREENS FOR KEYED ACCESS

ABSTRACT

The design of user interface inquiry screens is complex. There are many factors to be considered during the design phase. With the use of IMAGE or KSAM, the design is focused around a single key value to be used for the inquiry.

With the increased use of keyed access systems, the time devoted to design of the user interface is increased. These systems provide partial key, generic key, keyword, and relational access. This causes an additional work load on the systems analyst to define the real access needs of a user that does not understand these access systems or his own requirements.

An in-depth discussion of these access methods will be provided and practical examples of inquiry screen styles will be presented.

The design of user inquiry screens has been done much the same way since the HP/3000 environment of IMAGE and VPLUS has existed. The traditional access methods will be reviewed, access key types will be defined, the different types of access available will be discussed, application design requirements will be outlined, and some screen design examples will be described.

TRADITIONAL ACCESS. IMAGE and KSAM are two MPE subsystems used by most applications software developers to provide random and serial access to data. There are several disadvantages in both systems.

IMAGE allows only one key in each master dataset and that key has to be unique. There is no generic or indexed sequential key retrieval. To provide additional keys in a record, a system of automatic/manual masters and details must be implemented. This adds additional structures to a database that can degrade performance and complicate the program access of the database. Sorted chains are used to retrieve data records sorted by other fields. The updating of these chains can cause serious degradation in performance of the application program. IMAGE doesn't provide any relational access capability.

KSAM provides generic key and indexed sequential retrieval of data records. There are performance considerations in the use of KSAM with a large number of users. It doesn't provide the logging and security features of IMAGE. Key file corruption is possible with power/system failures. Key files must be checked for integrity after these events occur. Many people combine IMAGE and KSAM file structures to overcome the generic access limitations of IMAGE. This causes a more complex program environment with the possibility of loss of data integrity and causes more disk I/O.

Besides the performance considerations, the serial access of IMAGE and KSAM files have several disadvantages. The serial access of a master dataset can consume considerable time because all records are read whether they contain data or not. A detail dataset has a delete chain that is used to add records in the empty spaces left by deleted records. If the delete chain is empty, the records are added to the end of the dataset. The result is that a serial read of the dataset will retrieve records in no particular order. Even if a chained read is used, the records are returned in order by their placement on the chain. A sorted chain could be used to ensure a certain sequence, but with the associated performance overhead. KSAM records are stored in chronological order. Access of these records can be by any of the keys which is sorted in order by the key or in chronological order.

KEY DEFINITIONS. The types of keys available are simple, concatenated, grouped, and keyword. Custom and independent indexing can be done, as required. These types of keys provide different ways to access the data records. A B-Tree structure is used to store the keys and pointers.

The simple key is usually a single value in a field. This is the type of field normally used in KSAM or IMAGE. An example of this type of key would a customer number.

Several simple keys can be combined together to form a concatenated key. The order in which they are combined together are significant when retrieval is done by a keyed access system. Alphanumeric sorting order is left to right. An example of this type of key is an invoice number combined with the invoice line item number.

Multiple fields that are all to be searched at retrieval time are called grouped fields. They can be simple or keyword fields. Two address lines are an example of grouped fields.

A keyword key is created for each individual word in a field that is separated by spaces, slashes, dashes or any special characters. To save disk space, a keyword exclude file is available to enter common words that don't need to be keyworded.

Custom keys are used when additional intelligence is required to create the key. A special procedure is used to prepare the key. Data type conversion, reformatting dates, upshifting, embedded keys, and stripping unneeded characters are examples of key preparation.

Independent indexing is used for other than IMAGE databases. This includes MPE flat or word-processing files. The key datasets only contain the key values and any additional values required to be passed to the calling program.

TYPE OF ACCESS. The types of access to records in a keyed system are serial, partial key, generic key, keyword, relational, and Boolean.

Serial access is in sorted sequential order. Access of the indexes by traversing the B-Tree will return the entries in ascending or descending order.

Partial key retrieval used when only part of a value is significant. A wildcard character is used to fill the places that are to be ignored in the search. An example would be a search value HEW??TT that would retrieve any entry starting with HEW followed by any two characters followed by TT.

Generic search is used to find entries beginning with certain characters. A wildcard character is used to signify that all following characters are not significant. BRAD@ will find all entries starting with BRAD with zero or more characters following.

Since keys are created for each word in a keyworded field, the access is not positional. Keyword access is not case sensitive, either. These characteristics enable the user to locate values without regard to location or upper/lower case value. Generic or partial key access can be utilized, also.

Relational access is very powerful. It can provide access with multiple values across multiple fields, datasets, and databases using dynamically-joined indices. A common field in two datasets provide the link between records in each dataset. If a common field is not available, then a projection is made using a common field in a third dataset.

Boolean expressions can provide logical choices to be made qualifying entries. The AND, OR, and AND NOT operators are be used for making multiple criteria retrievals. A method of reduction retrieval can be used against a set of retrieved entries to reduce the number of entries displayed. This is achieved by making multiple DBFINDs against the same set of data records and changing the criteria each time.

APPLICATION DESIGN. Application design tasks are more complex with keyed systems due to the virtually unlimited ways to access the data. Two areas that need to have special consideration are intimate knowledge of the data and in-depth analysis of the retrieval requirements.

Knowledge of the data includes the normal field sizes and field data type descriptions with several other requirements concerning the contents of each field. The type of key, as defined above, must be determined for each keyed field. Any concatenations or grouping of fields must be specified. The length of the key is important so as not to use too much disk space and to provide enough length to make the key as unique as required for inquiry. How the data is represented in the field will determine patterns for partial lookups. For the keyword fields, a minimum number of characters per word will determine which words are keyed. The size of each word and the average number of words in a keyworded field will affect the B-Tree structure.

The retrieval requirements are more complex than with IMAGE or KSAM. The type of access required for each field or key must be selected as defined above. Since Boolean relationships are available between different keys, these relationships must be defined. A thorough knowledge of the user's manual systems must be obtained as with traditional systems. In addition, the user's knowledge of his data retrieval requirements and education level is very important. Since any field in an IMAGE database is now accessible, the user could very quickly be overcome by the complexity of the resulting screen design. The KISS (Keep It Simple, Stupid) principle should be followed. The system of screens can literally be too complex for a user to comprehend or use easily. The user could quickly reject the system design as too complex.

SCREEN DESIGN. A sample database with sample screens for a simple customer invoice system is illustrated in the copy of the slides to follow. A traditional approach will be shown with the addition of a keyed system design added. Also an example of the demonstration system that prompted the writing of this paper is included. It deals with the design decisions made to produce the system for the INTEREX CSL inquiry.

DESIGNING USER INQUIRY SCREENS FOR KEYED ACCESS

INQSCR01 11/18/89

CONTENTS

- TRADITIONAL ACCESS
- KEY DEFINITIONS
- TYPE OF ACCESS
- APPLICATION DESIGN
- EXAMPLES

INQSCR02 11/18/89

TRADITIONAL ACCESS

- IMAGE
  SINGLE/UNIQUE KEY MASTERS
  SORTED CHAINS
  AUTOMATIC MASTERS
- KSAM
- PERFORMANCE CONSIDERATIONS
- SERIAL ACCESS DISADVANTAGES

INQSCR03 11/18/89

KEY DEFINITIONS

- SIMPLE
- CONCATENATED
- GROUPED
- KEYWORD
- CUSTOM
- INDEPENDENT

INQSCR04 11/18/89

TYPE OF ACCESS

- SERIAL
- PARTIAL
- GENERIC
- KEYWORD
- RELATIONAL
- BOOLEAN

INQSCR05 11/18/89

APPLICATION DESIGN

- KNOW YOUR DATA

  TYPE OF KEYS

  KEY LENGTHS

  FIELD CONTENTS

- ANALYZE RETRIEVAL REQUIREMENTS

  TYPE OF ACCESS

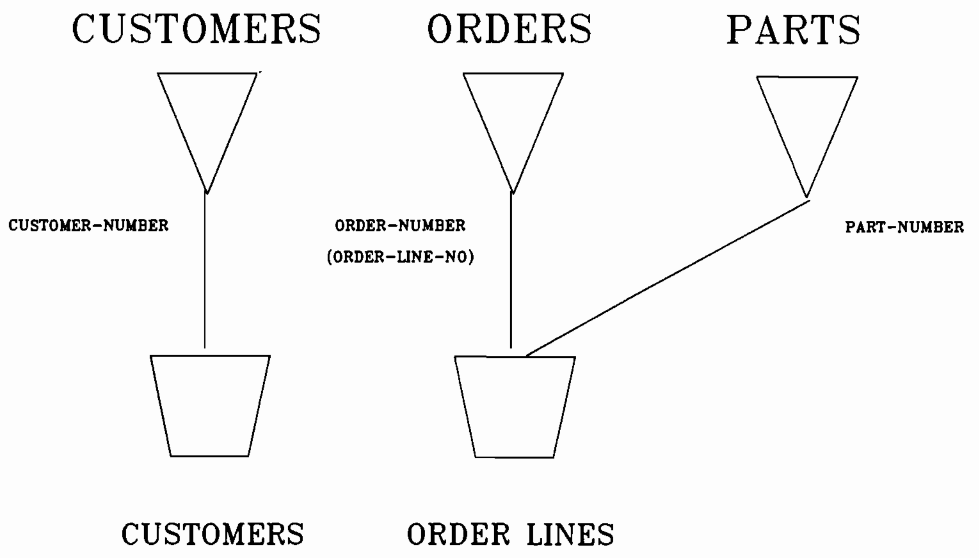
  BOOLEAN RELATIONSHIPS

  KNOW YOUR USER

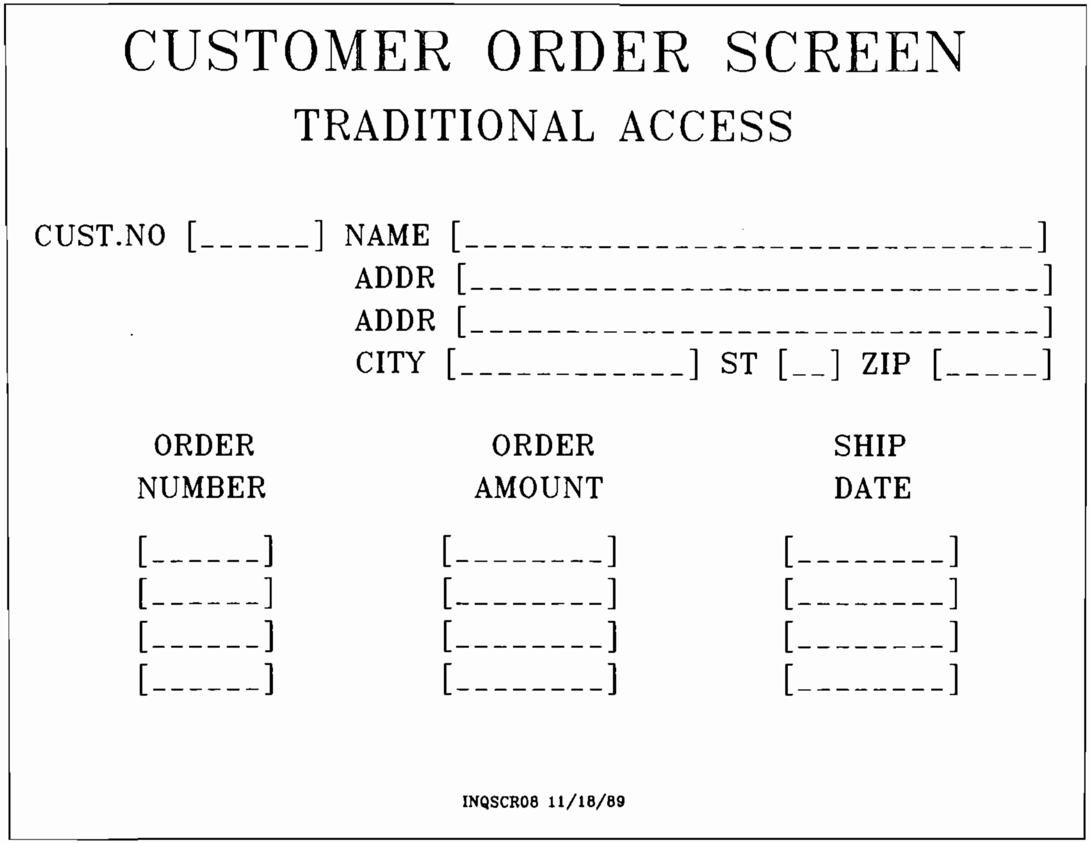
INQSCR06 11/18/89

CUSTOMER ORDER DATABASE

TRADITIONAL ACCESS



INQSCR07 11/18/89

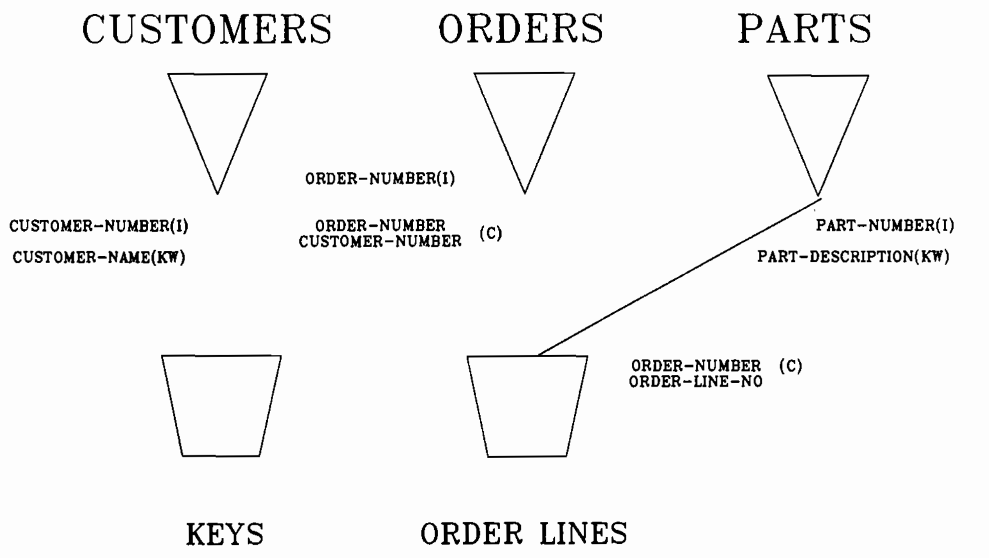


```
                CUSTOMER ORDER SCREEN
                 TRADITIONAL ACCESS

CUST.NO [______] NAME [__________________________________]
                 ADDR [__________________________________]
                 ADDR [__________________________________]
                 CITY [_____________] ST [__] ZIP [_____]

     ORDER              ORDER              SHIP
     NUMBER             AMOUNT             DATE

   [______]          [_________]        [_________]
   [______]          [_________]        [_________]
   [______]          [_________]        [_________]
   [______]          [_________]        [_________]

                 INQSCR08 11/18/89
```

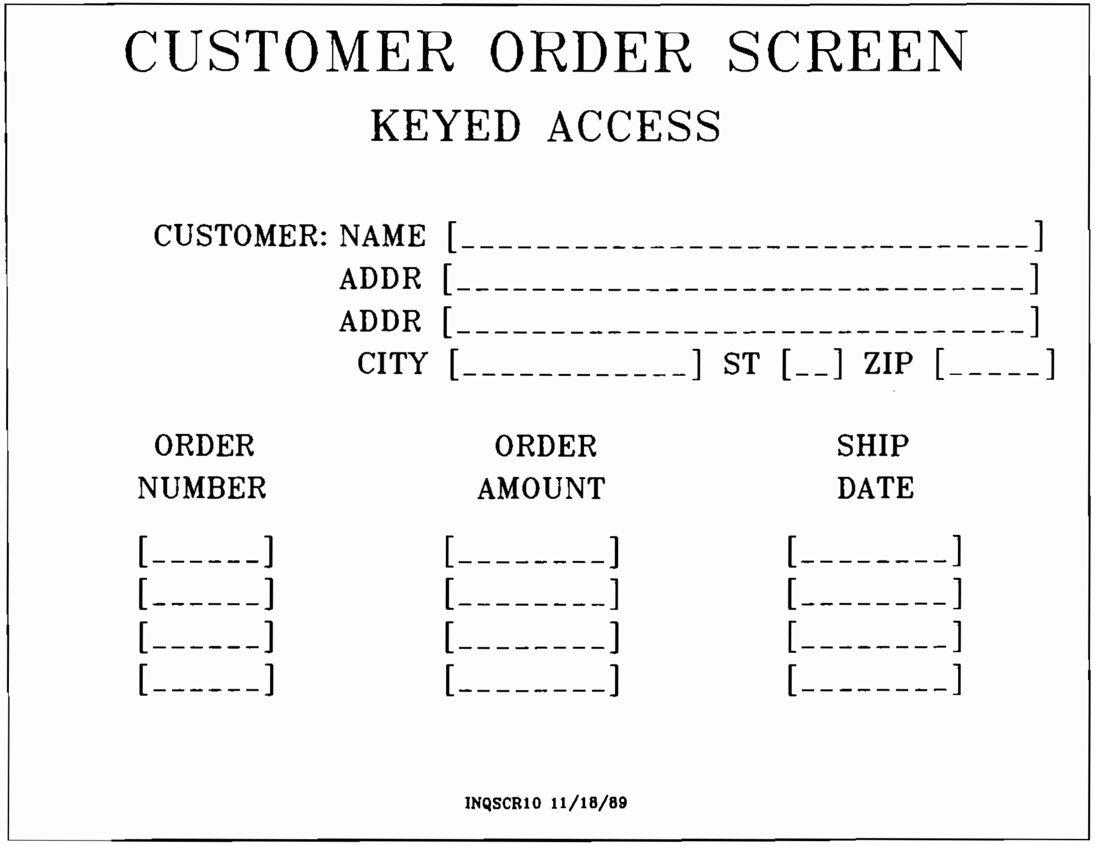
3041-14

CUSTOMER ORDER DATABASE

KEYED ACCESS



INQSCR09 11/18/89



```
                 CUSTOMER ORDER SCREEN
                     KEYED ACCESS

CUSTOMER: NAME [______________________________]
          ADDR [______________________________]
          ADDR [______________________________]
          CITY [____________] ST [__] ZIP [_____]

   ORDER            ORDER            SHIP
   NUMBER           AMOUNT           DATE

  [______]        [________]      [________]
  [______]        [________]      [________]
  [______]        [________]      [________]
  [______]        [________]      [________]

                  INQSCR10 11/18/89
```

3041-16

APPLICATION EXAMPLE

KEYED ACCESS SCREEN

```
            INTEREX INFORMATION INQUIRY SYSTEM
                       CSL INQUIRY

*************** SELECTION FIELDS *****************

CONTRIBUTION NUMBER [____] CONTRIB. NAME [_________]
TITLE KEYWORDS     [____________] AND [____________]
AUTHOR             [_________________]
KEYWORD            [_________________] (Press f1)
ACCOUNT            [_________________] (Press f2)
LANGUAGE           [_________________] (Press f3)
OPERATING SYSTEM   [_________________] (Press f4)
NO. OF ENTRIES FOUND  [_____]

                     INQSCR11 3/7/90
```

APPLICATION EXAMPLE

CSL MASTER SCREEN

```
            INTEREX INFORMATION INQUIRY SYSTEM
                       CSL INQUIRY

NUMBER [___] NAME [____] TITLE [______________________]
               AUTHOR [________________]
              COMPANY [________________]
              ADDRESS [________________]
              ADDRESS [________________]
                PHONE [______________]
        OP. SYS. [_________]   SPEC. CAP. [__________]
        LANGUAGE [_________]     KEYWORD [__________]
        ACCOUNT [____] CATAGORY [____] RELEASE [____]

                     INQSCR12 3/7/90
```

APPLICATION EXAMPLE

CSL DETAIL SCREEN

```
INTEREX INFORMATION INQUIRY SYSTEM
           CSL INQUIRY

NAME [_________] AUTHOR [______________________]
TITLE [___________________________]

[_______________________________________________]
[_______________________________________________]
[_______________________________________________]
[_______________________________________________]
[_______________________________________________]

                 INQSCR13 3/7/90
```

APPLICATION EXAMPLE

CSL CHOICE SCREEN

```
          INTEREX INFORMATION INQUIRY SYSTEM
                     CSL INQUIRY

NUMBER   NAME                  TITLE                      KEYWORD

[____] [________] [______________________________] [________]
[____] [________] [______________________________] [________]
[____] [________] [______________________________] [________]
[____] [________] [______________________________] [________]
[____] [________] [______________________________] [________]

          ENTER NUMBER OF DESIRED CHOICE [____]


                    INQSCR14 3/7/90
```

Fundamentals of IMAGE: Schema & Structure for the New User

Joseph D. Speier

Speier Associates
1720 Section Road, Suite 111
Cincinnati, Ohio 45237
(513) 351-8888

It wasn't so long ago that when you mentioned you were involved with data processing, people assumed you (a) worked for IBM, (b) were as smart as could be and (c) thought a lot. After all, THINK is IBM's motto.

But, that's ancient history. The data processing field has grown and marketing arguments have changed. Instead of think, today's users are urged to touch and feel. Yet, good data processing is impossible without effective thinking.

Consider the meaning of "data processing". It is made up of an object and a verbal noun. Data or information is the object of our efforts. Processing indicates the work to be done with the data.

In the HP 3000 environment, IMAGE is the object that holds the data. The processing, the collection of instructions and procedures that make up our HP 3000 information systems, can exist in one of many diverse languages. Any of the traditional languages referred to as 4GLs, can be used to process the data contained within an IMAGE database. No matter what language we use to handle the processing, the system will be only as good as the database that forms its foundation.

Therefore, an understanding of the data needed by the application area and also a basic knowledge of the structures used to store that data are essential for both the responsible user and the system professional.

In this article, I will attempt to explain IMAGE to anyone new to the HP 3000, curious about how it works, serious about making good information systems and willing to think first and implement later.

To clear up the relationship between an application's data and the IMAGE structures used to store that data, we'll design and define in IMAGE terms a simple database for an advertising application.

The Lesson Begins

Advertising is necessary to inform buyers that your products and services are available. An interested buyer might respond by asking for more information. This creates a lead. Let's consider a system to store those leads.

Find any magazine with ads that contain a "reader service number" that corresponds to a Reader Information Service card located elsewhere in the magazine. An interested reader can use this card to request more information about a product or service. The advertiser regularly receives a list of leads from the magazine.

Our main goal is to keep information about sales leads together with the ad that generated them. What develops is a file of ads containing a "subfile", if you will, of related leads.

A manual system for handling this data would require folders and file cabinets. To operate this system:

* place each ad in a labeled file folder
* place the labeled file folders in a file cabinet
* file all sales leads in the same file folder as the ad that generated the lead
* train someone to keep things running

Our simple, manual system consists of two files. The main file consists of all the manila folders for the ads and is kept in the file drawer. The "subfile(s)" are contained within the folders and consist of pieces of paper with information about the potential customers who responded to the ad.

This system works fine, but your business is growing and the marketing office is running out of space for additional file cabinets. But with an HP 3000 bundled with IMAGE, you can convert this file cabinet system to an electronic database.

An Introduction to IMAGE

IMAGE is Hewlett-Packard's database management system. A database is a collection of files whose data pertains to a common application. This collection is organized under one manager that controls the activities of these files.

The manager is a set of programs that determines the required structure of each database, constructs the database and allows the user to access the data through a set of special programs known as *intrinsics*. These intrinsics prepare the database for processing, place records within the database and retrieve records for reports.

The control which the database manager has over the data has to be well defined. This is specified in a blueprint called the *schema*. Strict rules to specify how the data will be shaped and stored are established before the database can be constructed.

The smallest building block of an IMAGE database is the *data item*. The name of the lead, the zip code, the telephone number and the date the ad ran are some examples of data items. Each item is named and described in terms of its size and data type (alphabetic or numeric).

The next building block is the *record* or *data entry*. Records contain data items for the objects they represent. Records for sales leads contain data items such as the name and address of the prospective client. Records for ads contain items that describe the ad's code, its description, its cost and so forth.

A group of these records, called the *file* or *data set*, form the next building block. The file contains only those records that represent members of the same class of objects. In our example, we need a file for the ads and another for the sales leads.

Methods for Storing Data

IMAGE has two types of files: *detail* and *master*. These are difficult to define in logical terms, but easy to explain in terms of organization.

A detailed data set is an area on a disc divided into a number of like-sized storage locations into which the values for the data items that make up the records will be stored. These locations are pre-assigned and numbered.

They eventually will contain not only the data relating to the application but also numbers of other records in the same file with which they are related, assuming a relationship exists and is specified by the database designer. In our example, leads are related if they come from the same ad.

As information is added to a detail file, IMAGE stores it in the next available slot, usually following the last record added.

A master data set physically is very similar to a detail data set, but IMAGE places records in a master data set in a unique manner. Each master record, by its very nature, contains one data item that will hold a unique value within the master data set. At no time will IMAGE permit two records in a master data set to have the same value. IMAGE accomplishes this by keeping the values that represent records in a field called a "key". No two keys ever are permitted to share the same value. In our manual filing system each folder shows the ad ID on its tab. These IDs should not be the same but could be through negligence.

The IMAGE manager guarantees that this cannot happen. Before IMAGE places a master record, it takes the value found in the key field, runs it through a formula that generates a number and stores the record at the location pointed to by that number. Master records appear scattered throughout the data set, but there is a method behind the apparent madness. To retrieve the record the known ID value must go through the same formula to compute the address. This formula is known as the *hashing algorithm*.

Retrieval

After storing information, you need to retrieve it. There are several ways to meet the demands of most applications. The records in a file can be processed one-by-one in the physical order they were stored. This process can start at the first and continue to the last or start at the last and work to the first. This is known as *sequential* or *serial access*. If you happen to know the number of the storage location containing the record you need, IMAGE lets you acquire that record directly. This is known as *random access* by *record number*.

With a master data set IMAGE will compute the record number and retrieve the record directly if you supply the value of its key. HP refers to this as a *calculated read* because the location of the desired record is calculated by the hashing algorithm.

A useful method of retrieving records from a detailed data set gets only those records that are related. In IMAGE, you can access directly only the related records you desire. You can read them one-by-one from the first to the last or the last to the first. This is known as *chained access*. It saves processing undesired records and is used, for example, when you want to list just the leads generated by a particular ad.

These are the main features of IMAGE. Applying these features starts with data items.

Building a Database

What items of information do we need in order to track the ads we place and the leads they generate?

In practice we would begin by reviewing the information collected in the manual system. An ad's folder contains a cover sheet for the ad, which contains constant information about the ad. Let's keep only the ad identifier, a brief description of its contents, the date the ad was placed, the name of the magazine in which it appeared and its cost. Five distinct items of information will make up the records for the ad file. For the leads, we will keep the person's name, address, phone number and the date the lead was received.

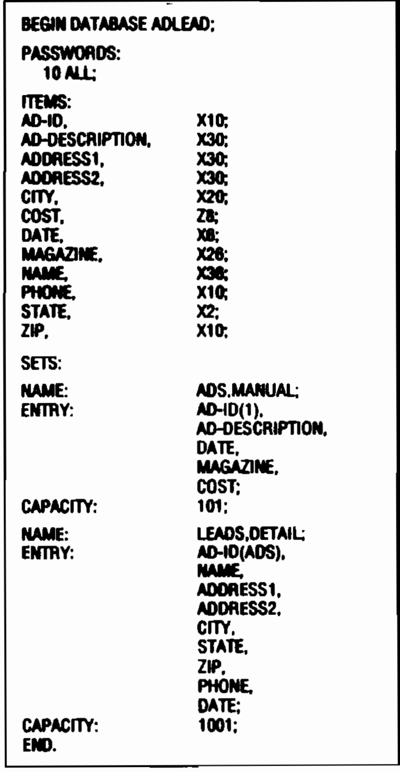
In our manual system, these leads were placed in the same folder as the responsible ad. In an IMAGE database, however, this relationship is achieved by storing the ad identifier with the other data items of the lead. All leads with the same ad identifier form a unit, analogous to the manual system's file folder. This unit is called a *chain*.

The chain begins with the record in the ad file with a unique identifier and continues with the detail lead records that also have this value as their ad identifier. IMAGE maintains the connection between these related records by pointers alongside the application information.

In this example, the ad is superior to its leads: It generates leads. In IMAGE terminology it is a *master file record*. The leads are called *detail records*.

This database is now completely designed.

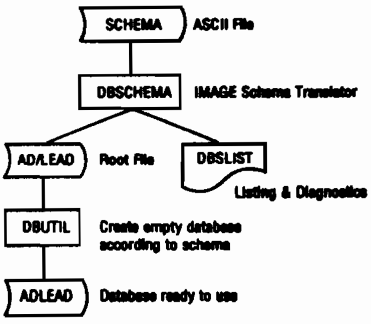
FIGURE I



```
BEGIN DATABASE ADLEAD;
PASSWORDS:
   10 ALL;
ITEMS:
AD-ID,                   X10;
AD-DESCRIPTION,          X30;
ADDRESS1,                X30;
ADDRESS2,                X30;
CITY,                    X20;
COST,                    Z8;
DATE,                    X8;
MAGAZINE,                X26;
NAME,                    X30;
PHONE,                   X10;
STATE,                   X2;
ZIP,                     X10;
SETS:
NAME:                    ADS,MANUAL;
ENTRY:                   AD-ID(1),
                         AD-DESCRIPTION,
                         DATE,
                         MAGAZINE,
                         COST;
CAPACITY:                101;
NAME:                    LEADS,DETAIL;
ENTRY:                   AD-ID(ADS),
                         NAME,
                         ADDRESS1,
                         ADDRESS2,
                         CITY,
                         STATE,
                         ZIP,
                         PHONE,
                         DATE;
CAPACITY:                1001;
END.
```

This schema defines the ad/lead database.

FIGURE 2



The schema is translated into machine language.

The actual construction of the database on the computer is a mechanical process. Not much thinking needed here. The design, which we just completed, must be expressed in a language that IMAGE understands. This is accomplished by entering an ASCII file known as the schema in which syntatical rules are followed in order to specify the database.

This schema, shown in *Figure 1,* has been formatted so that its appearance indicates the functional components. The BEGIN and END components stand for just that. Following the BEGIN DATA BASE is the name you chose for the database. This name must be used in the future whenever you work with the database. It consists of no more than six characters, the first of which must be a letter. Whenever a name is required it's important to choose a meaningful name; one that indicates the function and content of what it names. ADLEAD does this.

PASSWORDS allow for security. Through the passwords, IMAGE can restrict access to the database and limit the range of activities once the database is opened.

ITEMS marks the beginning of a list of data item definitions. Its objective is clear. Each item of information is assigned a name and described by the type of data it will contain. In our example, X is for alphanumeric characters, Z for numeric. Only the cost information will be used in calculations so this data item's type must be numeric, thus the Z. The number following the X or Z indicates the number of these characters the item will need. The ITEMS section must contain every data item that will appear anywhere in the database. It is helpful to arrange them in alphabetical order. Since our list is short this may seem unnecessary, but a database schema with more that 50 data items is easier to study when the data item names are ordered alphabetically.

SETS indicate the beginning of the file specifications. To define the data sets or files we need to specify several attributes. Assign a name to the data set following the label NAME: and indicate its type.

MANUAL indicates a master data set that will contain more data about the record besides its key. (There is another type of master known as an *automatic*, which contains only the key field but we need not discuss this here.) DETAIL indicates a detail data set. ENTRY is followed by the data item names, defined above under ITEMS, that will make up each record. They are listed as they occur on the record. Notice especially that one data item in each of these data sets stands out because of a pair of parentheses following it. It happens to be the same data item, AD-ID, the key to the master and item for grouping key details by ad. On the master, the parentheses contain a number that tells how many detail data sets this master file controls.

In our database, the ads control only one detail file, the leads. The information in the parentheses at the detail level indicates the key item and the master to which it belongs. CAPACITY is followed by the number of records this file eventually will hold. END is the last statement of the schema.

If you ever need to know what an IMAGE database contains and how it is organized, print the latest schema; it is the final authority. If the names were chosen well, you'll be surprised at how easy it is to understand.

The schema that you see has to be translated into machine language. This is done by a program called DBSCHEMA (see *Figure 2*). If there are no mistakes in the schema, it will generate a ROOT FILE, which is used to tell another program, DBUTIL, how to build an empty database. After DBUTIL creates the structure, data can be stored and retrieved. At this point the object of "data processing" is ready for whatever processing language your company has chosen.

This introduction to IMAGE is intended to help you begin thinking effectively about the information systems on your HP 3000. IMAGE is the foundation and the schema the authoritative blueprint.

HOW TO IMPROVE ALL YOUR PRINTED REPORTS USING HP LASERJETS

Jason Kent
Proactive Systems Ltd
Central Court
Knoll Rise
Orpington
Kent, BR6 0JA
England
(Tel: +44-689-77933)

ABSTRACT

Although the use of electronic forms on LaserJets to replace preprinted stationery has become well established, the author will argue that the advantages of using LaserJets to enhance all printed reports have been neglected.

There are many ways reports can be improved by using the typesetting and graphics capabilities of LaserJets. Some ideas for enhancing reports that anyone could implement will be covered, and the software technology that is necessary to do it.

Although the use of electronic forms on LaserJets to replace preprinted stationery has become well established, few people make use of the wealth of features which the LaserJet has to offer so as to enhance all their printed reports. There are many ways in which even the most standard of computer generated reports can be improved by using the typesetting capabilities of LaserJets. I will suggest some ideas for enhancing reports which anyone could implement, and cover the software technology required to do so.

Firstly let us consider some of the features which a LaserJet has which most standard computer printers do not possess. LaserJets are driven by the Printer Command Language (PCL) which enables it to use a vast variety of typefaces of varying style, size and weight. These typefaces can be proportional fonts; that is where the letter "I" would take up less space on the page than the letter "M". Compare this to a monospaced font where all the characters take up the same space regardless of their size e.g. standard typewriter print. How many times have you wished that you could get more than 132 characters across a page? For us, as computer users, the use of proportional fonts means that we are no longer constrained to using a fixed 10 or 12 characters per inch. If we wish to get more text in a set space we could chose a proportional and/or a smaller font. From the point of view of the reader, proportional fonts are far more pleasing to the eye and therefore more likely to be read. Take for example the following two short paragraphs.

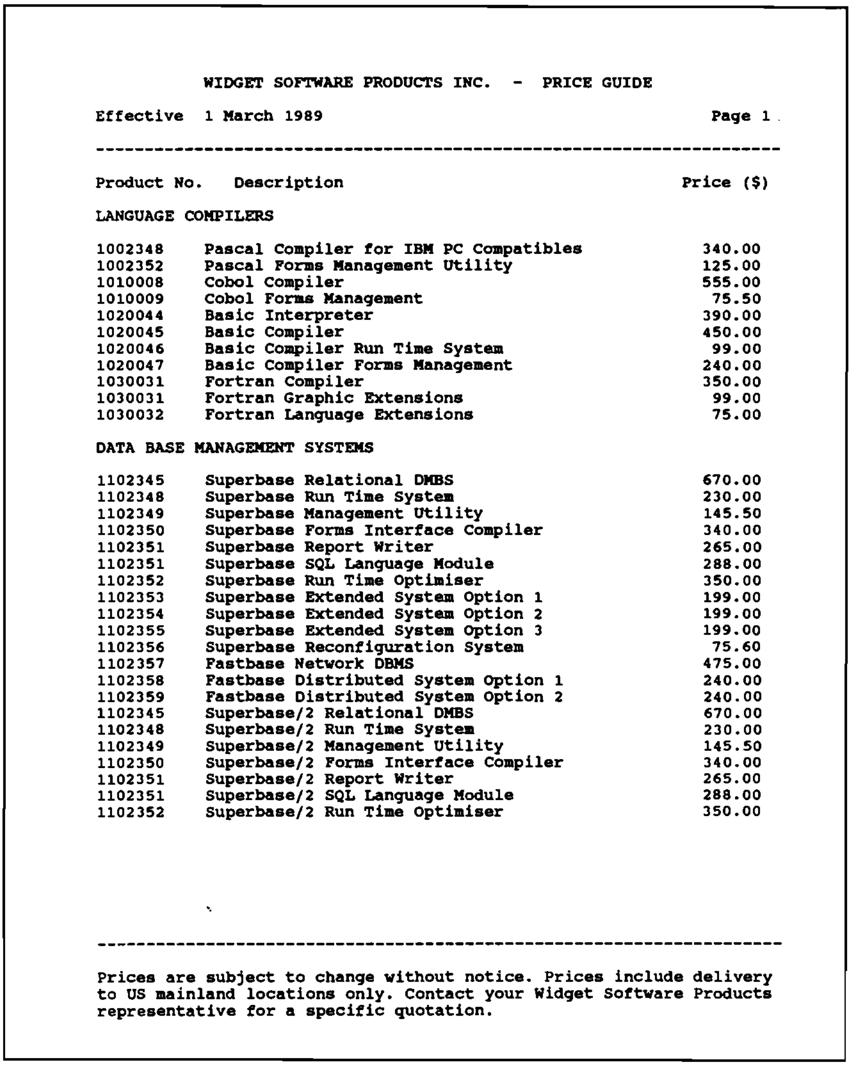
```
This    paragraph    is    printed    using    a
monospaced font. It has a fixed pitch of
10 characters per inch and is the type
of output generated from most computer
printers    and    typewriters.    It    is
readable,  but  I  think  you  will  agree
that  it  is  not  as  readable  as  the
paragraph below.
```

This paragraph is printed using a proportional font. Note that "I"s take up less space than "M"s. Using this typeface frees us from using a fixed number of characters per inch. Publishers use this type of font for printing books because it is more readable than the monospaced font above.

The advantage of using a proportional spaced font for your reports is that you will make it look nicer so that people will be tempted to read more of it. Also you can save paper at the same time (proportional fonts use less line space to maintain the same standard of legibility).

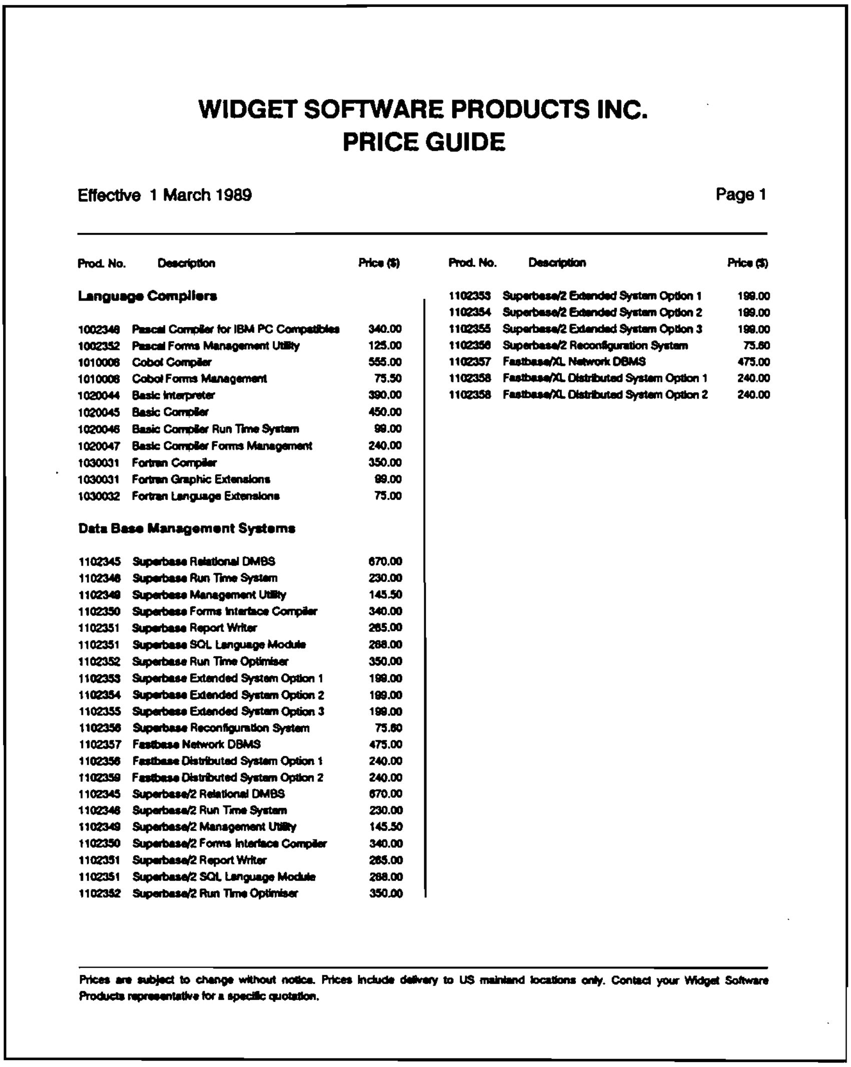
Another feature of PCL is it's ability to draw lines, boxes and a whole host of graphics. At the very simplest level, the use of lines can easily replace the dotted lines which a standard printer produces when trying to underline or emphasise an area of the report. Because LaserJets are page printers it is possible to create continuous vertical as well as horizontal lines, which are essential for drawing boxes. Take the following three examples of the same report. The first example (Figure 1) is a standard line printer style report. Notice the monospaced font and the use of dotted lines. That same report appears in Figure 2 after processing by appropriate software and printing on a LaserJet. Notice the impact given by solid lines, in particular the vertical one. The fonts are all proportional, and give a large heading, bold sub-headings and very readable data. Because a smaller proportional font has been used for the data it is actually possible to get more data on the page, by fitting two columns side by side, without any loss of legibility. The third example (Figure 3) shows the same report again but in landscape orientation. By using this feature of the LaserJet it is possible to spread the report out, improving the legibility still further. This example is in a compact and presentable form which most companies would be more than happy to show to their customers.

Figure 1



```
          WIDGET SOFTWARE PRODUCTS INC.  -  PRICE GUIDE

Effective  1 March 1989                                   Page 1

---------------------------------------------------------------------

Product No.  Description                                 Price ($)

LANGUAGE COMPILERS

1002348     Pascal Compiler for IBM PC Compatibles        340.00
1002352     Pascal Forms Management Utility               125.00
1010008     Cobol Compiler                                555.00
1010009     Cobol Forms Management                         75.50
1020044     Basic Interpreter                             390.00
1020045     Basic Compiler                                450.00
1020046     Basic Compiler Run Time System                 99.00
1020047     Basic Compiler Forms Management               240.00
1030031     Fortran Compiler                              350.00
1030031     Fortran Graphic Extensions                     99.00
1030032     Fortran Language Extensions                    75.00

DATA BASE MANAGEMENT SYSTEMS

1102345     Superbase Relational DMBS                     670.00
1102348     Superbase Run Time System                     230.00
1102349     Superbase Management Utility                  145.50
1102350     Superbase Forms Interface Compiler            340.00
1102351     Superbase Report Writer                       265.00
1102351     Superbase SQL Language Module                 288.00
1102352     Superbase Run Time Optimiser                  350.00
1102353     Superbase Extended System Option 1            199.00
1102354     Superbase Extended System Option 2            199.00
1102355     Superbase Extended System Option 3            199.00
1102356     Superbase Reconfiguration System               75.60
1102357     Fastbase Network DBMS                         475.00
1102358     Fastbase Distributed System Option 1          240.00
1102359     Fastbase Distributed System Option 2          240.00
1102345     Superbase/2 Relational DMBS                   670.00
1102348     Superbase/2 Run Time System                   230.00
1102349     Superbase/2 Management Utility                145.50
1102350     Superbase/2 Forms Interface Compiler          340.00
1102351     Superbase/2 Report Writer                     265.00
1102351     Superbase/2 SQL Language Module               288.00
1102352     Superbase/2 Run Time Optimiser                350.00

---------------------------------------------------------------------

Prices are subject to change without notice. Prices include delivery
to US mainland locations only. Contact your Widget Software Products
representative for a specific quotation.
```

Figure 2



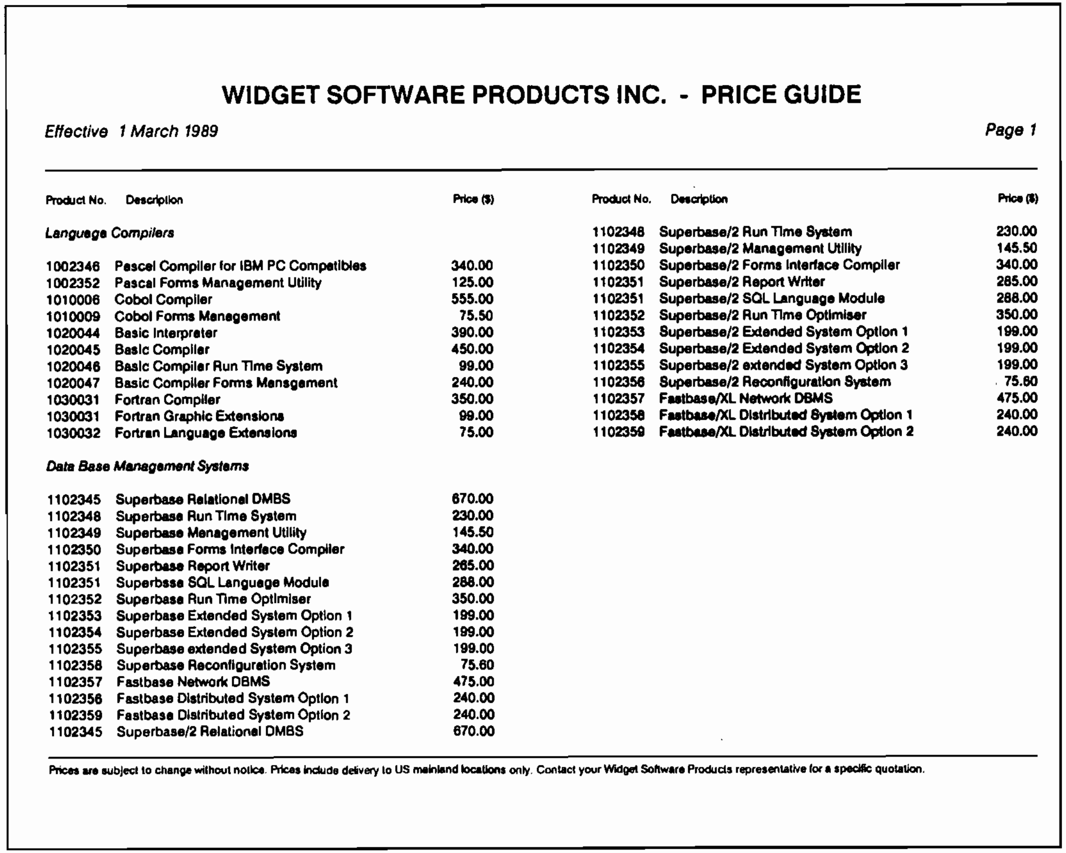
WIDGET SOFTWARE PRODUCTS INC.
PRICE GUIDE

Effective 1 March 1989 Page 1

Prod. No.	Description	Price ($)
Language Compilers		
1002348	Pascal Compiler for IBM PC Compatibles	340.00
1002352	Pascal Forms Management Utility	125.00
1010008	Cobol Compiler	555.00
1010008	Cobol Forms Management	75.50
1020044	Basic Interpreter	390.00
1020045	Basic Compiler	450.00
1020046	Basic Compiler Run Time System	99.00
1020047	Basic Compiler Forms Management	240.00
1030031	Fortran Compiler	350.00
1030031	Fortran Graphic Extensions	99.00
1030032	Fortran Language Extensions	75.00
Data Base Management Systems		
1102345	Superbase Relational DMBS	670.00
1102346	Superbase Run Time System	230.00
1102349	Superbase Management Utility	145.50
1102350	Superbase Forms Interface Compiler	340.00
1102351	Superbase Report Writer	265.00
1102351	Superbase SQL Language Module	268.00
1102352	Superbase Run Time Optimiser	350.00
1102353	Superbase Extended System Option 1	199.00
1102354	Superbase Extended System Option 2	199.00
1102355	Superbase Extended System Option 3	199.00
1102356	Superbase Reconfiguration System	75.60
1102357	Fastbase Network DBMS	475.00
1102358	Fastbase Distributed System Option 1	240.00
1102359	Fastbase Distributed System Option 2	240.00
1102345	Superbase/2 Relational DMBS	670.00
1102346	Superbase/2 Run Time System	230.00
1102349	Superbase/2 Management Utility	145.50
1102350	Superbase/2 Forms Interface Compiler	340.00
1102351	Superbase/2 Report Writer	265.00
1102351	Superbase/2 SQL Language Module	268.00
1102352	Superbase/2 Run Time Optimiser	350.00
1102353	Superbase/2 Extended System Option 1	199.00
1102354	Superbase/2 Extended System Option 2	199.00
1102355	Superbase/2 Extended System Option 3	199.00
1102356	Superbase/2 Reconfiguration System	75.60
1102357	Fastbase/XL Network DBMS	475.00
1102358	Fastbase/XL Distributed System Option 1	240.00
1102358	Fastbase/XL Distributed System Option 2	240.00

Prices are subject to change without notice. Prices include delivery to US mainland locations only. Contact your Widget Software Products representative for a specific quotation.

Figure 3



WIDGET SOFTWARE PRODUCTS INC. - PRICE GUIDE

Effective 1 March 1989 *Page 1*

Product No.	Description	Price ($)
Language Compilers		
1002346	Pascal Compiler for IBM PC Compatibles	340.00
1002352	Pascal Forms Management Utility	125.00
1010006	Cobol Compiler	555.00
1010009	Cobol Forms Management	75.50
1020044	Basic Interpreter	390.00
1020045	Basic Compiler	450.00
1020046	Basic Compiler Run Time System	99.00
1020047	Basic Compiler Forms Mensgement	240.00
1030031	Fortran Compiler	350.00
1030031	Fortran Graphic Extensions	99.00
1030032	Fortran Language Extensions	75.00
Data Base Management Systems		
1102345	Superbase Relational DMBS	670.00
1102348	Superbase Run Time System	230.00
1102349	Superbase Management Utility	145.50
1102350	Superbase Forms Interface Compiler	340.00
1102351	Superbase Report Writer	265.00
1102351	Superbsse SQL Language Module	288.00
1102352	Superbase Run Time Optimiser	350.00
1102353	Superbase Extended System Option 1	199.00
1102354	Superbase Extended System Option 2	199.00
1102355	Superbase extended System Option 3	199.00
1102358	Superbase Reconfiguration System	75.60
1102357	Fastbase Network DBMS	475.00
1102356	Fastbase Distributed System Option 1	240.00
1102359	Fastbase Distributed System Option 2	240.00
1102345	Superbase/2 Relational DMBS	670.00

Product No.	Description	Price ($)
1102348	Superbase/2 Run Time System	230.00
1102349	Superbase/2 Management Utility	145.50
1102350	Superbase/2 Forms Interface Compiler	340.00
1102351	Superbase/2 Report Writer	285.00
1102351	Superbase/2 SQL Language Module	288.00
1102352	Superbase/2 Run Time Optimiser	350.00
1102353	Superbase/2 Extended System Option 1	199.00
1102354	Superbase/2 Extended System Option 2	199.00
1102355	Superbase/2 extended System Option 3	199.00
1102356	Superbase/2 Reconfiguration System	75.60
1102357	Fastbase/XL Network DBMS	475.00
1102358	Fastbase/XL Distributed System Option 1	240.00
1102359	Fastbase/XL Distributed System Option 2	240.00

Prices are subject to change without notice. Prices include delivery to US mainland locations only. Contact your Widget Software Products representative for a specific quotation.

So much for improving the textual side of reports. What about the numeric data which is often found in reports. The LaserJet is capable of far more than fonts, lines and boxes. It can also produce shading and patterns. Both of which could be exploited for improving the understanding of numeric data. Look at the following two tables of figures:

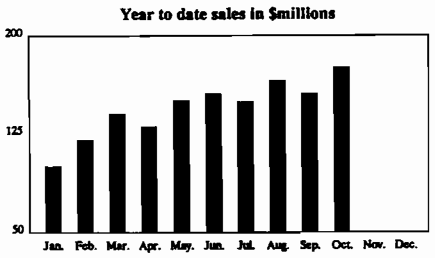
Year To Date Sales In $ Millions

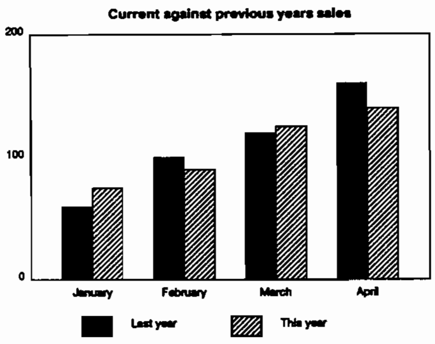
Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
100	120	140	130	150	155	148	165	155	175		

Current Against Previous Years Sales

	January	February	March	April
Last Year:	65	97	116	170
This Year:	74	88	120	135

Now compare the above representation of the figures with the representation of the same figures in the following charts. By using the shading and pattern facilities of the LaserJet we can create charts of the sort which have for years been used to convey the relationships between data, and to show the trends that are apparent.





And you should bear in mind that you can produce charts of the above kind at a very fast rate on the LaserJet, certainly at several pages per minute, so high-volume business graphics because practical for the first time.

The most important difference between a LaserJet and a standard computer printer, whether printing text or graphics, is its print quality. The LaserJet has a printing definition of 300 dots per inch. This gives it unrivaled quality which is not far short of that provided by phototypesetting devices, but at considerably less cost.

Having mentioned some of the features found in LaserJets which can be used to greatly enhance a standard report, I will now discuss how these can be used together to produce computer generated reports which would be hardly distinguishable from externally printed reports. The key to a good report is in the design. The intention of a report and therefore the design, is to convey the message of the data in it to the reader in the easiest, simplest and most economic way. A successfully designed report is generally one in which an acceptable balance has been struck between a number of conflicting demands. I will show that the invisible reconciling of opposing requirements is the essence of communications design.

Ideas and information may be conveyed in many ways. But if conviction, some durability, ease of distribution, economy and a reasonable speed of production are all required, the printing of text and pictures on paper can hardly be bettered. Although typesetting and printing methods change with changes in technology, the requirement for a printed end-product remains the same: to communicate as directly as possible. Visual design or appearance is only the means to that end. If it hinders or restricts the communication process it is failing the reader.

So many of the advantages of printed reports relate to rational design: ease of reading text, of identifying pictures, and of finding your way through the report, being obvious ones. Less evident are the advantages of using the various conventions regarding size and the relative positions of text and pictures which are already understood by the reader; or the cumulative effect of page after page if the report is treated consistently. Further, the report itself can be long or short; it may be a frequently produced report or a one-off; it can be mainly text supported by a few graphics or mainly graphics supported by a few words. All can be accommodated within a suitably articulated design which provides the reader with something easy to follow.

Design is the unifying factor which relates a companys reports, giving overall cohesion (without losing sight of the need for variety and visual incident), and assisting the reader to absorb the data, receive the message and respond accordingly.

TEXT SETTING

As we get closer to practicalities a further generalisation should be noted. There is another reconciliation; that between legibility and impact, between ease of reading and stimulation to the reader. Standard line printer output is inflexible in its ability to print lines and the use of other typographic devices. The LaserJet can make use of these devices for signposting, catching the eye, drawing the reader into the report and directing them to the data. The result of using graphic arts techniques to produce reports is the loss of the monotony of standard computer reports. If we take the example of the six column tabloid newspaper, we can see how it uses large headings that were designed for news stall prominence so as to attract customers to buy it. We too can use fonts and text positioning to give prominence to our computer reports, or to data items and fields. However, over-repeated highlights or fussy design interfere with the reading flow, so balance is the key.

What general rules are there for making up report layouts or designing masters which will serve as guides for subsequent reports? Firstly, although type is for reading, not all reading habits are identical. Some readers can be safely left to get on with it after a heading and perhaps a minimum introduction. Others need more of the aids which LaserJets and typography have to offer; straplines above main headings; explanatory introductions; bold continuation lines to maintain the thread or even repeat and reinforce particular items; enlarged quotations of the main points for those who just will not read the body type; bold paragraph headings for emphasis; hanging indents for itemising, and so on.

All of these devices can have a part to play in facilitating comprehension. But they should not unbalance the page by excessive contrast. Balance, that is to say having units rationally related in scale in the interest of comfortable reading, is always the goal.

A common mistake of people new to typesetting is to use too many fonts with a fussy layout. It's easy to spot the mailshots produced by someone who has taught himself to use a DTP system and hasn't taken any training in graphic design or typography. And if you are using soft fonts on a LaserJet, every extra font takes up memory - so for high throughput without buying extra memory, and in the interest of clean design, stick to a few consistent fonts.

Further ways to direct or inform the eye include putting items in boxes, overprinting them, etc. These methods are also used to produce "labels" for recurring features in the report. Such signposting not only adds a decorative element to the page but assists the reader to find his way and recognise the material he seeks. Similar devices help to solve the report designers problem of grouping a number of loose items together.

There are also many typographical devices such as arrows, fists, blobs, enlarged figures, punctuation marks, asterisks, thick and thin rules, fancy border units and the like, which used sparingly, can relieve a dull area or emphasise something special.

USING PICTURES

So much for the typographical contribution; how about pictures and graphics? These can tell a story, "a picture is worth a thousand words", enhance or validate a set of data, or serve as a single report to represent a set of data. The picture could be an electronic copy of your logo, but now you have the flexibility to position it wherever you like.

When using graphics elements to illustrate the data in your report the golden rule is to keep it simple. Let different treatments mean the same thing each time. Don't make changes for changes sake; each change should alert the reader to something, to signpost him. Variety should come from bringing out different aspects of the material itself or from contrasting material.

It is up to the reports designer to decide how to use graphics to bring out the important data or message in the report. Graphics have no value unless they elucidate something and make understanding the report easier.

This leads to copy analysis. This means reading the text and figures of the report to establish what is important and what needs bringing out by typographical emphasis, which mixture of text styles and graphics give the easiest correct understanding of the report. There are many ways to present a report; copy analysis is helpful in narrowing down the possibilities.

Two other related matters need touching upon, fashion and congeniality. There are fashions in print design as in everything else. The average person sees a wide variety of printed matter which has been professionally designed. Besides newspapers and magazines with their highly sophisticated advertisements, he sees poster hoardings and public notices, publicity material and glossy literature, sober instruction manuals and other printed aids. So he has some instinctive notion of what well assembled text and pictures generally look like, and he may have developed an ability to make comparisons. The element of of fashion is unconciously conditioning his response.

CONGENIALITY

Congeniality is the same sort of thing only more so. In a series of legibility research tests it was found that mathematicians achieved significantly higher results reading material printed in Modern No. 20 typeface, which contains the full complement of mathematical signs and symbols, and is therefore frequently used for mathematical text books. This sort of influence is known as the congeniality factor. It means that the designer needs to take some account of the reading matter that the majority of the reports readers are likely to be familiar with. But beware of inflated presentation of comparatively trivial material as this may lead to loss of confidence on the part of readers.

Good design simply makes the best of the material, for the benefit of the reader's comprehension, not for the creators sense of artistic achievement.

After due account has been taken of the readers visual conditioning, and a reports likely contents, it must still be remembered that every report deserves an unmistakable character of its own. Built up over time, a consistent style reflects stability which in turn implies authority, an essential element in effective communication. Good design is applied consciously to that end. Therefore it is a good idea to formulate some company standards concerning report layout, typefaces to be used, etc.

If a design scheme for a report is conceived as a means of containing a number of different categories within the report, but consistent in style, then there should be little problem in expanding it to cater for the unusual or the unexpected. The danger lies in believing that a report's design is a complete and absolute thing in itself rather than a flexible continuing process. Essentially, design is the function of reconciling necessary differences without showing how the trick was done, or even that any effort had to be made. Good design should be invisible.

SOFTWARE REQUIRED TO DRIVE A LASERJET

The software required to allow us to achive the quality and style of reports we want, needs to be able to satisfy certain criteria. It must be able to cope with output from any of the multitude of software packages on the market today. It must be able to access and drive all the facilities of a LaserJet, including the use of internal fonts, cartridge fonts and any soft fonts which the user may have (that includes being able to print any number and mix of fonts on a page of course). The software must also be able to merge output from more than one software package on to the same report.

It should be able to pick up existing reports and format them without application or program changes (even $STDLISTs on an HP3000 preferably).

When you are printing reports from computers such as the HP3000, it is very important that the software is fast and drives the LaserJet at full speed (8 pages per minuteon standard LaserJets, 20 pages per minute on LaserJet 2000s). It should support multiple spooled printers concurrently and preferably not require extra memory in the printer (if the software is any good you should not need more than the standard memory to do all kinds of forms, including charting). To maximize throughput it helps if the software tracks forms and fonts that are downloaded so that repeated downloading is avoided.

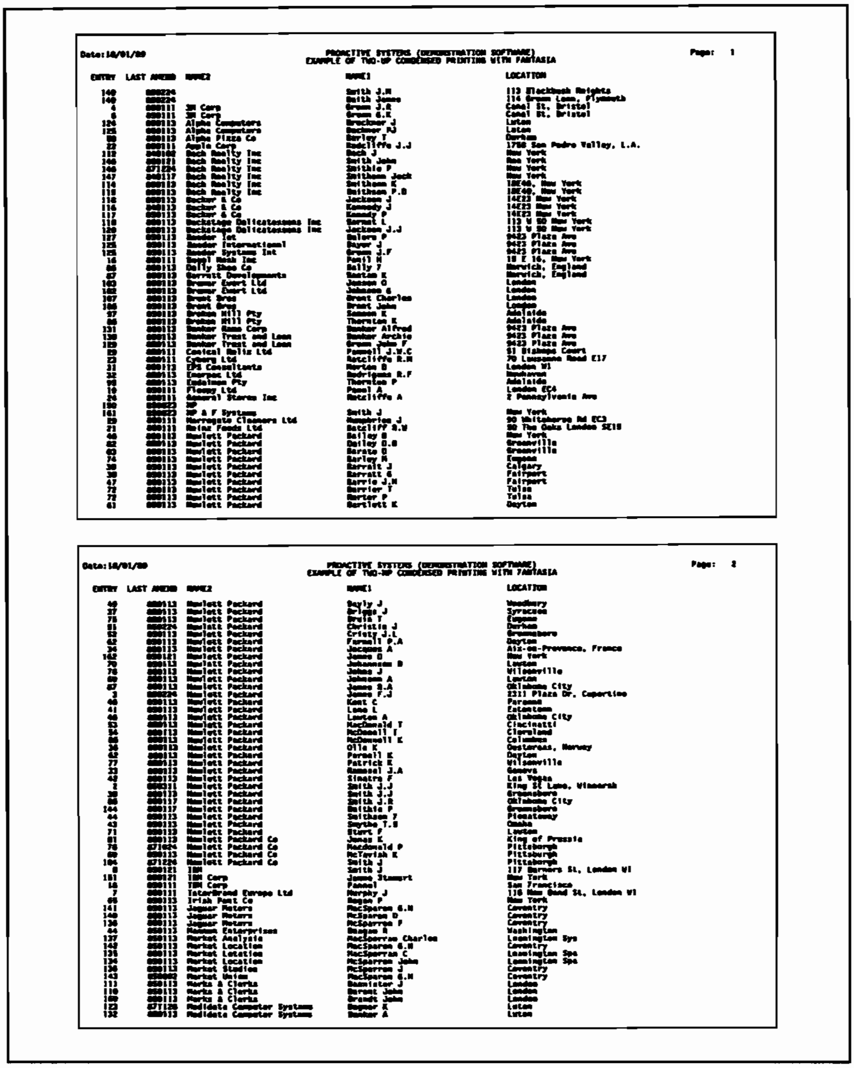
The software needs to be capable of sophisticated typesetting, including justification to both margins with automatic word wrap and hyphenation. It should easily handle columnar data where each column requires different justification (eg. some left, some right). And it should include charting capabilities so you can easily mix graphics and data.

Also in many applications it is necessary to have the layout change dynamically, ie. under program control, from page to page so support of this kind of need is advantageous.

Finally the user must gain something from using the software. The improvements in the appearance of reports are obviously desirable but they do not necessarily give a business case for the expense of the software - it depends on your view of our much a higher quality appearance and the clearer communication of information is worth, and that can vary a great deal from company to company. It may therefore be necessary to look for cost savings gained by using the software, and the most common of these is paper saving (obviously if you can also use the same software to print forms then there are other potential savings from that also). Take the following audit report example (See Figure 4). The software has taken a two page audit report, reduced each page in size, and then printed the two logical pages on one sheet of paper; thereby halving the amount of paper used. This idea could be taken further if the report was printed on the "duplex" LaserJet IID; it would be possible to print four logical pages on a single sheet of paper (or even 8 pages on one sheet if you have room to print 2-across).

I hope that I have shown that LaserJets can be used to enhance all computer printed reports, not just forms. After all what is a form? Simply some text and data with a few demarcation lines on it. Therefore the integration of forms printing with the other facilities mentioned herein gives an ideal solution.

Figure 4



Date:18/01/88 PROACTIVE SYSTEMS (DEMONSTRATION SOFTWARE) Page: 1
EXAMPLE OF TWO-UP CONDENSED PRINTING WITH FANTASIA

ENTRY	LAST AMEND	NAME2	NAME1	LOCATION
140	880224		Smith J.H	113 Blackbush Heights
140	880224		Smith James	114 Green Lane, Plymouth
4	880111	3M Corp	Green J.R	Canal St, Bristol
6	880111	3M Corp	Green G.K	Canal St, Bristol
124	880113	Alpha Computers	Bruckner J	Luton
125	880113	Alpha Computers	Buckner PJ	Luton
88	880113	Alpha Pizza Co	Barley T	Durham
22	880111	Apple Corp	Radcliffe J.J	1780 San Pedro Valley, L.A.
112	880102	Bach Realty Inc	Bach J	New York
146	880121	Bach Realty Inc	Smith John	New York
146	871224	Bach Realty Inc	Smithie P	New York
147	880117	Bach Realty Inc	Smithson Jack	New York
114	880113	Bach Realty Inc	Smithson K	18E40, New York
115	880113	Bach Realty Inc	Smithson P.B	18E40, New York
118	880113	Backer & Co	Jackson J	14E23 New York
116	880113	Backer & Co	Kennedy J	14E23 New York
117	880113	Backer & Co	Kennedy P	14E23 New York
118	880113	Backstage Delicatessens Inc	Barnet L	113 W 50 New York
120	880113	Backstage Delicatessens Inc	Jackson J.J	113 W 50 New York
127	880113	Bander Int	Balero P	9423 Plaza Ave
125	880113	Bander International	Bayer J	9423 Plaza Ave
126	880113	Bander Systems Int	Green J.F	9423 Plaza Ave
16	880111	Bagel Nosh Inc	Penil H	18 E 16, New York
86	880113	Bally Shoe Co	Bally Y	Norwich, England
87	880113	Barratt Developments	Banton E	Norwich, England
103	880113	Bramer Evert Ltd	Jonson O	London
102	880113	Bramer Evert Ltd	Johnson G	London
107	880113	Brent Bros	Brent Charles	London
106	880113	Brent Bros	Brent John	London
97	880113	Broken Hill Pty	Samson E	Adelaide
96	880113	Broken Hill Pty	Thornton K	Adelaide
131	880113	Bunker Rome Corp	Bunker Alfred	9423 Plaza Ave
130	880113	Bunker Trust and Loan	Bunker Archie	9423 Plaza Ave
129	880113	Bunker Trust and Loan	Green John F	9423 Plaza Ave
20	880111	Conical Holiz Ltd	Pannell J.W.C	81 Bishops Court
23	880111	Cyborg Ltd	Ratcliffe R.M	70 Lausanne Road E17
31	880113	EPS Consultants	Norton B	London W1
32	880113	Enerpac Ltd	Rodrigues R.F	[illegible]
98	880113	Endelman Pty	Thornton P	Adelaide
19	880111	Floppy Ltd	Panel A	London EC4
24	880111	General Stores Inc	Ratcliffe A	2 Pennsylvania Ave
150	880223	HP		
161	880223	HP & F Systems	Smith J	New York
20	880111	Harrogate Cleaners Ltd	Humphries J	90 Whitehorse Rd EC3
21	880111	Heinz Foods Ltd	Ratcliff R.W	80 The Oaks London SE18
46	880113	Hewlett Packard	Bailey B	New York
82	880113	Hewlett Packard	Dailey D.B	Greenville
83	880113	Hewlett Packard	Barato B	Greenville
74	880113	Hewlett Packard	Barley M	Eugene
30	880113	Hewlett Packard	Barrait J	Calgary
39	880113	Hewlett Packard	Barratt G	Fairport
47	880113	Hewlett Packard	Barrie J.M	Fairport
72	880113	Hewlett Packard	Barrier T	Tulsa
72	880113	Hewlett Packard	Barter P	Tulsa
61	880113	Hewlett Packard	Bartlett K	Dayton

Date:18/01/88 PROACTIVE SYSTEMS (DEMONSTRATION SOFTWARE) Page: 2
EXAMPLE OF TWO-UP CONDENSED PRINTING WITH FANTASIA

ENTRY	LAST AMEND	NAME2	NAME1	LOCATION
40	880113	Hewlett Packard	Bayly J	Woodbury
37	880113	Hewlett Packard	Briggs J	Syracuse
75	880113	Hewlett Packard	Bruin T	Eugene
51	880224	Hewlett Packard	Christie J	Durham
52	880113	Hewlett Packard	Cristy J.L	Greensboro
62	880113	Hewlett Packard	Farnell P.A	Dayton
34	880113	Hewlett Packard	Jacques A	Aix-en-Provence, France
162	880121	Hewlett Packard	James D	New York
70	880113	Hewlett Packard	Johansson B	Lawton
76	880113	Hewlett Packard	Johns J	Wilsonville
80	880113	Hewlett Packard	Johnson A	Lawton
87	880113	Hewlett Packard	James B.A	Oklahoma City
3	880224	Hewlett Packard	James F.J	2311 Plaza Dr, Cupertino
46	880113	Hewlett Packard	Kent C	Paramus
41	880113	Hewlett Packard	Lane L	Eatontown
48	880113	Hewlett Packard	Lawton A	Oklahoma City
53	880113	Hewlett Packard	MacDonald T	Cincinatti
54	880113	Hewlett Packard	McDonell T	Cleveland
86	880113	Hewlett Packard	McDonnell K	Columbus
36	880113	Hewlett Packard	Olla K	Oesteraas, Norway
42	880113	Hewlett Packard	Pernell K	Dayton
77	880113	Hewlett Packard	Patrick K	Wilsonville
33	880113	Hewlett Packard	Ramsel J.A	Geneva
49	880113	Hewlett Packard	Sinatra F	Las Vegas
2	880311	Hewlett Packard	Smith J.J	King St Lane, Winnersh
38	880113	Hewlett Packard	Smith J.J	Greensboro
85	880117	Hewlett Packard	Smith J.R	Oklahoma City
144	880117	Hewlett Packard	Smithie P	Greensboro
44	880123	Hewlett Packard	Smithson Y	Piscataway
43	880113	Hewlett Packard	Smythe T.B	Omaha
71	880113	Hewlett Packard	Start F	Lawton
81	880113	Hewlett Packard Co	Jones K	King of Prussia
78	871024	Hewlett Packard Co	Macdonald P	Pittsburgh
80	880113	Hewlett Packard Co	McTavish K	Pittsburgh
104	871224	Hewlett Packard Co	Smith J	Pittsburgh
8	880121	IBM	Smith J	117 Berners St, London W1
151	880121	IBM Corp	James Stewart	New York
16	880111	IBM Corp	Pannel	San Francisco
7	880111	InterBrand Europe Ltd	Murphy J	118 New Bond St, London W1
65	880113	Irish Pent Co	Regan P	New York
141	880113	Jaguar Motors	MacSparen G.H	Coventry
140	880113	Jaguar Motors	McSparen D	Coventry
138	880113	Jaguar Motors	McSparren F	Coventry
44	880113	Mannen Enterprises	Bragan R	Washington
137	880113	Market Analysis	MacSparran Charles	Leamington Spa
142	880113	Market Location	MacSparen G.H	Coventry
135	880113	Market Location	MacSparran C	Leamington Spa
134	880113	Market Location	McSparren John	Leamington Spa
136	880113	Market Studies	McSparren J	Coventry
143	880002	Market Union	MacSparen G.H	Coventry
111	880113	Marks & Clerks	Bannister J	London
110	880113	Marks & Clerks	Barent John	London
109	880113	Marks & Clerks	Brandt John	London
123	871128	Medidata Computer Systems	Wagner K	Luton
132	880113	Medidata Computer Systems	Bunker A	Luton

Improving Program Reusability

Larry Van Sickle
Tymlabs Corporation
811 Barton Springs Road
Austin, Texas USA 78704
Voice: 512-478-0611
Fax: 512-479-0735
Telex: 755.820
Email: lvs@cs.utexas.edu

Continuous improvement

The purpose of this paper is to focus attention on reuse of software as a valuable technique for improving developers' productivity, to provide a general framework for understanding the problems involved in reusing software, and to provide ideas for some immediate steps that developers can take to reuse software. The best way to improve productivity is to start a process of incremental improvements in the way we write and reuse code. A continuous process of incremental improvement, rather than adoption of any radically new methodology, will lead to improved productivity reliably, cheaply, and without disruption. The Japanese approach of constantly and repeatedly finding a 2% or 3% improvement in efficiency or performance has proven to be very successful. The compounding effect of 2% or 3% improvements over time is enormous.

This paper describes techniques that can be used to achieve those incremental improvements and describes possible future methods and tools for reusing software.

Software Reuse

Fourth-generation languages, application generators, CASE, and object-oriented programming have all been used as effective techniques for improving programmer productivity. While providing large productivity gains, many of these products require retraining or adoption of new methodology.

One of the most promising techniques for improving software productivity is reuse of existing code. Reuse can be adopted by an organization with no retraining, and with no major change in methodology. Reuse is a natural extension of code sharing and maintenance techniques already used in the organization.

Program reuse is additive. Whatever techniques you are using, reuse can improve them. If you use CASE, you may be able to reuse existing designs. If you use a 4GL, you may be able to reuse an existing program.

There is already a modest industry in reusable software components. Open any issue of Computer Language or Dr. Dobb's Journal and you will see advertisements for libraries for windowing, b-tree file access, virtual memory, printer drivers, and much more. Most of these libraries are written in C and designed to work with C programs.

Reuse improves productivity two ways. First, it reduces the amount of new code that must be written. Second, it reduces the chances for introducing errors into the code.

Technology needed for reuse

For reuse to improve productivity, it must be less effort to reuse code than to write new code. In most shops today, there is no mechanism for sharing, cataloging, and reusing code. Individual developers often reuse pieces of their own code, but there is very little sharing or reuse of code among groups of developers. Today it is usually easier to write a new piece of code than to reuse existing code.

Reusing code requires three steps:

1. Locating the appropriate code.
2. Understanding the code.
3. Adapting the code to its new use.

Locating the appropriate code requires a method for describing a piece of code, describing the problem we are trying to solve, and matching the problem to the pieces of code. This requires development of a language or classification scheme for describing code.

Classification schemes for software

Several classification schemes exist for computer science. ACM Computing Reviews has a hierarchical classification scheme. The AFIPS Taxonomy of Computer Science and Engineering is another. Commercial software catalogs, such as those from large mail-order software houses, have simple classification schemes, usually a few general classes with long listings of programs in each class. Buyers must read detailed descriptions of products to locate one that may meet their specific needs.

Any useful system for software reuse will need a very flexible representation and search scheme. A collection of reusable components must be large to be useful, and will be continuously expanding.

Instead of a hierarchical classification, an automated library of software components will probably use a multi-dimensional scheme. For each dimension there will be a defined vocabulary of terms.

Research in classification schemes suggests that a three-dimensional scheme will probably be most useful. The three dimensions are action, object, and structure. Examples of action are add, append, close, compress, encode, and move. Examples of object are array, character, digit, directory, expression, function, line, list, macro, page. Examples of structure are array, buffer, disk, file, keyboard, line, mouse, printer, screen, sensor, table, tape, tree. A module might be classified, for example, as <input,character,keyboard>, or <substitute,tabs,line>.

Object-oriented programming

Object-oriented programming is a programming methodology that promotes reuse of software components. Object-oriented programming was not developed solely to improve reuse, but that is one of its goals. Object-oriented programming is based on two central principles: *encapsulation* and *inheritance.*

Encapsulation requires that a data structure or software component have an abstract external interface that contains no assumptions about the implementation. Data structures are associated with a set of routines for accessing and updating the data structure. All accesses and updates are done through the routines and only through the routines. Therefore design changes to the data structure can be made without changing code that uses the data structure. Only the data structure definition and the routines need to be changed. The encapsulated data structures are called *objects.*

Inheritance is a form of reuse. An object can inherit, or reuse, characteristics and routines from another object.

For example, define a COMPANY data structure that contains all the attributes of a company: name, street address, city, state or political subdivision, country, postal code, voice telephone number, fax telephone number, and so forth. Next define all the code to validate the values of attributes. Now to define a CUSTOMER, we start with the COMPANY data structure and add attributes of customer account number, credit rating, shipping address. We also add code to validate the new attributes. To define a SUPPLIER, we also start with COMPANY and add attributes for vendor number, our customer number, and discount.

In object-oriented terms, COMPANY would be a *class.* CUSTOMER and SUPPLIER are *subclasses* of COMPANY, and they inherit all the attributes and code for COMPANY. In everyday terms, CUSTOMER and SUPPLIER reuse the code of COMPANY. An object-oriented programming language gives a formal syntax and mechanism for reusing code via inheritance.

Encapsulation in COBOL

Data structures can be encapsulated in COBOL by using *nondynamic subprograms* and multiple entry points. Non dynamic subprograms maintain the values of variables in their WORKING-STORAGE SECTION between calls. You can create a nondynamic subprogram by using either the SUBPROGRAM or ANSISUB in the $CONTROL line, or by including a LINKAGE SECTION without using DYNAMIC in the $CONTROL line.

The data structure is defined in the WORKING-STORAGE SECTION of the COBOL program. The different entry points each represent an operation that can be performed on the data structure. Calling programs cannot access or modify the data structure except through the defined entry points.

For example, many applications store codes in records. The codes might represent a type of customer, a sales region, or any other piece of information with a limited number of possible values. The users want to see the interpretation of the code, not the code itself. The application program has to keep a translation table giving the interpretation for each code value. In many cases the codes and their interpretations are stored in a file, rather than hard coded in a program, to make them easy to maintain. When the application program is executed, the file is loaded into a table which is used throughout that run of the program.

The table is an excellent example of a data structure that should be encapsulated. Many different programs use the table. If they all access the table only through a well-defined external interface, then the table structure can be changed without changing the calling programs. Figure 1 shows a COBOL implementation of the table.

The developer could completely change the implementation of the table without affecting any calling program. The table could be sorted and a binary search used to lookup a key, or the table could be stored in a data base instead of a file. Neither of these changes would be apparent outside of the encapsulation program.

Three levels of reuse

We can identify three levels of reuse of software. The first level is use of code skeletons that contain the common parts of a program. For a COBOL program this would be a code skeleton that had the IDENTIFICATION DIVISION, WORKING-STORAGE SECTION, and all the other standard parts of a COBOL program, in the right order. In a C program, this would be a skeleton that contained the `#includes`, the declaration of the `main()`, declaration of `argc` and `argv`, and other standard parts. This type of reuse keeps programmers from having to remember and create the basic structure of the program. Instead, they simply add their code to the skeleton. Every programmer reuses code at this level, and this saves a lot of time. Most COBOL programmers have never written a COBOL program from scratch, they have always picked up an existing program and modified it, although sometimes all they save is the basic COBOL structure.

The second level is the use of standard components in the form of subroutine libraries, standard pieces of code, or standard specialized code skeletons. This is a level that can be achieved today with existing tools. Much of the work will be manual, and discipline is required to keep the libraries updated and consistent.

The third level is automated management of code libraries. This level of reuse will require development of automated tools that document, restructure, and classify existing code; that search libraries and match requirements against code descriptions; and that aid programmers in understanding and transforming existing code.

Conclusion

Reuse is a valuable method for increasing developers' productivity. Most developers practice limited forms of reuse already, but there is enormous room for improvement. Developers can apply the ideas of encapsulation and inheritance from object-oriented programming to COBOL and other languages to improve reusability of code. Developers can also begin to build libraries of reusable components with associated descriptions and indexing. The ideal of building software by assembling existing components awaits the development of automated methods of documenting, classifying, matching, and modifying existing code.

```
$CONTROL SUBPROGRAM
 IDENTIFICATION DIVISION.
 PROGRAM-ID. TABLE-HANDLER.
 AUTHOR. LARRY VAN SICKLE.
 ENVIRONMENT DIVISION.

 CONFIGURATION SECTION.
 SOURCE-COMPUTER. HP-3000.
 OBJECT-COMPUTER. HP-3000.

 INPUT-OUTPUT SECTION.
 FILE-CONTROL.

     SELECT TABLE-FILE  ASSIGN TO "TABLE1"
       ORGANIZATION IS INDEXED ACCESS MODE IS DYNAMIC
       RECORD KEY IS TABLE-REC-KEY.

 DATA DIVISION.
 FILE SECTION.

 FD  TABLE-FILE
     RECORD CONTAINS 128 CHARACTERS
     DATA RECORDS ARE TABLE-REC.

 01  TABLE-REC.
     02  TABLE-REC-KEY                PIC X(20).
     02  FILLER                       PIC X(108).

 WORKING-STORAGE SECTION.

 01 TABLE-AREA.
    05 TABLE-ENTRY                  OCCURS 100 TIMES.
       10 TABLE-KEY                 PIC X(20).
       10 TABLE-DATA                PIC X(108).
 01 TABLE-SIZE                      PIC S9(4) COMP VALUE 100.
 01 TABLE-MAX                       PIC S9(4) COMP VALUE 100.
 01 TABLE-FILE-STATUS               PIC XX.
    88 TABLE-FILE-EOF               VALUE "10".
    88 TABLE-FILE-OK                VALUE "00".
 01 I                               PIC S9(4) COMP.
 01 NO-MATCH                        PIC S9(4) COMP VALUE 1.
 01 MATCH                           PIC S9(4) COMP VALUE 0.

 LINKAGE SECTION.

 01 KEY-PARM                        PIC X(20).
 01 DATA-PARM                       PIC X(108).
 01 LOOKUP-STATUS                   PIC S9(4) COMP.
    88 KEY-MATCHED                  VALUE 0.

 PROCEDURE DIVISION.
 MAINLINE SECTION 10.

     STOP RUN.

*************************************************************
*   1-LOAD-TABLE
*************************************************************

 1-LOAD-TABLE.

 ENTRY "TABLELOAD".

 1-OPEN-TABLE-FILE.

     OPEN INPUT TABLE-FILE.
     IF TABLE-FILE-STATUS NOT = "00"
       MOVE 1 TO LOOKUP-STATUS
       GOBACK
     .
     PERFORM 1-READ-TABLE-FILE VARYING I FROM 1 BY 1 UNTIL
       I > TABLE-MAX OR TABLE-FILE-EOF.
     GOBACK.

 1-READ-TABLE-FILE.

     READ TABLE-FILE INTO TABLE-ENTRY(I)
       AT END
         SUBTRACT 1 FROM I GIVING TABLE-SIZE


*************************************************************
*   1-LOOKUP-TABLE
*************************************************************

 2-LOOKUP-TABLE.

 ENTRY "TABLELOOKUP" USING KEY-PARM DATA-PARM LOOKUP-STATUS.

 2-SEARCH-TABLE.

     MOVE NO-MATCH TO LOOKUP-STATUS.
     PERFORM 2-MATCH-TABLE-ENTRY VARYING I FROM 1 BY 1 UNTIL
       I > TABLE-SIZE OR KEY-MATCHED.
     GOBACK.

 2-MATCH-TABLE-ENTRY.

     IF TABLE-KEY(I) = KEY-PARM
       MOVE MATCH TO LOOKUP-STATUS
       MOVE TABLE-DATA(I) TO DATA-PARM
```

Figure 1. Example of Encapsulation in COBOL

```
$CONTROL USLINIT,SOURCE,WARN,MAP
 IDENTIFICATION DIVISION.
 PROGRAM-ID. id.
 AUTHOR. author.
 ENVIRONMENT DIVISION.

 CONFIGURATION SECTION.
 SOURCE-COMPUTER. HP-3000.
 OBJECT-COMPUTER. HP-3000.

 INPUT-OUTPUT SECTION.
 FILE-CONTROL.

 SELECT  file
         ASSIGN TO "mpename"
         ORGANIZATION IS INDEXED
         ACCESS MODE IS DYNAMIC
         RECORD KEY IS file-KEY
         FILE STATUS IS file-FILE-STATUS.

 DATA DIVISION.
 FILE SECTION.

 FD  file
     RECORD CONTAINS recordlen CHARACTERS
     DATA RECORDS ARE file-REC.
 01  file-REC.
     02  field                        PIC pic.

 WORKING-STORAGE SECTION.

 01 file-KEY                          PIC pic.

 PROCEDURE DIVISION.

 DECLARATIVES.

 IO-FILE-ERR SECTION 10.
     USE AFTER STANDARD ERROR PROCEDURE ON I-O.
 REPORT-ERR-PARA.
     EXIT.
 END DECLARATIVES.

 MAINLINE SECTION 10.
```

Figure 2. Example of COBOL skeleton

IMPROVING REMOTE DATA BASE ACCESS PERFORMANCE

Roger Lawson
Proactive Systems Inc.
339 South San Antonio
Los Altos
CA 94022
Tel: (415)-941-9316

ABSTRACT

One of the problems when building distributed systems based on IMAGE using HP3000 computers, is the slow response time and low throughput apparent when accessing remote data bases over NS. The author will describe various techniques for overcoming this problem so that it becomes practical to build high performance distributed applications.

WHY PEOPLE BUILD DISTRIBUTED SYSTEMS

Distributed systems can meet the needs of many organisations for more flexible and lower cost computer applications.

Distributed systems simply use multiple computers to disperse processing in a network instead of using one centralized computer. They range from the extremes of two computers placed back to back to large networks of linked computers. The benefits of distributed systems are:

■ You can minimize computer hardware and data communication cost. Multiple smaller computers give you more effective power for your dollar than one large computer.

■ You can minimize data communication costs. Doing local processing minimizes data transmission needs.

■ Increased data redundancy can provide 24-hour uptime, good disaster protection (you don't have all your eggs in one basket) and better user response times.

■ Geographically dispersed organizations can obtain local processing and local management control.

■ Computing capacity growth can be more incremental - simply add another computer to expand your processing capability.

■ There is more flexibility in hardware selection, computer placement, management structures, application design, etc.

However if you want to build such distributed systems on HP3000 computers using NS communication links and IMAGE data bases, you are likely to meet some performance bottlenecks unless you are experienced in building such systems. For example, doing a remote log-on to a system, opening a data base and retrieving records will all take much longer over an NS link than on a local system, as anyone who has used NS much will tell you. But you can easily avoid these performance problems by careful design and by the use of a few technical "tricks".

DATA LOCATION TRANSPARENCY

Once you spread your data bases over multiple computers, you need to be able to easily access data on remote nodes with no more difficulty than if it was on the local node. This can be summarised by the phrase "data location transparency". Users or application programs should also not need to have to worry where the data is located. See Figure 1.

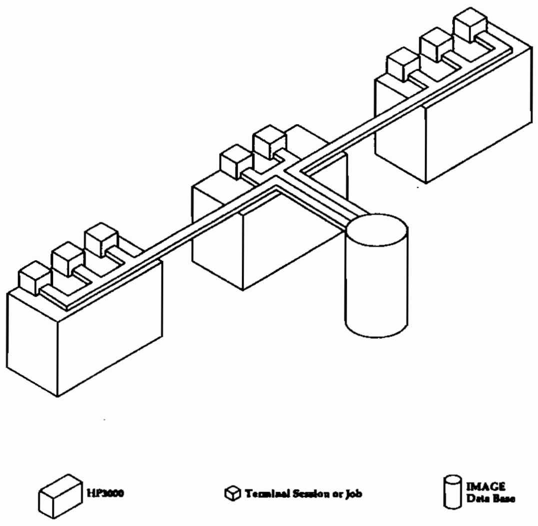


Figure 1 - Data Location Transparency Example

POOR PERFORMANCE AND COSTLY DATA COMMUNICATIONS

All of you who have used NS or DS know that the speed of access to remote data bases, both in terms of response times and data throughput, can be apparently much worse than local access. If you have a lot of user sessions accessing remote IMAGE data bases, your whole system can grind to a halt.

Remote data base access using standard NS facilities not only uses more cpu processing capacity but also generates a lot of data communications traffic - this by its very nature costs more than a local disc access. Data communication costs can be a significant proportion of overall project costs when building distributed systems. Overcoming these problems is what this paper is about.

THE SOLUTIONS

■DATA REPLICATION

Maintaining multiple copies of data can do wonders for performance. Having a local copy of a data base can be much more cost effective if the ratio of read to update access is high, as is common in many commercial systems.

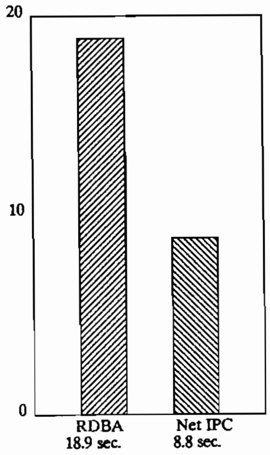
For example reference information such as price lists are best held on every node. The amount of disc space needed to do this may be small and it reduces dependance on one central copy. Any updates (which are typically low volume to such files) can be automatically replicated from the central site - possibly in pseudo batch mode if you don't need real time updating and prefer not to keep the data comm link open.

Data replication also enables you to maximize performance by using load spreading. For example you could off-load all enquiry and reporting activity to a second copy and thus improve update response on the primary. Obviously data replication uses extra disc space so there is a trade off that must be studied in each situation between the cost of additional disc space against the data communication and cpu processing cost. However disc storage costs are certainly falling more rapidly than data communication costs.

■ SOFTWARE

Another approach is to tackle the performance problem at a technical software level. Two particular performance bottlenecks that can be solved on IMAGE data bases are:

1. It takes a long time using RDBA to open a communication link and open a data base in response to ad-hoc enquiry or update activity. The solution is to hold a pipeline open to the remote data node so that response can be immediate (or at least comparable to local data base open time). Figure 2 shows some comparative figures I obtained comparing the two methods on our development systems.



NetIPC versus RDBA

Elapsed time in seconds to establish remote session, open data base, read three records from master set, read one detail set record and close data base. Average of twelve runs accessing MPE/V system from MPE/XL system.

Figure 2

The pipeline can be shared by any number of users and can simply be created when the first user connects.

2. Poor throughput or high response times can be caused due to the large amount of data transmitted by RDBA over communication links which are always relatively slow in comparison to a local access path. For example RDBA transmits the whole data base buffer which is created on a remote node. However typically you don't need all the data items in all the records. You probably only want some of the data items from some of the records. Solution: do the processing on the remote node (where it is local to the data base) and only transmit the data required by the user program. Figure 3 shows some comparative figures I obtained.

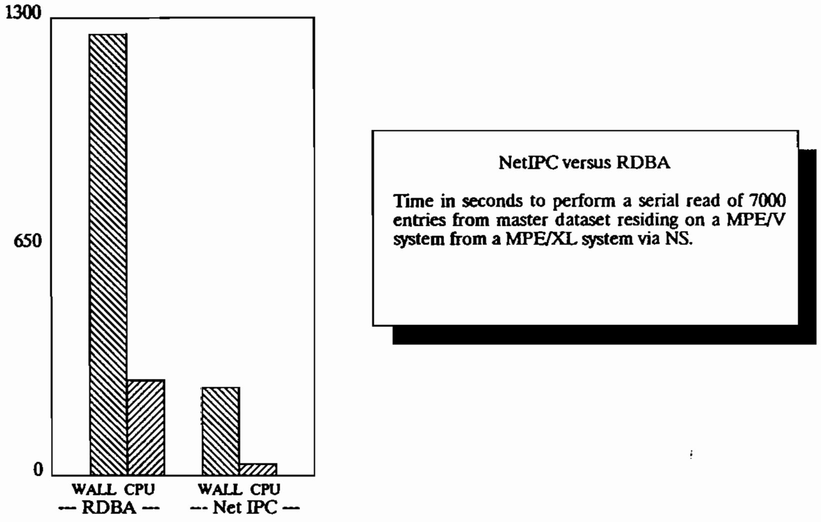


Figure 3

Both of the above techniques can be implemented in a generalized manner and we did this using the NetIPC service of NS, ie. you don't have to change the application software to get these kind of performance improvements. By using NetIPC and appropriate buffering strategies you can effectively optimize what you get transmitted over a network which is the key to performance maximization. Making the best use of each NS transaction is the key!

■ SYSTEM DESIGN

You should process the data where most of it is located so as to minimise the amount of data transmission over the network. Consider the following example: you have two computers named A and B with the users logged onto A but most of the data they access on B - they want to run a report which simply wades through the data base to produce a total. It is much more efficient to run the reporting program on computer B and route a few report lines back to A, than run the program on computer A in which case every data base I/O goes through the computer link.

You could use remote process handling with the pipeline architecture mentioned above to achieve this kind of result transparently to the user.

Using the above data replication and software optimization techniques can reduce the data transmitted and hence the cost by a factor of up to 10. Also careful consideration to optimal data communication configuration (which is outside the scope of this paper) also pays good dividends.

As always, careful application design is also a key factor in minimising data traffic and maximising performance.

With distributed systems you need to spend more time on careful design of the application system because every I/O in a network "costs" a lot more than the same local I/O.

CONCLUSION

To summarise, building high performance distributed systems is possible on HP3000s. Using the techniques described above you can construct applications such that overall costs are lower than they would be in a centralised system, without having to accept lower response times.

Integer Keys: The Final Chapter

Fred White

Adager

Sun Valley, Idaho
83353-0030
U.S.A.

(208) 726-9100

Introduction

The calculation of primary addresses for IMAGE keys of data types X, U, P and Z is performed by a hashing algorithm whose goal is to generate a uniform distribution of primary addresses on the closed interval [1,C] where C is the capacity of the master dataset.

Despite what you may have read or heard from various IMAGE evangelists, this is not true for keys of data types I, J, K and R. Keys of these types are called "non-hashing" keys for the simple reason that they are NOT hashed! IMAGE makes NO attempt to distribute them uniformly! The user has ABSOLUTE control over their primary address assignment! This control is exercised by the user's method of assigning key values and his choice of master dataset capacity.

There are two kinds of non-hashing keys: "type R" and "types I, J and K". I shall refer to key types I, J and K as "integer keys".

With the proper tools and knowledge, integer-keyed master datasets can be created so that they have no synonyms and are not wasteful of disc space. The ill-advised use of integer keys typically leads to performance disasters!

History

In January of 1972, the IMAGE/3000 project team agreed to provide for non-hashed master datasets in which the primary address calculation would be in the hands of the user rather than controlled by IMAGE's hashing algorithm.

After considering various options, we decided on the following:

1. IMAGE keys of types I, J, K and R would NOT be hashed.
2. These keys could be of any length acceptable to IMAGE.
3. Only the rightmost 31 bits (the "determinant") would be used to calculate the primary address. (For one-word keys, their 16 bits are padded on the left with zeroes.)
4. The determinant is then divided by the dataset capacity yielding a remainder which becomes the primary address unless it is zero, in which case the capacity is assigned as the primary address.

Notice that, for a given capacity C, if we use determinant values N between 1 and C, the primary address for each N is N.

Furthermore, if these determinant values are all unique, the user will have taken advantage of IMAGE's integer key facility to provide himself with the good old, traditional, Direct Access Method (DAM).

However, IMAGE does not demand uniqueness of determinants nor does it restrict their values to the range 1 to C. We shall see that this "loosening up" of the constraints on the values of N, if used, will typically lead to a horrible performance problem unless the user is armed with a tool for intelligently selecting a capacity which will enable him to avoid such a performance pitfall.

Non-hashing Key Performance Pitfalls

In this section I address two distinct examples of bad uses of IMAGE's integer key facility. Both of these appeared in an earlier paper of mine "The Use and Abuse of Integer Keys".

The first example demonstrates that our choice to not hash keys of IMAGE type R was a horrible design decision.

Example 1: The Synonym Pitfall

This "pitfall" arises whenever a user elects to use a key of IMAGE type R4 whose key values are, for the most part, integers.

To understand why, one must be knowledgeable about the format of 64-bit reals as represented on the HP3000 family of computers.

The leftmost bit is the sign bit, the next 9 bits are the exponent, and the rightmost 54 bits constitute the mantissa (excluding the most significant bit).

As a consequence, the floating point format of all integers of magnitude less than

8388609 (2**23+1) is such that the low order 31 bits are all zeroes. All entries with keys like this will be in a single synonym chain having the dataset capacity as its primary address!

To add a new entry to this chain, DBPUT must traverse the entire synonym chain to ensure that the key value of the new entry is not a duplicate before adding it to the chain. This has an impact on performance proportional to the number of entries in the chain (which could be in the thousands) and inversely proportional to the blocking factor.

Also, each DBFIND (or mode 7 DBGET) will, on average, be forced to traverse half of the chain to locate the desired entry!

The picture improves somewhat if an R2 field is used. In this case, the rightmost 31 bits (which are reduced modulo the capacity) include the exponent bits and all bits of the mantissa.

Consequently, the various key values will not all be assigned the same primary address. However, if these values have only a few significant binary digits in their mantissa, the rightmost bits will tend to be all zeroes which will lead to a high percentage of synonyms regardless of the capacity. This is especially true if the capacity is a power of 2 because we are treating the R2 field as a double integer and, if several key values have zeroes in their rightmost N bits, they are all divisible by 2**N and thus will all be synonyms.

For either R2 or R4 keys there will always be significant synonym problems unless the rightmost 31 bits of the keys, in and of themselves, form a set of doubleword values which represent a uniform distribution over the closed interval [1,2**31-1]. In this event, the primary addresses assigned will tend to be uniformly distributed over the master dataset even though no hashing occurs.

The bottom line is: if your R2 or R4 values don't fit this pattern, avoid using them as keys.

When Hewlett-Packard finally introduces an IEEE real data type in IMAGE, similar warnings will apply since the data formats differ from the Classic 3000 reals only in the number of bits in the exponents. The 32-bit IEEE real has an 8-bit exponent, the 64-bit IEEE real has an 11-bit exponent and the 128-bit IEEE real has a 15-bit exponent.

The next example demonstrates the problems which can arise when using integer keys (IMAGE types I, J or K).

Example 2: The Primary Clustering Pitfall

One Friday in 1978 I received a phone call from an insurance firm in the San Francisco Bay Area. I was told that their claims application was having serious performance problems and that, in an attempt to improve the situation, they had, on the previous Friday, performed a DBUNLOAD, changed some capacities and then started a DBLOAD which did not conclude until the early hours of Tuesday morning!

They were a US$100 million-plus company which couldn't stand the on-line response they were getting and couldn't afford losing another Monday in another vain attempt to resolve their problems.

Investigation revealed that claims information was stored in two detail datasets with paths to a shared automatic master. The search field for these three datasets was a double integer key (IMAGE type I2) whose values were all of the form YYXXXXX (shown in decimal) where YY was the two-digit representation of the year (beginning with 71) and where, during each year, XXXXX took on the values 00001, 00002, etc. up to 30000.

Although the application was built on IMAGE in late 1976, earlier claims information (from 1971 through 1976) was included to be available for current access. I do not recall the exact capacity of the master dataset but, for purposes of displaying the nature of the problem (especially the fact that it didn't surface until 1978) I will assume a capacity of 370000.

Although the number of claims per year varied, the illustration will also assume that each year had 30000.

The first claim of 1971 was claim number 7100001 to which (using a capacity of 370000) IMAGE would assign a primary address of 70001. This is because 7100001 is congruent to 70001 modulo 370000.

The 30000 claims of 1971 were thus assigned the successive record numbers 70001 through 100000 (a cluster of primaries).

Similar calculations show that the claims for each year were stored in clusters of successive addresses as follows:

Claim Numbers	Record Numbers
7100001 through 7130000	70001 through 100000
7200001 through 7230000	170001 through 200000
7300001 through 7330000	270001 through 300000
7400001 through 7430000	1 through 30000
7500001 through 7530000	100001 through 130000
7600001 through 7630000	200001 through 230000
7700001 through 7730000	300001 through 330000

Note that no two records had the same assigned address and thus that there were no synonyms and that all DBPUTs, DBFINDs and keyed DBGETs were very fast indeed!

Along came 1978!!!

Unfortunately 7800001 is congruent to 70001 modulo 370000 so that the first DBPUT for 1978 creates the very first synonym of the dataset. Claim 7800001 is, in fact, a synonym of claim 7100001.

DBPUT attempts to place this synonym in the block occupied by claim 7100001 but that block is full so DBPUT performs a serial search of the succeeding blocks to find an unused location. In this case, it searches the next 60000 records before it finds an unused address at location 130001! Even with a blocking factor of 50, this requires 1200 additional disc reads making each DBPUT approximately 200 times as slow as those of all previous years!!

Note that the next claim of 1978 (claim 7800002) is congruent to 70002 and is thus a synonym of claim 7100002. This also leads to a serial search which ends at location 130002! Thus the DBPUT of each claim for 1978 results in a search of 60000 records 59999 of which were inspected during the preceding DBPUT!!

Primary clustering had claimed another victim!! The designer of this system had unknowingly laid a trap which would snap at a mathematically predictable time, in this case 1978. After struggling with this problem for months, the user escaped the clustering pitfall by converting to "hashed keys" (in both the database and the software); a very expensive conversion!

Note that the problem was NOT a "synonym" problem in the sense that synonym chains were long nor was it a "fullness" problem since the master dataset was less than 57% full when disaster struck.

The problem was due to the fact that the records were severely clustered when the very first synonym occurred and DBPUT's serial space searching algorithm is efficient only in the absence of severe clustering.

It should be apparent by now that designers may avoid this cluster collision pitfall by carefully (mathematically) investigating the consequences of their assignment of integer key values together with their choice of master dataset capacity.

Look Mom, No Synonyms

As we have seen, the use of integer keys of IMAGE types I, J or K, coupled with the assignment of key values created by concatenating pairs (or even triplets) of integer subfields whose values are sequential, always leads to these clusters of primaries; a new cluster arising whenever a new value is assigned to any but the last subfield.

There are, however, many situations which lend themselves to the use of integer keys in this manner. In our example, the YY major values form the sequence 71, 72, 73, ... and the XXXXX minor values form the sequence 00001, 00002, 00003, ...

Notice that, for any particular value of YY, the primary addresses for keys with values YY00001, YY00002, ... form a set of (circularly) consecutive record numbers X, X+1, ... where X is the primary address generated by reducing YY00001 modulo the capacity C. In other words, they form a cluster of consecutive primaries. I will refer to such a cluster as a "run".

Notice that each increment of the minor value by 1 merely lengthens the run by 1 and that each increment of the major value marks the beginning of a new run with the minor value restarting at 1.

This works great (i.e., no synonyms) until a new run collides with a pre-existing run. When this happens, you have a performance disaster on your hands as shown in our example.

The question arises: "Is there a way to determine a capacity such that for a specified range of major, intermediate and minor values, the resulting dataset will have no run collisions (i.e., no synonyms) and yet not be unsatisfactorily wasteful of disc space?".

Some IMAGE evangelists have simplistically "answered" this question by recommending that you select a capacity equal to or greater than the largest key value. We shall refer to this technique as "Method 1".

Applying Method 1 to our 1978 example above would have yielded a capacity of at least 7830000 to hold 240000 entries (8 years worth). Unfortunately, the dataset would only be about 3.0% full which, because of its size, would be very wasteful of disc space.

A better "answer", which we shall refer to as "Method 2", is to choose a capacity 1 greater than the difference between the highest and lowest key values. In our example, this would equal 7830000-7100001+1 = 730000 and the dataset would be about 32.8% full. Not nearly as bad and yet not great either.

The question then arises: "Is there a way to calculate the smallest capacity which will yield no synonyms?".

To any mathematician worth his salt, the answer to this question is, "Hell, yes".

To prove this, note that we have already established that the set of all capacities which will yield datasets without synonyms is not empty. There is, in fact, an infinite number of answers satisfying Method 1.

Next, since capacities are always positive, the set of all successful capacities must have a smallest value since the set of positive integers is "well ordered" and bounded below by zero.

Lastly, since the method of key value assignment is well defined, as is the method of primary address calculation, all that remains to be done is to convert this knowledge to a programmable algorithm which will calculate this smallest capacity.

Clearly, we could simply start with the minimum possible capacity C which equals the product N x L, where N is the "number of runs" and L is the "run length". By calculating the first address of each run, we can determine whether any two runs collide. If they do, we can increment C and try again. This will ultimately yield a successful value of C which is, for multiple runs, generally far better than the capacity determined by Method 2.

Some months ago, after years of procrastination and wearying of the simplistic and inadequate advice being peddled by others, I finally developed and programmed an algorithm which converges on a minimum C value in a matter of seconds.

We refer to this algorithm as "Method 3".

If I had possessed Method 3 in 1978, I could have applied it to our earlier example of a "pitfall" to achieve a no-synonym dataset 90% full by DECREASING the capacity from 370000 to 265000 yielding the following runs:

Claim Numbers	Record Numbers
7100001 through 713000	210002 through 240001
7200001 through 723000	45002 through 75001
7300001 through 733000	145002 through 175001
7400001 through 743000	245002 through 10001
7500001 through 753000	80002 through 110001
7600001 through 763000	180002 through 210001
7700001 through 773000	15002 through 45001
7800001 through 783000	115002 through 145001

If the user wanted to maintain 15 years of claims, Method 3 yields a 79% full dataset with a capacity of 565000 and no synonyms:

Claim Numbers	Record Numbers
7100001 through 713000	320002 through 350001
7200001 through 723000	420002 through 450001
7300001 through 733000	520002 through 550001
7400001 through 743000	55002 through 85001
7500001 through 753000	155002 through 185001
7600001 through 763000	255002 through 285001
7700001 through 773000	355002 through 385001
7800001 through 783000	455002 through 485001
7900001 through 793000	555002 through 20001
8000001 through 803000	90002 through 120001
8100001 through 813000	190002 through 220001
8200001 through 823000	290002 through 320001
8300001 through 833000	390002 through 420001
8400001 through 843000	490002 through 520001
8500001 through 853000	25002 through 55001

By now it should be clear that, under the right circumstances and with the proper tools, integer keys are superior to hashing keys since we can guarantee a dataset with no synonyms.

Let's look at some other examples.

First, suppose we have an application where access is keyed on "day-of-year". We can define a master dataset with an I1 key and a capacity of 366 (remember leap year). If we subsequently reference all 366 days of the year, the master will be 100% full with no synonyms. In this simple case, all three methods yield the same result.

Suppose, however, we want to key on "day-of-month". In this case our I1 (or J1 or K1) key values (in decimal notation) could be in the form MMDD where 1<=MM<=12 and 1<=DD<=31. The smallest value represented will be 101 and the largest 1231.

Method 1 yields a capacity of 1231. Since the dataset has exactly 366 entries, it is thus only 29.73% full.

Applying Method 2, we achieve a capacity of 1231-101+1 = 1131 and a master which is 32.36% full.

Applying Method 3 yields a capacity of 431 and a master which is 84.91% full with no synonyms:

Key Values	Record Numbers
0101 through 0131	101 through 131
0201 through 0231	201 through 231
0301 through 0331	301 through 331
0401 through 0431	401 through 431
0501 through 0531	70 through 100
0601 through 0631	170 through 200
0701 through 0731	270 through 300
0801 through 0831	370 through 400
0901 through 0931	39 through 69
1001 through 1031	139 through 169
1101 through 1131	239 through 269
1201 through 1231	339 through 369

Let's go one step further. Suppose again that you wish to key on "day-of-year" but also to distinguish by year and want to span 10 years. Here we can have a key of type I2 represented by a decimal format of YYDDD (or YYYYDDD).

Choosing the YYDDD format with YY values between 87 and 96, leaves us with a minimum value of 87001 and a maximum of 96366.

Method 1 leads to a capacity of 96366 and a dataset 3.79% full.

Method 2 yields a capacity of 96366-87001+1 = 9366 and a dataset 39.07% full.

Method 3 yields a capacity of 5366 and a dataset which is 68.2% full with no synonyms. Not great, but much better than 39.07% and fantastically better than 3.79%:

Key Values	Record Numbers
87001 through 87366	1145 through 1510
88001 through 88366	2145 through 2510
89001 through 89366	3145 through 3510
90001 through 90366	4145 through 4510
91001 through 91366	5145 through 144
92001 through 92366	779 through 1144

93001 through 93366	1779 through 2144
94001 through 94366	2779 through 3144
95001 through 95366	3779 through 4144
96001 through 96366	4779 through 5144

Now, in anticipation of the next century, let's use a YYYYDDD format for a 20-year span starting with 1986 and ending with 2005, inclusive.

Method 1 results in a capacity of 2005366 and a dataset 0.36% full. Wow!!

Method 2 yields a capacity of 2005366-1986001+1 = 20000 and a dataset 36.6% full.

Method 3 yields a capacity of 10366 and a dataset 70.61% full with no synonyms:

Key Values	Record Numbers
1986001 through 1986366	6095 through 6460
1987001 through 1987366	7095 through 7460
1988001 through 1988366	8095 through 8460
1989001 through 1989366	9095 through 9460
1990001 through 1990366	10095 through 94
1991001 through 1991366	729 through 1094
1992001 through 1992366	1729 through 2094
1993001 through 1993366	2729 through 3094
1994001 through 1994366	3729 through 4094
1995001 through 1995366	4729 through 5094
1996001 through 1996366	5729 through 6094
1997001 through 1997366	6729 through 7094
1998001 through 1998366	7729 through 8094
1999001 through 1999366	8729 through 9094
2000001 through 2000366	9729 through 10094
2001001 through 2001366	363 through 728
2002001 through 2002366	1363 through 1728
2003001 through 2003366	2363 through 2728
2004001 through 2004366	3363 through 3728
2005001 through 2005366	4363 through 4728

Now let's look at a few keys which involve three subfields such as date fields of the forms YYMMDD and YYYYMMDD.

Suppose we want to span the five years 1989 through 1993 and, for simplicity, ignore the fact that not all months have 31 days. Each of the five "years" will have 12*31 = 372 days and the number of entries will be 5*372 = 1860.

Note that "fullness" percentages based on such 372-day years will always be about 1.6% high since 6 or 7 of the days of these "years" do not exist and hence don't require data entries.

Method 1 requires a capacity of 931231 for a dataset 0.19% full.

Method 2 requires a capacity of 931231-890101+1 = 41131 for a dataset 4.52% full.

Method 3 yields a capacity of 2241 for a dataset 82.99% full with no synonyms.

The charts showing the Record Numbers for the runs of examples involving three subfields take up too much space to include in this paper. If you should want them, please contact me.

If we span ten years from 1989 to 1998, inclusive, Method 3 yields a capacity of 5221 for a dataset 71.25% full with no synonyms. I will no longer bore you with the results of Methods 1 and 2.

Proceeding to the YYYYMMDD format we find that, to span fifteen years from 1989 to 2003, a capacity of 6246 yields a dataset which is 89.33% full with no synonyms. To include twenty years, say from 1989 to 2008, a capacity of 8746 yields a dataset 85.06% full with no synonyms.

Before leaving this section, I would like to show one final chart based on an integer key of the form YYXXXXX where the values of YY span the 15 years 1971 to 1985 and the XXXXX values go from 1 to 25000. Method 3 yields a capacity of 375000 and a dataset 100% full with no synonyms:

```
Key Values                        Record Numbers

7100001 through 7125000           350002 through      1
7200001 through 7225000            75002 through 100001
7300001 through 7325000           175002 through 200001
7400001 through 7425000           275002 through 300001
7500001 through 7525000                2 through  25001
7600001 through 7625000           100002 through 125001
7700001 through 7725000           200002 through 225001
7800001 through 7825000           300002 through 325001
7900001 through 7925000            25002 through  50001
8000001 through 8025000           125002 through 150001
8100001 through 8125000           225002 through 250001
8200001 through 8225000           325002 through 350001
8300001 through 8325000            50002 through  75001
8400001 through 8425000           150002 through 175001
8500001 through 8525000           250002 through 275001
```

Summary

There are several gurus writing papers or giving talks or publishing books about IMAGE internals, features and idiosyncracies. Some of these gurus stick to the facts. Others cloud the issue by peddling information which, however plausible and amusing, is either false, imprecise or overly simplistic. I call such gurus "IMAGE evangelists".

Having been the project leader of the IMAGE development team and co-developer of the primary address calculation algorithm for both hashing and non-hashing keys, I am amused when I hear or read the bad trash being peddled by such evangelists.

Some of the statements and recommendations emanating from these evangelists which this paper has shown to be false, imprecise or simplistic are:

1. "If you use integer keys, choose a capacity at least as large as the maximum key value."

   SIMPLISTIC. This advice leads to Method 1. Also IMPRECISE since the author meant to refer to the maximum "determinant".

2. "If you use integer keys, IMAGE hashes them to determine the primary address."

   FALSE. Keys of types I, J, K and R are NOT hashed.

3. "Clustering is bad."

   FALSE. When used properly, clustering is the VIRTUE of integer keys.

4. "If you use integer keys, always choose a prime number for the capacity."

   FALSE. None of our three capacity selection methods care about the "primeness" of numbers.

5. "If you use keys of type R4, IMAGE uses the leftmost 32 bits to calculate primary addresses."

   FALSE. IMAGE uses the rightmost 31 bits.

6. "If DBPUT has to search for a free spot in which to add a new entry (or to move an existing secondary), it inspects the next higher block and then the next lower and then the second higher and then the second lower and continues this ping-pong style of searching until it succeeds."

   FALSE. This statement is pure Science Fiction; truly bad trash. See also either of my previous papers "The Three Bears of IMAGE" or "The Use and Abuse of Non-hashing Keys in IMAGE". I wrote DBPUT's space-searching routine, so I know it doesn't "ping-pong".

7. "Don't let master datasets become more than 80 to 85% full."

   IMPRECISE. This is true only if there is a synonym problem. You may also get good performance up to 95% full if you have a large blocking factor such as 20 or 30. Integer keyed masters can be great even if 100% full.

JCWs: An Underutilized MPE Feature

David L. Largent

The N.G. Gilbert Corporation
700 South Council
P.O. Box 1032
Muncie, Indiana 47308-1032

317/284-4461

1.0 Introduction - "Is this for me?"

Have you been looking for a way to make your jobstreams a little "smarter?" Would a way to automate some of your procedures be helpful? Would you like a way for a program to pass a numeric value to another program without needing to use a file or interprocess communication? If you have any of these problems and have never considered using JCWs, now is the time to do so. If you are like me, you have known JCWs existed, but thought they were some difficult, mysterious, (impossible?) thing to master. This is not so!

On the following pages will be found answers to questions relating to Job Control Words (JCWs) such as:

* What are they?
* How can they help me?
* How are they created?
* How are they used?
* What are some common problems that arise?
* Are they worth the effort of learning something new?

To find the answers, we will first examine what JCWs are and how they are created and used. We will look at some examples to reinforce your learning. Lastly, we will look at how to use JCWs and UDCs together effectively to create a synergism of the two.

This paper is directed at new users of an HP 3000 who have not yet explored the world of JCWs. However, any user may find some tidbits to put in his or her "tool box" that will prove useful either now or in the future. This paper assumes you have a general knowledge of the HP 3000; specifically, the following are "prerequisites":

* Knowledge of many MPE commands and how to use them.
* Knowledge of jobstreams and how to use them.
* Knowledge of user defined commands (UDCs) and how to use them.

One further note. The information provided in this paper is correct and up-to-date (to the best of my knowledge) and reflects the way JCWs are used and work as of the G.03.08 version of MPE V/E (V Delta 8). I am aware changes have been made in MPE XL, but have not made any attempt to include that information here.

If you are working on an MPE XL system, I strongly encourage you to investigate and learn about the new variables and functions that are available for your use. Life is often much simpler with them.

2.0 What are Job Control Words (JCWs)?

JCW is one of the many acronyms used in the HP 3000 world. This one comes from the phrase Job Control Word. A JCW is MPE's way of permitting programs and commands to communicate with each other within a given job or session. JCWs are unsigned integer variables used at the operating system level with values ranging from zero through 65,535. Each JCW has a name and can be set and/or interrogated either by MPE commands or programs.

2.1 How can JCWs help me?
-or-
Why would I want a program to talk to my commands?

Good questions! Properly used JCWs will permit you to create "smarter" jobstreams. They can help to automate some of the decision making process in procedures. JCWs will permit a program to pass a numeric value to another program without the use of a file, or interprocess communication. They can even help you catch errors before they become a problem! All of this is to say: JCWs can help make the system more user friendly.

By testing JCWs against specific values, the user can program conditional statements that take action(s) based on the results of the test. JCWs can be set to predetermined values to indicate completion of steps within a procedure. JCWs can be checked to determine if certain events (usually errors) have occurred within MPE. With some limitations, JCWs can be used to pass control totals between programs.

2.2 OK, I think I see how they could help me.
So, how do I create and use a JCW?

First, some background information. Three classes of JCWs exist: user-defined JCWs; system-defined JCWs; and system-reserved JCWs. In some ways, they are exactly the same -- in other ways, they are completely different.

User-defined JCWs are named and assigned values solely by the user. MPE never changes the value of, or interrogates a user-defined JCW. The user creates and assigns a value to this class of JCWs with the SETJCW command or the PUTJCW intrinsic. The JCW name must begin with an alphabetic character and consists of a maximum of 255 alphabetic or numeric characters. You may not begin a JCW name with the mnemonic names OK, WARN, FATAL, or SYSTEM except under very specific conditions. (If you want to know what they are, see the HP "Commands" manual.) The value assigned to a user-defined JCW must be in the range of zero to 65,535 inclusive. User-defined JCWs can be interrogated by the user with the SHOWJCW command and the FINDJCW intrinsic. These new commands and intrinsics will be discussed later, so please be patient.

System-defined JCWs are named by the system and assigned values by the system and/or by the user. Both the system and the user may interrogate system-defined

JCWs. Only two system-defined JCWs exist: JCW and CIERROR. Both are created and set to zero at the beginning of every job or session. They will remain zero unless the user changes their value or an error occurs. The JCW named JCW has two special values:

49,152 (System 0)	Program aborted per user request.
A value greater than 49,152	Program terminated in an error state.

The CIERROR JCW keeps track of the command interpreter (CI) errors. If a CI error occurs, CIERROR is set to reflect the most recent error number. Valid commands do not reset CIERROR to zero. Thus, it always contains the number of the last error that occurred, unless the user resets its value. Generally, it is best not to alter the values of the system-defined JCWs. If you need to control a JCW, it is best to use a user-defined JCW.

System-reserved JCWs are named and assigned values solely by the system. Users may not change the value of a system-reserved JCW. They can, however, interrogate it. There are six system-reserved JCWs: HPMINUTE, HPHOUR, HPDAY, HPDATE, HPMONTH, and HPYEAR. The following briefly explains what each is:

HPMINUTE	Minute of the hour: values are 0 through 59.
HPHOUR	Hour of the day: values are 0 through 23.
HPDAY	Day of the week: values are 1 through 7; Sunday = 1.
HPDATE	Day of the month: values are 1 through 31.
HPMONTH	Month of the year: values are 1 through 12; January = 1.
HPYEAR	Year of the century: values are 0 through 99.

2.21 JCW usage in jobs and/or sessions.

OK, so now you know what JCWs are. You even know about the three classes of JCWs and what who can do to what. But, how do you look at or set their values? For jobs or sessions, the answer is: with the SHOWJCW and SETJCW commands.

The SHOWJCW command displays the current value of one or more JCWs. Its syntax is:

```
SHOWJCW [jcwname]
```

where "jcwname" is a valid JCW name (any class). If a name is provided, then only the value for that JCW will be displayed. If a name is not provided, then all system-defined and user-defined JCWs and their values are displayed -- system-reserved JCWs are not displayed. This command can be executed from a session, job, in BREAK, or from a program. It is BREAKable (it aborts execution of the command).

If no user-defined JCWs have been created and the user types:

```
SHOWJCW
```

the system will respond with:

```
JCW=0
CIERROR=0
```

unless some error has occurred prior to this command.

If you wish to see the current value of a specific JCW, you might type:

```
SHOWJCW HPDAY
```

and the system would respond with:

```
HPDAY=3  SYSTEM RESERVED JCW
```

Or, if you typed:

```
SHOWJCW UPDATEERRORS
```

and UPDATEERRORS was a valid user-defined JCW name, the system would respond with:

```
UPDATEERRORS=2
```

Big deal, so you can look at the value of a JCW. So what?!? OK, let me tell you how you can change or set the value of a JCW -- that is a little more productive. We need the SETJCW command to do this:

```
                          [+value]
SETJCW jcwname=value[-value]
```

where "jcwname" is the name of a new or existing user- or system-defined JCW and "value" represents one of the following:

1. An octal number between zero and %177777, inclusive.
2. A decimal number between zero and 65,535, inclusive.
3. An MPE-defined JCW value mnemonic (OK, WARN, FATAL, or SYSTEM)
4. The name of an existing JCW.

All values must be in the range of zero to 65,535, inclusive. That is, if the "+" or "-" option is used, the result of the arithmetic must be in the range as well. (The equal sign following the "jcwname" may actually be one or more punctuation characters or spaces, except "%" and "-". If you prefer some other notation, feel free...) This command can be executed from a session, job, in BREAK, or from a program. It is not BREAKable.

A word or two about the four JCW value mnemonics. They are:

OK	value is zero
WARN	value is 16,384
FATAL	value is 32,768
SYSTEM	value is 49,152

These are strictly mnemonics for specific values -- they cannot be used as JCW names. You may use a combination of a mnemonic and a number to indicate a value between two mnemonics. If you specify:

```
FATAL32
```

for example, an implied addition takes place (32,768 + 32) and the value would be 32,800. The "+" and "-" option can also be used with mnemonics. For example:

```
FATAL - 768
```

would result in a value of 32,000. If the SHOWJCW command is used to display current JCW values, and a value is greater than one of the mnemonics, then the value will be displayed as the mnemonic plus the amount over. For example, a value of 16,386 will be displayed as:

```
WARN2
```

An exception to this is that any value less than 16,384 will be shown as the actual number.

When the SETJCW command is executed, it causes the MPE JCW table to be scanned for the name of the specified JCW. If the name is found, the JCW is set to the value provided. If the name is not found, it is added to the JCW table and then set to the value provided. If "@" is used for "jcwname", all JCWs will be set to the value provided. Once a JCW is created, it exists for the duration of that session or job. There is no way to delete a JCW, short of logging off.

You are still not feeling very productive with JCWs yet, right!?! OK, here is the good stuff. JCWs are most often used to control the flow of batch jobs (they can also be used in UDCs and/or in sessions), taking various actions based on the results of previous steps. To do this, the IF/THEN, ELSE, and ENDIF commands are used. For purposes of this paper, I am going to assume you either know how these MPE commands work or can easily acquire the knowledge as we look at examples. (Some examples will come later that should clear up some of your questions.)

2.22 JCW usage in programs.

Now, for you programmer-type people, we will look at how to interrogate and set JCWs from within a program. My examples are based on COBOLII usage; however, I will try to provide information in a general way.

The programmatic equivalent to the SHOWJCW command is the FINDJCW intrinsic. Its syntax (in COBOLII format) is:

```
CALL INTRINSIC "FINDJCW" USING jcwname,jcwvalue,status
```

where "jcwname" is an alphanumeric variable (byte array) containing the name of the JCW to be found, "jcwvalue" is an unsigned one-word integer variable (logical) to which the JCW value is returned and "status" is a signed one-word integer variable (integer) to which a value denoting the execution status of the intrinsic is returned. The "jcwname" parameter may contain up to 255 alphanumeric characters, starting with a letter and ending with a non-alphanumeric character, such as a blank. If the requested JCW is found, its value is returned to the program in the "jcwvalue" parameter; if not found, no change is made to this parameter. The "status" parameter will be returned with one of four possible values:

0	Successful execution; the JCW was found.
1	Error: "jcwname" is longer than 255 characters.
2	Error: The value of "jcwname" does not start with a letter.
3	Error: The JCW was not found in the JCW table.

The FINDJCW intrinsic can be used to return the value of any of the three classes of JCWs.

To accomplish the same task in RPG, use the FNDJW operation. Specify the JCW to be found in the Factor 2 Field, and the name of a variable to contain the JCW value in the Result Field. One or more Resulting Indicators are required for return of the operation status.

The SETJCW command's programmatic equivalent is the PUTJCW intrinsic. Its syntax (in COBOLII format) is:

```
CALL INTRINSIC "PUTJCW" USING jcwname,jcwvalue,status
```

where "jcwname" is an alphanumeric variable (byte array) containing the name of the JCW to be created or changed, "jcwvalue" is an unsigned one-word integer variable (logical) containing the value for the JCW, and "status" is a signed one-word integer variable (integer) to which a value denoting the execution status of the intrinsic is returned. The "jcwname" parameter may contain up to 255 alphanumeric characters, starting with a letter and ending with a non-alphanumeric character, such as a blank. If "@" is the value used, all JCWs will be set to the value provided. If the specified JCW already exists in the JCW table, its value is changed to the value provided in the "jcwvalue" parameter; if not there, an entry is created and then it is assigned the specified value. The "status" parameter will be returned with one of six possible values:

0	Successful execution; value entered in the JCW table.
1	Error: "jcwname" is longer than 255 characters.
2	Error: The value of "jcwname" does not start with a letter.
3	Error: JCW table overflow; no room to create this new JCW.
4	Error: Attempted to assign a value to a JCW value mnemonic.
5	Error: Attempted to assign a value to a system-reserved JCW.

The PUTJCW intrinsic can be used only to assign a value to a user-defined or system-defined JCW.

To accomplish the same task in RPG, use the PUTJW operation. Specify the value to be assigned to the JCW in the Factor 1 Field, and the JCW to be assigned the value in the Factor 2 Field. One or more Resulting Indicators are required for return of the operation status.

As mentioned earlier, if the JCW named JCW is set to exactly 49,152, when the program stops running, the system will display:

```
PROGRAM ABORTED PER USER REQUEST (CIERR 989)
```

If it is set to any value greater than 49,152, then the message displayed will be:

```
PROGRAM TERMINATED IN AN ERROR STATE (CIERR 976)
```

In either case, if the program is running in a batch job or a UDC, the job or UDC will terminate unless a CONTINUE command precedes the RUN command.

Another possibility with JCWs is to use them to permit separate processes within the same job or session to communicate with each other. If a process were to set a JCW to a given value when a certain event occurred, then any other related process could check that JCW to find out when it occurred or what has occurred. Remember, however, that only numeric information can be assigned to JCWs.

An example of this is to have program "A" calculate a control total for a file that is needed in program "B". Rather than write the value to a file, display the value on the screen for later human input, or use interprocess communication, we could have program "A" set a JCW to the control total value (as long as it was less than 65,536). Then when program "B" needed the control total, it would simply call FINDJCW to retrieve the value, thus eliminating the need for human intervention, a file, or interprocess communication. The two programs must be run from the same session or batch job however, since JCWs are lost when a logoff occurs.

Just so you can't say I didn't tell you about them, two other intrinsics exist: GETJCW and SETJCW. They only permit you to interrogate and set, respectively, the value of the system-defined JCW named JCW. Since they have limited usability and you can use FINDJCW and PUTJCW to accomplish the same thing, I do not suggest learning about them.

2.3 How about some examples?

Good idea! First, we will look at some examples in batch jobs, then we will take a look at a couple example programs that use JCWs.

2.31 Batch Job Examples.

Since most of us are likely programmers, I will start with an example jobstream I use when working with a program. Virtually all programs created at the N.G. Gilbert Corporation are written in the PROTOS language. For those not familiar with PROTOS, it is a program generator whose output is a complete, structured, COBOLII program. Once the program has been written in PROTOS (and keyed into an editor file), the next step is to have PROTOS create the COBOLII source program. When PROTOS completes that task, you need to compile the COBOLII source program. The final step is to prepare the object program to produce the program file.

Since I do not want to tie up a terminal (and the system) while all of this transpires, I use an "intelligent" jobstream to handle the various tasks.

```
!Job prog,PRGRMR.NGG
!Comment   This jobstream performs a ProWrite, COBOLII compile,
!Comment   PREPare, and SAVE of a program. If any errors
!Comment   are encountered along the way, a message is sent
!Comment   to PRGRMR.NGG, and the jobstream stops running.
!SetJCW JCW = OK
!SetJCW CIError = OK
```

```
!Continue
!ProWrite progP,prog,$STDLIST,2000
!If JCW <> OK Then
! Tell PRGRMR.NGG; ProWrite of prog aborted<ctrl>G.
!Else
! If PROTOSError <> OK Then
!   Tell PRGRMR.NGG; Errors in ProWrite of prog<ctrl>G.
! Else
!   Tell PRGRMR.NGG; ProWrite of prog done. Compile started.
!   File COPYLIB=PROCOPY
!   Continue
!   COBOLII prog, $NewPass
!   If JCW <> OK Then
!     If JCW = Warn Then
!       Tell PRGRMR.NGG;prog had questionable errors<ctrl>G.
!     Else
!       Tell PRGRMR.NGG;Compile of prog had severe errors<ctrl>G.
!     Endif
!   Else
!     Tell PRGRMR.NGG; Compile of prog done. Prepare started.
!     Purge progU
!     Save $OldPass, progU
!     SetJCW CIError = OK
!     Continue
!     Prep progU,$NewPass;MaxData=20000
!     If CIError <> OK Then
!       Tell PRGRMR.NGG;Prepare of prog was abnormal<ctrl>G.
!     Else
!       Purge progX
!       Save $OldPass, progX
!       Tell PRGRMR.NGG;Program prog is done <ctrl>G.
!     Endif
!   Endif
! Endif
!Endif
!EOJ
```

This example demonstrates how you can structure a jobstream to check for errors and take different actions based on the occurrence or non-occurrence of errors. Some points of interest:

1. JCWs JCW, and CIERROR are set to OK (i.e., zero) at the start of the job to guarantee they start at zero. It is possible a LOGON UDC may have failed, and thus one or both of them may have a non-zero value.

2. Even if one of the programs (PROWRITE, COBOLII, or PREP [SEGMENTER]) aborts, the jobstream will continue because of the CONTINUE commands. This permits us to check JCW after each one to see if it aborted. If the CONTINUE commands were not used, the jobstream would abort before we could check the JCWs.

3. PROTOSERROR is a JCW that the PROWRITE program creates and sets equal to the number of errors found in your PROTOS program. By checking it, we can determine whether it is worthwhile to continue with the compile and prepare steps.

4. Notice that you may nest the IF statements to create whatever logic that might be required.

Here is a variation of this same jobstream. This version just tells the programmer there is an error, but does not report in detail like the previous version:

```
!Job prog,PRGRMR.NGG
!SetJCW JCW = OK
!SetJCW CIError = OK
!SetJCW False = 1
!SetJCW NoProblems = OK
!Continue
!ProWrite progP,prog,$STDLIST,2000
!If JCW <> OK Then
! SetJCW NoProblems = False
!Else
! If PROTOSError <> OK Then
!   SetJCW NoProblems = False
! Else
!   File COPYLIB=PROCOPY
!   Continue
!   COBOLII prog, $NewPass
!   If JCW <> OK Then
!     SetJCW NoProblems = False
!   Else
!     Purge progU
!     Save $OldPass, progU
!     SetJCW CIError = OK
!     Continue
!     Prep progU,$NewPass;MaxData=20000
!     If CIError <> OK Then
!       SetJCW NoProblems = False
!     Else
!       Purge progX
!       Save $OldPass, progX
!     EndIf
!   EndIf
! EndIf
!EndIf
!If NoProblems = False
! Tell PRGRMR.NGG;Errors in program prog <ctrl>G.
!Else
! Tell PRGRMR.NGG;Program prog is done.
!EndIf
!EOJ
```

In this version, JCW NOPROBLEMS is set to indicate when an error has occurred and then is checked at the end of the jobstream to determine what to tell the programmer. (Note the creation of JCW FALSE.)

Here is an example of making use of the system-reserved JCWs. This is a modified version of the jobstream we use to do our daily backups.

```
!Job BackUp,Operator.Sys
!SetJCW Monday = 2
!ShowAllocate
!ShowCache
!Run FREE5.PUB.SYS
!Report
!If HPDay = Monday Then
!  FullBackUp
!Else
!  PartBackUp
!Endif
!Stream JHPTREND.HP35136A.TELESUP
!If HPDay = Monday or HPDay = Monday+3 Then
!  Stream PREDICTJ.HP51467A.TELESUP
!Endif
!EOJ
```

Points of interest in the backup jobstream:

1. Notice the creation of the JCW MONDAY. This is not necessary, but it makes the IF command read a little nicer.

2. If the JCW HPDAY has a value of two, then it is Monday and a full backup should be performed. Otherwise, a partial is done.

3. When the backup has completed, the HP Trend jobstream is restarted, and if it is Monday or Thursday (Monday + 3), the HP Predict jobstream is also started.

Here is an example of using the IF command and JCWs to do something they were not designed for, but it works, so why not? The following could be included at any point in a jobstream where JCW would not be equal to FATAL (which could be virtually anywhere):

```
!If JCW = Fatal Then
    From this point on (up to an ELSE or ENDIF command), you can type
    whatever you wish. The lines do not even need to start with an
    exclamation point! The reason this works is that when the condition
    in an IF command is false (which it is in this instance), all command
    lines are ignored until an ELSE or ENDIF command is read. Thus, this
    provides an easy way to include comments without using the COMMENT
    command. The "IF" could actually be written "If 1 = 2 Then" (or
    anything that would evaluate to false), but then it wouldn't be an
    example of JCW usage!
!Endif
```

Have you ever wanted a way to tell if a file was empty (i.e. no records) before running a program? Here is a jobstream fragment that shows one way to do it:

```
!SetJCW JCW = OK
!File DUMMYXYZ;Temp
!FCopy From=file-to-test;To=*DUMMYXYZ;New
!If JCW = OK Then
!       Run program          (the file has something in it)
```

```
!Else
!      If JCW = WARN Then
!             (The file is empty)
!      Endif
!Endif
!Purge DUMMYXYZ
```

The key that makes this work is that if FCOPY does not find anything to copy, it sets the JCW JCW to a value of WARN. Otherwise, it does not change the value. You might consider making this set of commands a UDC. The UDC could set a JCW to indicate whether the file is empty, rather than running a program. Then you would simply execute the UDC, and check the setting of the JCW to decide what you should do.

Two other sources of example jobstreams are the PREDICTJ and JREDUCE jobstreams from HP. They make extensive use of JCWs to control the flow of the jobstream logic.

2.32 Programmatic Examples.

The following is a COBOLII subroutine (actually it was written in PROTOS) I created to call in any situation where I want to end a program and have:

```
PROGRAM ABORTED PER USER REQUEST (CIERR 989)
```

displayed afterwards (granted, this is not very often!). The sole purpose of this program is to set the JCW JCW to a value of 49,152 (octal 140000).

```
$CONTROL DYNAMIC,BOUNDS
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SETABORTJCW.
 AUTHOR. DAVID L LARGENT.
 DATE-WRITTEN. TUE, OCT 20, 1987,  3:41 P.M.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

*******************************
 Sets JCW to "Program Abort"
*******************************

 01 JCW-NAME                          PIC X(4)
      VALUE "JCW ".
 01 JCW-STATUS                        PIC S9(4) COMP.
 01 JCW-VALUE                         PIC 9(4) COMP
      VALUE %140000.

 PROCEDURE DIVISION.

 MAIN-LINE-SECTION SECTION.
 MAIN-LINE.

      MOVE ZERO                       TO JCW-STATUS.
```

```
          CALL INTRINSIC "PUTJCW" USING JCW-NAME JCW-VALUE
               JCW-STATUS.
          IF JCW-STATUS NOT = ZERO
               DISPLAY "Program SETABORTJCW:   JCW not set."

          GOBACK.
```

This may not be a very useful program to you, as it is printed; however, it does show the basics of what needs to be done to create and/or set a particular JCW to a given value. If your JCW name is longer than three characters, make sure you increase the length of field JCW-NAME. Also, make sure you have at least one blank or some other non-alphanumeric character following your JCW name. The value for JCW-VALUE can be specified as an decimal number if you prefer, provided it is less than 10,000. In the example, the octal value was required because 49,152 was too large.

If you want the program to stop instantly, you are better off calling the QUIT intrinsic with a parameter value of 16384. However, if you want the program to continue running, so you can perform a "controlled shutdown" of the program, you must take the approach shown above.

Here is a trivial example of the FINDJCW intrinsic, but again it shows the basics of what needs to be done to retrieve the value of an existing JCW.

```
$CONTROL BOUNDS
 IDENTIFICATION DIVISION.
 PROGRAM-ID. DISPLAYJCW.
 AUTHOR. DAVID L LARGENT.
 DATE-WRITTEN. WED, APR 27, 1988,  4:24 A.M.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

******************************
  Displays the Value of JCW
******************************
 01 JCW-NAME                              PIC X(4)
      VALUE "JCW ".
 01 JCW-STATUS                            PIC S9(4) COMP.
 01 JCW-VALUE                             PIC 9(4) COMP.
 PROCEDURE DIVISION.

 MAIN-LINE-SECTION SECTION.

 MAIN-LINE.

      MOVE ZEROS                          TO JCW-STATUS
                                             JCW-VALUE.

      CALL INTRINSIC "FINDJCW" USING JCW-NAME JCW-VALUE
           JCW-STATUS.
      IF JCW-STATUS NOT = ZERO
           DISPLAY "Program DISPLAYJCW: JCW not found."
```

```
          ELSE
               DISPLAY "JCW = " JCW-VALUE

          STOP RUN.
```

Same comments as last time: make sure you set up JCW-NAME large enough to hold your JCW name and make sure you end it with at least one blank or other non-alphanumeric character.

2.4 What are some problems I may have while using JCWs?

Throughout the paper, I have provided a number of warnings and limitations. Listed here (in somewhat random order) are a few worth repeating.

* A JCW name must start with a letter and can consist of a maximum of 255 alphabetic or numeric characters.

* The value assigned (whether a number or a calculated value) must be in the range of zero to 65,535, inclusive.

* The system-reserved JCWs may not be assigned a value by a program or the user.

* If you want to check for program errors (e.g., program aborts), make sure to include a CONTINUE command before the RUN command. Also, make sure JCW and CIERROR are set to zero beforehand.

* Remember, the only thing that changes the values of CIERROR and JCW (except the user) is another error. That is, a valid command does not set them to zero!

* Although JCWs can have any value from zero through 65,535, only values from zero through 16,383 will be displayed as numbers. Values larger than 16,383 will be displayed as a JCW value mnemonic and an offset beyond the mnemonic value.

* When calling PUTJCW or FINDJCW from a COBOLII program, make sure the JCW-STATUS field is defined as S9(4) COMP, and the JCW-VALUE field is defined as 9(4) COMP. The program will not compile or run correctly if they are not.

3.0 How can I use UDCs and JCWs together?

UDCs and JCWs can be used together very effectively. JCWs can control the flow of logic within a UDC. A UDC (by way of its parameters) can gather information and use that information to set JCWs so that a program can then interrogate them and take appropriate actions. So, let us look at some examples of combining these two powerful capabilities.

3.1 Examples of Combining UDCs and JCWs.

To start off with, we will look at some logon UDCs. First, a simple addition to the system-level logon UDC to make use of the system-reserved JCW HPDAY easier:

```
SysLogOn
Option LogOn,NoBreak
SetJCW Sunday = 1
SetJCW Monday = 2
SetJCW Tuesday = 3
SetJCW Wednesday = 4
SetJCW Thursday = 5
SetJCW Friday = 6
SetJCW Saturday = 7
**
```

By including this UDC, these seven JCWs would always be available for use in IF commands. For example:

```
If HPDay = Monday Then
 FullBackUp
Else
 PartBackUp
EndIf
```

is easier to read and understand than "If HPDay = 2...".

Next, a way to have one thing automatically happen if a session logs on and something else if a job logs on:

```
UserLogOn
Option LogOn,NoBreak
SetJCW CIError = OK
Continue
Resume
If CIError = 978 Then
 Run (batch program)
Else
 Run (online program)
 Bye
EndIf
**
```

The RESUME command is not allowed in job mode (thus, CI error 978), so we can use the result of its execution to determine if a session or a job is logging on. Also, note that a job could be set up to do other things after the logon program runs because there is not an automatic BYE as there is for the session.

Here is a way to control when people logon and play games:

```
GamesLogOn
Option LogOn,NoBreak
If HPDay = Sunday or HPDay = Saturday or &
 HPHour < 8 or HPHour > 17 or HPHour = 12 Then
 Display "Welcome to the Game Room"
```

```
Else
 Display "Sorry, the Game Room is closed."
 Display "  Hours:       Saturday and Sunday: all day"
 Display "              Monday-Friday: Before 8 a.m., After 5 p.m."
 Display "                            and Noon to 1 p.m."
 Bye
Endif
**
```

If it is Saturday or Sunday, or before 8 a.m., after 5 p.m., or sometime during the noon hour, this UDC will let the user stay logged on -- any other time and the user will automatically be logged off. Note that if the user gets logged on during an "open" time, they can continue playing forever -- there is nothing to force them off when the game room closes.

The use of DISPLAY in this and the following examples, references a UDC that issues a COMMENT command followed by an escape sequence to delete the current line. It then goes on to show whatever phrase was passed to it. The end result: only the phrase is left on the screen -- a little "cleaner" than just using COMMENT by itself.

The following UDC is executed every morning by our operator or anytime the machine is restarted. It provides a quick, easy way of getting everything set to a known value:

```
SysUp
Option List
Streams 10
JobFence 0
JobPri CS,DS
JobSecurity Low
Continue
DiscRPS 3,Disable
Continue
StartCache 1
Continue
StartCache 2
SetJCW CIError = OK
Continue
StartCache 3
If CIError = OK Then
 Stream JHPTREND.HP35136A.TELESUP
Else
 Display "<esc>&dJ The HPTrend jobstream should be"
 Display "<esc>&dJ running.  If it is not,"
 Display "<esc>&dJ STREAM JHPTREND.HP35136A.TELESUP"
Endif
ShowJob JOB=@J
OutFence 1
HeadOff 6
Continue
StartSpool 6
Limit 3,16
SetOpKeys
**
```

A few points of interest:

1. Since we leave disc caching turned on virtually all of the time, this UDC assumes that if the "StartCache 3" command succeeds, the system must have just been started and, therefore, the HP Trend jobstream needs to be initiated.

2. The last thing that is done is to load the function keys on the system console with a UDC called SETOPKEYS.

Here are a couple of UDCs that can make things more convenient:

```
FindRun Program,Parm=0
SetJCW CIError = OK
Continue
Run !Program.Group1;Parm=!Parm
If CIError = 622 Then
 SetJCW CIError = OK
 Continue
 Run !Program.Group2;Parm=!Parm
 If CIError = 622 Then
  SetJCW CIError = OK
  Continue
  Run !Program.Group3;Parm=!Parm
  If CIError = 622 Then
     Display "This program was not found in"
     Display "group1, group2 or group3."
  Endif
 Endif
Endif
**
List FromFile=$StdIn,ToFile=$StdList
SetJCW CIError = OK
Continue
File PRINT;Dev=!ToFile
If CIError = 301 or CIError = 344 Then
 File PRINT=!ToFile
Endif
FCopy From=!FromFile;To=*Print
**
```

The FINDRUN UDC will try RUNning your program from three different groups before it gives up. This could be changed to be different accounts as well, and could have more nesting added to try in more locations if you wish.

The LIST UDC provides a lot of flexibility. The "to" file can be specified by providing a device class name, a logical device number, or a file name! To list a file on your terminal, you would type:

```
LIST file
```

To print a file on the line printer, you could type either one of these:

```
LIST file,LP
LIST file,6
```

To list a file on a terminal with a logical device number of 27 (the terminal would need to be turned on, but logged off), you would type:

```
LIST file,27
```

To copy a file to an existing disc file, you would type:

```
LIST file,file2
```

It is not possible to create a new disc file. Another option available with this UDC is to use the default "FromFile" value of $STDIN, which will accept input from your terminal keyboard, thus permitting you to "create" a file as you go.

Now let us look at a UDC in which information is passed to a program by way of JCWs. This example is from the UDCs PROTOS Software Company provides with the PROTOS language:

```
PROWRITE F,C="C",L=$NULL,S=1023,R=ROOTDB,Q=0,G=PROTOS
SETJCW PROBUILDWRITE = 2
SETJCW QEDITOUT = !Q
FILE SEMDOPE=SEMDOP01.!G.PROTOS;SHR
FILE SEMPASS=SEMPAS01.!G.PROTOS;SHR
FILE SSERR=SSERR.!G.PROTOS;SHR
FILE SEMTEMP=SEMTMP01.!G.PROTOS;SHR
FILE ATNIN1 = !F
FILE ATNOUT2 = !L
PURGE !C
BUILD !C;REC=-80,16,F,ASCII;CODE=EDTCT;DISC=!S
FILE COBOLOUT = !C
FILE ROOTDB = !R
IF QEDITOUT = 1 THEN
 PURGE QECOBOUT,TEMP
 FILE QECOBOUT;REC=256;DISC=!S;TEMP
ENDIF
RUN PROTOS.!G.PROTOS;LIB=G
IF QEDITOUT = 1 THEN
 PURGE !C
 RENAME QECOBOUT,!C,TEMP
 SAVE !C
ENDIF
RESET SEMDOPE
RESET SEMPASS
RESET SSERR
RESET SEMTEMP
RESET ATNIN1
RESET ATNOUT2
RESET COBOLOUT
RESET ROOTDB
**
```

In this example, there are two JCWs used to pass information to the PROTOS program: PROBUILDWRITE and QEDITOUT. The QEDITOUT JCW gets its value from the Q parameter of the UDC. Also, based on the QEDITOUT JCW, different parts of the UDC are executed.

JCWs can be assigned numeric values only. So how do you make use of character string information from a UDC parameter? Here is one way:

```
CopyW2ToTape CharSet=X
SetJCW A = OK
SetJCW E = OK
SetJCW !CharSet = 1
If A = OK and E = OK Then
 Display "An 'A' or an 'E' must follow the copy command.<ctrl>G"
Else
 File W2TAPE;Dev=TAPE;Rec=-276,25,F,ASCII
 If A = 1 Then
  FCopy From=PR997TAP;To=*W2TAPE
 Else
  FCopy From=PR997TAP;To=*W2TAPE;EBCDICOUT
 Endif
 ListF PR997TAP,1
 Display "<esc>dJ Check the number of records copied.<ctrl>G"
Endif
**
```

The purpose of this UDC is to copy a disc file to magnetic tape. The catch is that we need the option of copying it in EBCDIC format; thus, the CHARSET parameter. There are two "correct" responses: A and E. At the beginning of the UDC, two JCWs (A and E) are created and set to zero, then the user's choice is set to one. By having a default value that is not correct, if the user does not provide a value, the UDC will provide a message to the user and "remind" him/her of the correct values.

Here is another example of using parameter values from a UDC to establish JCWs that control what the UDC actually does. This UDC comes from Eric Omuro of Complimate, Inc., and is an example of how to eliminate multiple UDCs that are each a slight variation on the same theme -- in this instance, the SHOWJOB command.

```
SJ Parm1 = "SE" Parm2 = " " Parm3 = " "
SetJCW SEO = Warn
SetJCW SJ = Warn
SetJCW SS = Warn
SetJCW SST = Warn
SetJCW SSC = Warn
SetJCW SU = Warn
SetJCW SW = Warn
SetJCW SX = Warn
SetJCW S!Parm1 = OK
If SJ = OK Then
 ShowJob Job = @J;!Parm2
Else
 If SS = OK Then
  ShowJob Job = @S;!Parm2
```

```
Else
 If SX = OK Then
  ShowJob Job = @J;Exec
 Else
  If SW = OK Then
   ShowJob Job = @J;Wait
  Else
   If SSC = OK Then
    ShowJob = @J;Sched
   Else
    If SST = OK Then
     ShowJob Status
    Else
     If SU = OK Then
      ShowJob Job = !parm2;!parm3
     Else
      If SEO = OK Then
       ShowJob !Parm2
      Else
       ShowJob #!Parm1
      EndIf
     EndIf
    EndIf
   EndIf
  EndIf
 EndIf
EndIf
EndIf
**
```

The first parameter of the SJ UDC, "Parm1", identifies which format of the SHOWJOB command to execute. Generally, "Parm2" and "Parm3" are only used when executing SJ U. The occurrences of "Parm2" on the other commands offer additional flexibility and challenge the advanced SJ UDC user to figure out how they can be used.

4.0 Closing Thoughts – Is it worth the effort of learning something new?

We have looked at Job Control Words. Numerous examples have been explored to see how they work and how they can be used. They are a very powerful feature of the HP 3000. So, is it worth the effort? My answer is a (qualified) resounding YES! The qualification is that JCWs must be carefully planned and monitored to reap the greatest benefit, but, oh, what a benefit it is: increased operator, programmer, and user productivity and a computer system that is easier to use overall.

Acknowledgements

A special thanks go to the following people who assisted in creating this paper in some way -- all of their efforts (and tolerance!) are greatly appreciated:

H. Lyle Harp
Dan Hinds
Jonathan Largent
Julia Largent
Lois Largent
Karen Morgan
Dawn Thomas
Martha Walsh

Bibliography

Griffin, Brad	"Another Way to SetJCWs as Part of a Log-on UDC", SuperGroup Association Magazine, June, 1986, page 12.
Hewlett-Packard Company	MPE V Commands Reference Manual, Second Edition, 1989, Chapter 2.
Hewlett-Packard Company	MPE V Intrinsics Reference Manual, Third Edition, 1989, Chapters 2 and 5.
Hewlett-Packard Company	RPG/V Reference Manual, Fifth Edition, Update 1, 1989, Chapter 8.
Hewlett-Packard Company Atlanta Response Center (specifically B.A.)	Calls to the Atlanta HP Response Center during April, 1988.
Hewlett-Packard Company North American Response Centers	HP3000 Application Note #21: COBOLII and MPE Intrinsics, January 15, 1987.
Largent, David L.	"A Beginner's Guide to UDCs and JCWs: How to Use Them to Your Benefit (Part 3)", Interact, May, 1989, page 130ff.
Omuro, Eric	"Smarter UDCs", Interact, April, 1990, page 22ff.
PROTOS Software Company	PROTOS Documentation, version 3.5.
Volokh, Eugene	"Conditional Execution - :IF, :ELSE, :ENDIF, ET. AL.", SuperGroup Association Magazine, February, 1986, page 28ff.

KSAM 101

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

1.INTRODUCTION

KSAM has been the poor relation of IMAGE ever since the HP3000 was first produced, and it seems that there are many people who can use IMAGE with flair, but have no idea how to begin using KSAM. This is unfortunate, since there are many things that are difficult or impossible with IMAGE that can easily be done with KSAM. Now that HP is about to release a Native Mode version for the XL machines, it is time to take another look at what KSAM can do for you.

This article will attempt to give a friendly introduction to KSAM. It assumes that the language of choice is COBOL and that the reader is a well-rounded programmer, but it is not attempting to cover every subtlety of using KSAM.

1.1 The Key Structure

KSAM is a system for obtaining rapid random access to the contents of a data file. It does this with a key structure that allows a program to find the data record corresponding to that key with a small number of reads from the disc.

The method used is known as a B-tree structure. This access method is probably the most frequently used on computers today, far more common than the IMAGE method of hashing. Many of the well-known database structure use this method -- examples would be VSAM on IBM mainframes, Oracle or C-ISAM on minicomputers, Dbase3 on micro-computers.

The logical structure of this key file is like a tree, and is shown in the Appendix B of the KSAM Manual. The file has blocks of key records, each record consisting essentially of a key value and pointers that point to either a data record or another key-block.

When data is deleted the keys are removed from the tree structure, but the data record is not. It is marked as deleted by placing high-values in the first two bytes of the data record. This method of deletion has certain impacts which will be discussed later.

1.2 Operations on a KSAM file

The KSAM file can be read sequentially as a simple data file, without any reference to the key file. This is known as reading chronologically, since data records are always added at the physical end of the file.

It is more usual to read a KSAM file sequentially by key. The default is to use the primary key but, if desired, the read may be a "sequential" read by any desired key.

Records can also be retrieved randomly by specifying the key of the record desired. The overall effect of this is similar to reading a primary record in an IMAGE dataset.

The third method of retrieval is essentially a combination of these two methods. A random read is made on a particular key, and then sequential reads by key can follow, thereby reading sequentially by key starting at any arbitrary place in the file.

1.3 Why Use KSAM?

Table I lists the more significant differences between KSAM and IMAGE. KSAM has some distinct advantages over its rival IMAGE. It can retrieve records by key just as IMAGE does, but, in addition, its capability to retrieve sequentially allows it to achieve results that are impossible with IMAGE. It is ideal for any situation such as an alphabetical directory, that requires a random jump into a data set, followed by a sequential read in a particular order.

There is however, another advantage that depends on its method of updating secondary keys. When a change is made to a data record that changes a secondary key, but not the primary, KSAM, unlike IMAGE, accepts this as an update, and changes the appropriate key structure to allow for it. In a file with a complex data structure, this can provide a very significant performance advantage over IMAGE.

Finally, KSAM can be accessed by a partial key. This means that we can, for example, combine a major key, similar to a master key in IMAGE, with a sort key, such as we might use in a detail set, into a single key, and get the logical equivalent of a sorted detail chain without the excessive update overhead usually present in an IMAGE dataset.

2 HOW TO USE KSAM

We will now discuss how to access a KSAM file from a COBOL program. There are two basic methods, one to use the COBOL verbs, the other to use the "CK" intrinsics. Each of the

"CK" intrinsics except one has an equivalent COBOL verb. The intrinsics are listed below, with the COBOL verb equivalents.

Intrinsic	COBOL Verb
CKOPEN	OPEN
CKOPENSHR	OPEN (with SHR on file equation)
CKCLOSE	CLOSE
CKSTART	START
CKREAD	READ NEXT
CKREADBYKEY	READ ... KEY IS
CKWRITE	WRITE
CKREWRITE	REWRITE
CKDELETE	DELETE
CKLOCK	EXCLUSIVE
CKUNLOCK	UN-EXCLUSIVE
CKERROR	no equivalent

Generally, the COBOL verbs can be used without problem. However, because COBOL will not usually allow a file to be passed from one program to a subprogram, the "CK" intrinsics must be used in a situation in which the data access is done in a dynamic subprogram and the file is kept open from one call to the next.

2.1 Using COBOL Verbs

2.1.1 OPEN. Coding the procedure division is simple, as the file is simply opened for input or I/O. Opening for output is also possible, but rather unlikely. The real meat of how the file is opened is in the "SELECT" statement and in working storage.

An example of a "SELECT" statement and associated data areas is given in Fig. 1. The "organization" statement identifies the file as a KSAM file. "Access mode" could be set to "sequential", "random" or "dynamic". "Dynamic" indicates that the file can be accessed randomly or sequentially. The primary and secondary keys are indicated and the location of the status word is indicated. The status area is the same, regardless of whether COBOL verbs or intrinsics are being used.

2.1.2 OPEN in shared mode. There is no way to indicate in the COBOL program that the file is to be opened in the shared mode. This must done by a file equation when the program is run.

2.1.3 CLOSE. This is the same as any other file opened in COBOL.

2.1.4 START. This verb tells KSAM where to start reading the file. The key value desired is moved into the corresponding position in the actual record. An "Invalid Key" error indicates that the record value could not be found. In the example in Fig 2, "invalid key" would indicate that the logical pointer is at the end of the file.

2.1.5 READ NEXT RECORD. Read the next record sequentially in the file for the present key. If we only want to read the records corresponding to a particular partial key value, then we must check that condition in the program.

2.1.6 READ .. KEY IS. This form of the read statement is a "Read By Key" and will retrieve the first record corresponding to the key value. It is equivalent to a "START" followed by a "READ NEXT".

2.1.7 WRITE. This verb will write to the file in a manner similar to writing to a sequential file. In particular it writes the new record to the physical end of the data file. An "invalid key" condition is possible if any key duplicates the value of a non-duplicate key on the file or if you have reached the end of the file. In sequential mode, it can also indicate that the records are out of order.

2.1.8 REWRITE. This verb updates the file. Never use this verb if the primary key is changed, as KSAM finds the record by means of the primary key. To change a primary key, the record should be deleted and a new record added with the new key value. Secondary keys may be changed with impunity, and usually with great efficiency, provided that you do not create unacceptable duplicate values.

The tricky thing about this process is what it may do to the logical pointer if a key value is changed. Check the manual for the details if you are a glutton for punishment, but, to be on the safe side, always do a "START" and a "READ" or a "READBYKEY" after an update that changes a key value.

2.1.9 DELETE. This verb deletes the record last read. Be careful to make sure you have the right record.

2.1.10 EXCLUSIVE. This locks the file. It also clears the file buffers to ensure that the most recent data is read from disk.

2.1.11 UN-EXCLUSIVE. This unlocks the file. Before unlocking, all data in the buffers are flushed to disk.

2.2 Using COBOL Intrinsics

2.2.1 CKOPEN. Unlike the COBOL verb, the critical items for opening a KSAM file are kept in a special area in working storage, the first word of which is the file number. This area, an example of which is given in Fig. 3, must be passed to any subprogram that is going to make calls to the KSAM intrinsics. The i/o parameter indicates whether the file is opened for input, output or I/O. Be careful, because if you open the file for output, you will clear all the records from the file. The access mode parameter indicates sequential, random or dynamic. The actual values are available in the KSAM manual.

2.2.2 CKOPENSHR. This intrinsic is identical to "CKOPEN" except that it causes the file to be opened in share mode.

2.2.3 CKCLOSE. This intrinsic closes the file.

2.2.4 CKSTART. This intrinsic corresponds to the "START" verb, and indicates where a sequential read will start.

Setting up this intrinsic is a little peculiar. The first two parameters are obvious enough, but the third needs a little explanation. It indicates whether the key comparison is equal to, greater than or not less than, by the values 0, 1, or 2 respectively.

Similarly the last two parameters are a little unusual for COBOL, in that they are integers indicating the starting position of the key and the length of the key. In COBOL, one can use the pseudo-intrinsics ".LOC." and ".LEN." to determine them.

2.2.5 CKREAD. This intrinsic is straightforward in use. However, note that the length used in CKREAD is the length of the record, not the key as in CKSTART.

2.2.6 CKREADBYKEY. This intrinsic is similar to the CKSTART in that it requires both record length and key location passed to it. The key length is unnecessary as, by definition, the comparison is equality and the key lengths must be the same.

2.2.7 CKWRITE. This intrinsic is straight-forward in use, and corresponds to the WRITE verb above.

2.2.8 CKREWRITE. This intrinsic uses the same parameters as "CKWRITE". It is the equivalent of the REWRITE verb and has the same problems and restrictions.

2.2.9 CKDELETE. Deletes the record just read. Calling this intrinsic is easy.

2.2.10 CKLOCK. Locks the file. Can only be used if the file were opened with CKOPENSHR. The third parameter is an integer variable to request conditional locking with a zero or unconditional locking with a "1".

2.2.11 CKUNLOCK. Unlocks the file. Easy to call.

2.2.12 CKERROR. This is a special intrinsic that converts the second status byte into a display number, so that it can be usefully displayed in a COBOL program. There is no equivalent COBOL construct, so this intrinsic is used even in situations where the KSAM file is being handled by the COBOL verbs.

2.3 Using Other File Intrinsics

Other file intrinsics may occasionally be used with KSAM. To call Fcontrol, or any other file intrinsic for a file opened by COBOL, use the "FD" label as the "file-number".

2.3.1 FCONTROL. There are several calls to the FCONTROL intrinsic that are used with KSAM files. Note that in many cases, the function executed is not the same as the function executed with other files as described in the intrinsic manual. Also note that Fcontrol 2 and 7 are executed automatically when the file is unlocked and locked respectively. The effect of Fcontrol 5 can equally well be obtained with the appropriate "START".

2.3.1.1 Fcontrol 2. Flush the KSAM buffers out to disc.

2.3.1.2 Fcontrol 6. Same as Fcontrol 2, except that the MPE end-of-file marker is written as well.

2.3.1.3 Fcontrol 7. Clear the buffers. This ensures that a new set of key and data records is read in from disc for the next START or READBYKEY.

2.3.1.4 Fcontrol 5. Reposition to the first logical record for the current key.

2.3.2 FSPACE. This intrinsic may be used for moving the pointer backwards or forwards in the file. Occasionally usefully when "paging" forward or backward on a screen.

2.3.3 FGETINFO. Gives general information about the file.

2.3.4 FGETKEYINFO. Gives information about KSAM and the key file.

2.4 UTILITIES

2.4.1 Using KSAMUTIL

KSAMUTIL is the major utility for handling KSAM files. It creates them, extracts information about them, and allows the files to be recovered when damaged.

2.4.1.1 Building KSAM files. KSAM files cannot be created simply by opening a new file. They must first be created by using KSAMUTIL. The command of concern is the BUILD command, an example of which is given in Fig. 4. This is similar to the MPE BUILD command, except that there are several additional parameters, specifying the key file and the key structure. Because these specifications are usually so complex, I often write them as a jobstream, so that the format of the file is recorded and so that the file can always be rebuilt correctly.

2.4.1.2 Renaming KSAM files. KSAM files must be renamed using the KSAMUTIL function and not the MPE function. The name of the keyfile is stored internally and the connection between the files is lost if the MPE function is used.

2.4.1.3 Purging files. The MPE function can be used, providing that both the data file and the keyfile are purged.

2.4.1.4 Erasing files. A useful function. Rapidly removes all the data in a KSAM file. Much faster than purging and rebuilding.

2.4.1.5 Verify command. This function is useful for seeing the exact amount of data in the file, verifying the key structure and determining the level structure of the keys.

2.4.1.6 Keyinfo command. This command is very important as it recovers KSAM files if they have been damaged. There are several kinds of damage that can occur, either from a system failure or from poor programming practices. Keyinfo confirms the state of the file, corrects many types of problems, and determines whether a reorganization of the file is necessary.

You CANNOT use a KSAM file after a system failure that occurred when the file was open unless you have run "keyinfo" against that KSAM file. This is to protect you against any damage that may have been done to the file.

2.5 Using FCOPY

2.5.1 Reorganizing. The most important function of FCOPY for KSAM files is its use in reorganizing. If FCOPY is used to copy a KSAM file, it will automatically reorganize it. In other words, all deleted records will be omitted from the new copy and the records will be written in primary key order.

The latter effect can be overridden by using the "KEY" parameter. The default is that the data will be read sequentially by the primary key. "KEY=n" causes the data to be read by the key starting at location n. "KEY=0", as shown in Fig. 4, causes the data to be read chronologically, which should be significantly faster than by the default method. However, when reorganizing, the reading speed is the least of your worries.

2.5.1.1 Undeleting. Because KSAM doesn't actually delete the records, but only marks them for deletion, it is possible to retrieve records that have been deleted in error. The process is somewhat time-consuming, so it is wise not to make a habit of it. By copying a file with the parameter "NOKSAM", the data file is read as a sequential file, deleted records and all.

The file may be copied to a flat file and then reorganized into a new KSAM file to recover all the records. However, this may not be possible if you create unacceptable duplicates in the process.

You can find only the deleted records by adding the parameter "SUBSET=#%377,%377#,,EXCLUDE". By copying to a printer, you can read all the deleted records and select only those that you want to keep.

2.5.2 Other Software. There is only one utility applicable to KSAM that I am aware of on the market, namely "Copyrite" from Tymlabs. The techniques used by this program make drastic cuts in the copying time, and, most importantly, reorganizing time. Tymlabs claim up to 95% improvement in efficiency and I have myself seen improvements of five to ten times. When this comes out to the difference between five hours and half an hour, the utility can pay its way in a very short time indeed. Furthermore, it changes one's approach to using KSAM, if the time taken to reorganize the files after a system failure is no longer a serious consideration.

There are also some programs on the user tapes that deal with KSAM files, but I have no personal experience of them.

3. PROBLEMS

There are some catches in using KSAM and there are a two or three that need to be mentioned.

3.1. Setting up the File

As previously discussed, KSAM deletes a record by placing high-values in the first two bytes of the data record, and this fact can be used to retrieve records deleted in error. However, to make recovery complete, it is important that the delete mark not be written over data that will be needed. For this reason, I always leave these two bytes blank, and I recommend this approach to anyone setting up a KSAM file.

I would also suggest that you normally make the primary key unique. It is not absolutely required, but because KSAM uses this key to decide which record to update, it can be very difficult to be sure that you are updating exactly the right record if the primary key is not unique.

It is important to realize that the key file uses a lot of disc space, depending on the length of the keys and how many keys there are. Often the key file is significantly larger than the data file and it is important to take this into consideration during the planning stage, or disc space can be seriously underestimated.

3.2. When to Lock

Locking is a major consideration on KSAM files when sharing access, which is almost always the case in an on-line situation. The basic rule is really simple: always lock the file!

This sounds, and is, rather more severe than the KSAM manual would have you believe, but there are good reasons for it. The locking can be relaxed in a few places but only with the proper precautions.

First, always lock when updating the file in any way: add, change or delete. If this is not done, there is a significant possibility that the key file will be scrambled and the only rescue from this is a reorganization. If you don't lock and the file is well-used, you can count on being down for reorganization several times a day, so don't take any chances.

How about locking when reading? The KSAM manual will give you the bad news; when doing a keyed sequential read you should lock before the "start" and hold the lock until after the last read. The reason is simple. If you don't, someone else may come along and change the key block that you are using on the disc and you have no way of knowing it. Several records will apparently vanish into thin air. So, if it is important to know that you read all the records (and it usually is) lock the file.

I suppose you are hoping that I will at least let you off for a read-by-key. After all, the KSAM manual doesn't say anything about locking in this case, so by implication it should be unnecessary --- right? Wrong! Yes, most of the time you will get away with it, but now and then it will go wrong.

What is the explanation? Unlike IMAGE, KSAM makes use of the ordinary HP intrinsics without any special protection. When you ask for a read-by-key you will actually be doing one disc I/O for each key record before reading the data record. If you don't have a lock, someone else can change the key file in between these reads and again your record will apparently disappear into thin air. However, there is a little relief; this time, you know about it; you will get a "record not found" error. You may have good reason to be sure that it does exist, in which case you know you have a problem. So, if you don't want to lock on a read-by-key, put in a little extra code. If you can't find the record on the first unlocked read-by-key, lock the file and try again. This way you avoid the performance degradation associated with locking most of the time, but can retrieve the record infallibly when needed. I am indebted to Lisa Burns for explaining this particular peculiarity to me.

This is one of those places that "FCONTROL" comes in useful. Before doing the unlocked read by key use an "FCONTROL" with a control code of 7. This which will clear the buffers before reading, thereby improving your chances of finding the record that you are looking for.

Actually this situation of losing records during an unlocked read can also arise on an IMAGE detail dataset (see Thoughts and Discourses on HP3000 Software by Eugene Volokh pp. 180-185). The reason that it occurs less frequently is directly related to the locality of the keys in an IMAGE dataset. A put to an IMAGE dataset will affect only other records with the same key; a write to a KSAM file can affect the keys to any of the other records in the whole file.

4. WAYS TO USE KSAM

KSAM thus has certain advantages that should make it the method of choice in many situations. Good examples of this are systems that require a sorted, often alphabetic key and situations with transactions, particularly batched transactions, that go through various stages of processing. These will now be described.

4.1. Alphabetical Sort

As an example of a common use made of KSAM, we will consider a customer file that has as its primary key a customer number. Because this number may sometimes be unknown or unavailable, we want to access the record alphabetically by name. The common way to solve this problem is to use an IMAGE master dataset with the customer number as its key and to build a KSAM index file containing the customer number as a primary key and the name as a secondary sort key. In this way a name can be looked up on the KSAM file and the data retrieved from the master dataset.

Sometimes the dataset needs more than one key, maybe a zip-code or a telephone number. This necessitates the data being put in an IMAGE detail dataset with several automatic masters and a KSAM index file constructed in a similar manner. The most efficient way of doing this is to use the record number as the primary key so that direct access is made to the record desired.

However, this mixture of KSAM and IMAGE is unnecessary in many cases. Why not put all the data into the KSAM file and dispense with the IMAGE datasets altogether? In the case of the multiple keys, all the secondary keys can be put in the KSAM file. This saves a lot of unnecessary updating when creating or changing the file and unnecessary reads when accessing the data. The uses made of the file should be studied to determine the best method. If it is read frequently by customer number and changed infrequently, then an IMAGE dataset may be valuable, particularly if it is kept as a master. But frequent changes, particularly to key values, militate against an IMAGE dataset.

4.2. Transaction Control

One system design problem that I have seen frequently is the access method to be used for a batch transaction file. Such a file usually has groups of transactions, each transaction represented by a single record. Sometimes, the actual data is kept elsewhere, and the transaction file is used solely as a way of keeping track of the processing. Usually the file contains some kind of status field that indicates the status of the transaction such as "new", "posted", "suspended", etc. depending on the application. The status is used as a key so that processing programs can read this key to determine the records to be processed. However, this key must then be updated to indicate the change in status. This process is usually accomplished with a detail dataset, often with several paths, but I have also seen it done with multiple flat files.

The big problem with using a detail dataset is that changing the status requires a delete and a put, which are both very time-consuming. With several paths through the dataset, the operation becomes particularly inefficient. But the key is needed, for without it there would be no reasonable way of finding the desired transactions.

Using MPE flat files tends to be an even greater problem than using detail datasets, particularly if additional key data is needed, such as processing date. This often requires concocting complex file-naming conventions to determine which file is which. Processing requires a large number of file opens and closes, and, worse still, this scheme essentially uses the system directory as a key file. Since the directory is single-threaded, this method is likely to have a major impact on the overall system performance.

Using KSAM for the transaction file solves all these problems. If we make the status a secondary key, we can access the data just as easily as we could by using a path through an IMAGE dataset; but we can change it with a minimal amount of disc I/O. Furthermore, we may be able to group our updates, such as putting together all the suspended transactions in a certain batch, thereby making the updating even more efficient.

An added advantage of using KSAM is that the key, as well as containing the status, can contain priority fields. Date is the obvious field to add to the status field, but we can also add a priority based on urgency of processing. Rather than a simple FIFO queue, we can then insert transactions into the queue at any point desired, without any performance deterioration.

4.3 Text Library

A very interesting application of KSAM is to use it to construct a text library. The most obvious example is a COBOL copy library, which is set up as a KSAM file.

This is a case in which a duplicate primary key can be used, although this is not the case with copylibs. The disadvantage of a duplicate primary key is that it is difficult to delete a specific record and there is absolutely no way to insert a new record between two others. But if all the records for a particular key are deleted and then rewritten, this is no longer of concern.

In a case such as this, one can consider all the records with the same key as being like a single variable-length record that is changed as a whole. There is no updating of records.

One interesting aspect of an application like this is ensuring that there is enough room on the file for the new data to be written. This can be easily determined by using the FGETINFO intrinsic.

5 CONCLUSION

KSAM has been very much neglected, possibly because not too many people feel confident enough about using it. In fact, it can be easier to use than IMAGE and can offer advantages both in capabilities and in performance that should not be ignored.

```
    select queue-file
      assign to "queue.file"
      organization is indexed
      access mode is dynamic
      record key is key5
      alternate record key is key1 with duplicates
      alternate record key is key2 with duplicates
      file status is ksam-status.

    ..........

file section.

fd  queue-file.
01  queue-recd.
    05  delete-word                          pic xx.
    05  key1.
        10 .......

    05  key2.
        10 .......

        10  key5 .........

    ..........

working storage.

01  ksam-status.
    05  ksam-status-1                        pic x.
    05  ksam-status-2                        pic x.
01  ksam-result                              pic s9(4).
```

Fig. 1   Setting up a KSAM File

```
procedure division.

     ..........

     open i-o queue-file.

     .........

     close queue-file.

     .........

     move spaces to queue-recd.
     move "WD001" to key2.
     start queue-file
       key is not less than key2
       invalid key,
         move high-value to eof-flag
         go to x.

      .........

     read queue-file next record
       at end
         move high-value to eof-flag
         go to x.

     if key2 not equal to "WD001"
       move high-value to eof-flag
       go to x.

     .........

     exclusive queue-file.

     .........

     rewrite queue-recd
       invalid key,
         display "Error Rewriting Queue-Recd ", key5
         call "CKERROR" using ksam-status, ksam-result
         call intrinsic "QUIT" using ksam-result.

     ..........

     un-exclusive queue-file.
```

Fig. 2   Using COBOL Verbs

```
01  link-area.
       05  queue-file-area.
          10  file-number                          pic s9(4) comp.
          10  file-name                            pic x(8).
          10  i-o-type                             pic s9(4) comp.
          10  a-mode                               pic s9(4) comp.
          10  prev-op                              pic s9(4) comp.

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


     move "QUEUE" to file-name.
     move zero to file-number,
                  i-o-type,
                  prev-op.
     move 2 to a-mode.
     call "CKOPENSHR" using queue-file-area,
                  ksam-status.

     .............
```

Fig. 3   Using CK Intrinsics

```
>build queue;
>  rec=-40,25,f,ascii;
>  disc=10000;&
>  keyfile=queuek;&
>  key=a,25,10;&
>  key=a,3,14,,duplicate
```

```
fcopy from=filea;to=(fileb,filek);key=0
```

Fig. 4   Using KSAM Utilities

KSAM	IMAGE
"Global" Keys	"Local" Keys
"Global" key damage	"Local" key damage
Sequential by key	Non-sequential by key
Overlapping keys	No overlapping keys
Partial keys	No partial keys
Low overhead "sorted chains"	High- overhead sorted chains
Access loosely controlled	Access tightly controlled
Locking not controlled	Locking tightly controlled
Locking by dataset only	Locking by key and dataset
No logging and recovery	Logging and recovery
Secondary key update	No secondary key update

Table I. KSAM vs. IMAGE

K S A M -- A D I F F E R E N T A P P R O A C H

DARRELL CONGER
AMARILLO COLLEGE
BOX 447
AMARILLO TEXAS 79178
806-371-5112
806-371-5370 (FAX)

B A C K G R O U N D I N F O R M A T I O N

AMARILLO COLLEGE converted from UNIVAC to HEWLETT-PACKARD in 1979. (SEE JOURNAL OF HP 3000 INTERNATIONAL USERS GROUP VOL.6, NO.3 JULY/SEPT 1983) The choices for the conversion were:

1. Convert the ISAM database to HP'S IMAGE;
2. Convert the ISAM database to HP'S KSAM and then later convert to IMAGE.

NOTE:

A. Approximately 600 UNIVAC COBOL programs needed conversion.
B. Approximately 120 UNIVAC ASSEMBLER communication programs had to be converted to HP'S COBOL.

Our first choice was ISAM to KSAM with a conversion to HP'S IMAGE at a later date. The ISAM TO KSAM conversion led to the development of the KSAM I/O handler and supporting COBOL standards. We felt it was worth the change from the standard indexed I/O COBOL coding (see implementation considerations) to a format that closely follows an IMAGE implementation. We hoped that KSAMIO would provide the following:

1. An organized "shorthand" method of coding COBOL programs for KSAM files while providing a format for a future decision to convert to IMAGE.

2. Program development and maintenance would concentrate on PROCEDURE DIVISION logic rather than on accessing the database.

3. Environment promoting new program development by the very nature of how easy it might be to implement applications with the KSAMIO tool.

Conversion using the I/O handler resulted in the major campus applications PAYROLL, PERSONNEL, BUSINESS, REGISTRAR and FINANCIAL AIDS using the KSAM routine for access to the integrated KSAM databases. Currently there are over 300 KSAM files on our HP-3000 Series 70. About half have a primary key with most of the remaining having both a primary and a secondary key. There are a few with three keys, and one has five keys. Record lengths vary from 20 to 2048 bytes while key lengths vary from 1 to 46 bytes.

O V E R V I E W

KSAMIO is a COBOL subroutine that uses HEWLETT-PACKARD'S COBOL "CK" PROCEDURES to access the KEYED SEQUENTIAL ACCESS METHOD (KSAM) file organization. It replaces the COBOL II INDEXED MODULE coding. It has been used as a conversion tool to aid in converting ISAM to HEWLETT PACKARD'S KSAM and also as an ongoing development tool for access to KSAM files. Some of its provisions are:

- Automatic "RECORD LOCKING" using RANDOM or POINTER reads with REWRITE functions.
- Implements HEWLETT-PACKARD COBOL "CK" (KSAM) PROCEDURE
- Offers COBOL programmers:
  1. Organized structure for coding COBOL statements.
  2. Ease of identifying system wide changes in an application and types of file access.
  3. Flexible and dynamic accession to KSAM files.
- Programmatic changes in:
  Input/Output file type
  Access mode
  Key position and length.
- Access to multiple unique and repeating keys.
- Uses the WHO and DATELINE INTRINSICS.
- CHRONOLOGICAL access. (FREADC)
- Automatic "ZSIZE" INTRINSIC on KSAM file CLOSES.
- "TRACE" of KSAMIO functions.
- Enables an optional "CONTROL Y" escape capability.
- Standard KSAM file organization and a "SPECIAL" KSAM file automatic key assignment record creation.
- Supports a character mode screen handler.
- Access to COPYLIB KSAM files.

The programming characteristics are:

- Eliminates COBOL ENVIRONMENT statements and DATA DIVISION statements related to INDEXED files.
  (REPLACES COBOL II'S INDEXED I/O MODULE CODING)
- REQUIRES only WORKING-STORAGE SECTION entries.
- Uses a FILE TABLE with each COBOL RECORD description.
- Uses COPYLIBS.
- COBOL CALL statement used for KSAMIO functions.
- User ERROR CHECKING in COBOL FILE STATUS CLAUSE.
- Automatic file OPENING on initial I/O function.
- Automatic "LOCKING/UNLOCKING" for "WRITES, REWRITES, and DELETION" functions.
- User choice for LOCK/UNLOCK rationale on "READS".
- "RECORD LOCKING" is available through two of the functions for applications that require "overwrite" protection.

The routine operates in the following environment:

GENERAL
  HEWLETT-PACKARD 3000 SYSTEMS
  HEWLETT-PACKARD SOFTWARE:
    COBOL OR COBOL II
    COBOL "CK" KSAM PROCEDURES
    KSAM
    KSAMUTIL
    COPYLIB

AT AMARILLO COLLEGE
- COBOL II (FROM HP)
- KSAM (FROM HP)
- KSAMIO (FROM AMARILLO COLLEGE)
- FCOPY (FROM HP)
- KSAMUTIL (FROM HP)
- SORT (FROM HP)
- QUIZ (FROM COGNOS)
- MPEX (FROM VESOFT)
- QEDIT (FROM ROBELLE)
- INTERFACE (FROM SATCOM)

I N D E T A I L

KSAMIO is a subroutine that resides in the system SL. The COBOL "CK" PROCEDURES (see Appendix A) use parameters (see Appendix B) which are passed to the subroutine. Parameters included are:

1. FILE TABLE;
2. FUNCTION;
3. KEY (if needed);
4. KEY-RELATION (if needed); and
5. RECORD (if writing or re-writing).

Information passed back to the "calling" program includes an error status and record depending on the function requested. There are currently nineteen functions in KSAMIO (JAN 1990). Thirteen of these functions use one or more of the twelve COBOL "CK" PROCEDURES. (See FUNCTIONS in the Appendix D). These functions require parameter information in order to execute the COBOL "CK" PROCEDURES. Some of the parameters are set in the FILE TABLE statements that precede the COBOL record description. Other parameters are set with COBOL STATEMENTS. EXAMPLE --> MOVE SOC-SEC-NO TO KIO-KEY.

The following are steps used at AMARILLO COLLEGE to add another KSAM file and write an update screen program:

1. Define the record description. (see COA-RECORD Appendix C)
2. Put record description into the RECORD COPYLIB.
```
(RUN COBEDIT.PUB.SYS)
>LIB RLIB99DF.PUB.COMPCNTR
>EDIT
/(ADD RECORD STATEMENTS)
/EXIT
>KEEP ???-RECORD
>EXIT
```
3. Put the KSAM file table in the FILE COPYLIB.
```
(RUN COBEDIT.PUB.SYS)
>LIB FLIB99DF.PUB.COMPCNTR
>EDIT FILE (TEXT IN PROTOTYPE)
/(MAKE CHANGES - SEE COAFILE Appendix C)
/E
>KEEP ???FILE
```
4. Use KSAMUTIL to create the KSAM key and data files.
```
!RUN KSAMUTIL.PUB.SYS
BUILD COA99DF;REC=-64,32,F,ASCII;DISC=4000,10,1;&
KEYFILE=COA99KF;KEY=B,41,6;KEY=B,1,46;KEYENTRIES=6000
EXIT
!EOJ
```

5. **Define the update screen.**

   NOTE: **Amarillo College uses a character mode screen handler developed for internal use. The handler uses KSAMIO for maintenance to the KSAM screens file. Each session program calls a subroutine for screen functions.**

6. **Insert the screen into the KSAM screen file.**

   ```
   RUN TSCR01CX.PUB.COMPCNTR
   ```

   NOTE: **TSCR01 is the maintenance program for adding, changing, deleting, printing, and scanning screen records. One of the standards at Amarillo College is that all screens display data elements with descriptions on the first twenty-one lines while the twenty-second line is for data entry which "pops" the data on the screen with numbered data elements for update recall.**

7. **Use a prototype program that already conforms to screen update procedures and standards.**

   ```
   QEDIT
   /T NEWPROGRAM = (GOOD EXAMPLE)
   /(MAKE CHANGES)
   /EXIT
   ```

8. **Test program.**

All of the "CK" PROCEDURES are implemented in KSAMIO. (See the HP manual "KSAM/3000 REFERENCE MANUAL" part no 30000-90079 for a detailed description of these procedures). Each "CK PROCEDURE" with a brief description follows:

PROCEDURE	DESCRIPTION
CKCLOSE	terminates file processing
CKDELETE	deletes a record
CKERROR	file system error returned
CKLOCK	dynamically locks file
CKOPEN	initiates file processing
CKOPENSHR	same as CKOPEN with dynamic locking/shared access
CKREAD	makes next logical record available
CKREADBYKEY	makes record available by key value
CKREWRITE	replaces record having matching primary key
CKSTART	positions record pointer by primary or alternate key
CKUNLOCK	unlocks a locked file dynamically
CKWRITE	copies record in program's data area to I/O file

The following TABLE shows the relationship between COBOL II'S INDEXED I/O MODULE VERBS, COBOL "CK" PROCEDURES, and MPE'S INTRINSICS.

COBOL VERBS and modifiers	COBOL PROCEDURES	MPE INTRINSICS
OPEN filename	CKOPEN	FOPEN
		CKOPENSHR
CLOSE filename	CKCLOSE	FCLOSE
DELETE recordname	CKDELETE	FREMOVE *
(NO COBOL EQUIVALENT)	CKERROR	FCHECK
		FERRMSG
		PRINTFILEINFO
EXCLUSIVE filename	CKLOCK	FLOCK
READ filename		FREAD
READ filename KEY IS	CKREADBYKEY	FREADBYKEY *
READ filename NEXT RECORD	CKREAD	FREAD
REWRITE recordname	CKREWRITE	FUPDATE
START filename KEY IS	CKSTART	FFINDBYKEY *
UNEXCLUSIVE filename	CKUNLOCK	FUNLOCK
WRITE recordname	CKWRITE	FWRITE

* INTRINSICS FOR KSAM FILES ONLY

A description of each parameter (COBOL name in parentheses) follows:

FILE TABLE (KIOFILE)

The FILE TABLE (SEE Appendix C) provides information used by the COBOL "CK" PROCEDURES. The table precedes its related COBOL record description in the WORKING-STORAGE SECTION. The following lists the table entries with their COBOL coding clauses, initial values, and any "hard fast" rules that are mandatory for execution.

1. FILE NUMBER . . . . PIC S9(04) COMP VALUE 0.

   The "CK" PROCEDURES use this to identify the correct file. It is always set initially to zeros and never changed programmatically!

2. FILE NAME . . . . PIC X(08) VALUE '????????'.

   This name is the actual name of the KSAM file or whatever is equated to the actual file. Can be any combination of characters that follow the HEWLETT-PACKARD file naming rules.

   NOTE: By setting a standard scheme for naming KSAM files you will be able to access different files of the same format programmatically.
   EXAMPLE: SCR871DF, SCR872DF, SCR873DF
   where the 4th, 5th, and 6th positions reflect the year and semester code.

3. FILE TYPE I/O . . . . PIC S9(04) COMP VALUE ?.

   Value is: 0=input only; 1=output only; 2=input/output. Will always dictate the COBOL "CK" PROCEDURES that are executable. (0=input --> "READS" only; etc).

4. FILE ACCESS MODE . . . . PIC S9(04) COMP VALUE ?.

   Value is: 0=sequential; 1=random; 2=dynamic (both). Dictates the COBOL "CK" PROCEDURES that are "executable". (1=random implies only "RANDOM READS"; etc).

5. PREVIOUS OPERATION . . . . PIC S9(04) COMP VALUE ZEROS.

   Set and checked by each COBOL "CK" PROCEDURE. Is set to zeros initially and never changed programmatically.

6. RECORD SIZE . . . . PIC S9(04) COMP VALUE ?.

   Size of logical record remains as set initially for most instances but can be programmatically changed.

7. KEY LOCATION . . . . PIC S9(04) COMP VALUE ?.

   Primary key starting position in record. Will normally be left as initially set but, if alternate key access is needed then can be programmatically changed.

8. KEY LENGTH .... PIC S9(04) COMP VALUE ?.

Length of primary key. Is normally set for the PRIMARY KEY but again may be changed programmatically. (changing length can be used to "CKSTART" then "CKREAD" on a shortened key)

9. LOCK CONDITION .... PIC S9(04) COMP VALUE ?.

Determines program action if file is locked by another user. 0=control returns if already locked; 1=program suspends until it can be locked. LOCK CONDITION depends on what type of environment and if file is shareable to all users. A setting of ZERO indicates thst you will programmatically check for the lock condition and if "locked" will alter your logic accordingly while a setting of ONE suspends your program until the file is unlocked by the current user.

10. RECORD LOCK POSITION .... PIC S9(04) COMP VALUE ?.

Location of RECORD LOCK BYTE in COBOL record description. Can be part or all of a field already being used or may be a data element reserved just for saving the record lock code. For new records the last position might always be used.

11. TYPE OF ACCESS .... PIC X(01) VALUE ?.

Code determines "how" file is opened. S=Opened as shared file while E=Opened as non-share (exclusive). TYPE OF ACCESS dictates "how" the file will be "OPENED". A value of "S" opens the file as shareable while a value of "E" opens as non-shareable (exclusive) access.

12. OPEN/CLOSE CODE .... PIC X(01) VALUE ?.

Always should be initially = "C". Used to determine open/close logic for various COBOL "CK" PROCEDURES. A "closed" file is always opened for you "automatically".

13. TYPE OF KSAM FILE .... PIC X(01) VALUE ?.

KSAMIO uses two types of KSAM files. R=Regular (normal) and M=Maintenance Pool (where next record added is automatically given the next sequential numbered key - ie; the chronological order is the same as the primary key order).

14. RECORD LOCK BYTE .... PIC X(01) VALUE ' '.

Storage byte for the position in the data record that is being used to set the record lock code "~". Before REWRITE FUNCTION is completed byte is moved back to the data record.

NOTE: User considerations for setting the "FILE TABLE" are:

1. Set the "normal" values in the table.
2. Change the values at execution time with program logic for dynamic file processing.

FUNCTIONS (KIO-FUNCTION)

KSAMIO is function driven (See FUNCTIONS in Appendix D). The functions are users of the COBOL "CK" PROCEDURES and/or other intrinsics. Most program development uses a subset of the functions such as open, close, some type of reads, and either write or rewrite.

KEY (KIO-KEY)

KSAMIO uses a key for the "RANDOM" AND "POINTER" reads. The key is defined in the FILE TABLE with a starting position and length definition. The key may start anywhere in the record and have a maximum length of 50. The starting position and length may be changed dynamically for the various options within the "READ" functions. An example of a COBOL statement for setting a key is: MOVE SOC-SEC-NO TO KIO-KEY.

KEY RELATION (KIO-KEY-RELATION)

The "POINTER READ" needs one of the following conditions set in order to complete the read: 1. EQUAL, 2. GREATER, or 3. EQUAL OR GREATER. A partial key read may also be used by changing the "LENGTH" of the key within the FILE TABLE. An example of pointer reading the ZIP file for only those ZIPS with first 3 digits greater than 790 to display ZIP CODE AND ZIP NAME follows:

```
     MOVE 79000 TO ZIP-CODE.
     MOVE 3 TO ZIP-LENGTH.
     PERFORM POINTER-READ-ZIP UNTIL KIO-ERROR-BYTE-1 = '1'.
     .
     .
     .

 POINTER-READ-ZIP.
     MOVE ZIP-CODE TO KIO-KEY.
     COPY KSAM-PR IN KSAMLIB NOLIST REPLACING ==???== BY ==ZIP==
     ==??== BY ==GREATER==.
     DISPLAY ZIP-CODE ' ' ZIP-NAME.
```

RECORD (KIO-RECORD)

The RECORD is a COBOL description of the format of the KSAM file record. Each KSAM file record to be used by KSAMIO must have a FILE TABLE preceding the record description. The record is made available to the programmer by each COBOL "CK" PROCEDURE that uses the input/output area (KIO-RECORD).

```
01  COAFILE.
    02  FILE-NUMBER                                  PIC  S9(4) COMP VALUE 0.
    02  FILE-NAME.
        03  FILE-NAME-PREFIX                         PIC  X(03) VALUE 'COA'.
        03  FILE-NAME-YEAR                           PIC  X(02) VALUE '99'.
        03  FILE-NAME-SUFFIX                         PIC  X(03) VALUE 'DF '.
    02  TYPE-IO                                      PIC S9(04) COMP VALUE 2.
    02  ACCESS-MODE                                  PIC S9(04) COMP VALUE 2.
    02  PREVIOUS-OPER                                PIC S9(04) COMP VALUE 0.
    02  RECORD-SIZE                                  PIC S9(04) COMP VALUE 64.
    02  KEY-LOCATION                                 PIC S9(04) COMP VALUE 41.
    02  KEY-LENGTH                                   PIC S9(04) COMP VALUE 6.
    02  LOCK-CONDITION                               PIC S9(04) COMP VALUE 1.
    02  RECORD-LOCK-POSITION                         PIC S9(04) COMP VALUE 64.
    02  TYPE-ACCESS                                  PIC  X(01) VALUE 'S'.
    02  OPEN-CLOSE                                   PIC  X(01) VALUE 'C'.
    02  TYPE-KSAM-FILE                               PIC  X(01) VALUE 'R'.
    02  RECORD-LOCK-BYTE                             PIC  X(01) VALUE ' '.

COPY COA-REC IN RECDLIB.
```

The above table is an expansion of a COBOL COPY USAGE module. The COBOL 02 levels represent the HP COBOL KSAM "CK" PROCEDURES that "drive" the intrinsic logic. The prefix COA is an acronym defining the CHART OF ACCOUNTS at AMARILLO COLLEGE and is an example of the NAMING scheme for COBOL element names within the record and file descriptions. Note the last statement being another COBOL COPY USAGE that copies in the corresponding COBOL record description associated with the TABLE.
See below for the COBOL record description.

```
01  COA-RECORD.
    02  COA-SECONDARY-KEY.
        03  COA-NAME                                 PIC  X(40).
        03  COA-PRIMARY-KEY                          PIC  X(06).
        03  COA-ACCOUNT-NUMBER REDEFINES COA-PRIMARY-KEY  PIC  9(06).
        03  COA-DEPARTMENT-NUMBER REDEFINES COA-PRIMARY-KEY.
            04  COA-FUND-FUNCTION.
                05  COA-FUND                         PIC  9(01).
                05  COA-FUNCTION                     PIC  9(02).
            04  COA-UNIT.
                05  COA-AREA                         PIC  9(01).
                05  COA-SUBDEPT                      PIC  9(02).
        03  COA-KEY2 REDEFINES COA-PRIMARY-KEY.
            04  FILLER                               PIC  X(03).
            04  COA-OBJECT-CODE.
                05  COA-OBJECT-1-2.
                    06  COA-OBJECT-1ST               PIC  9(01).
                    06  COA-OBJECT-2ND               PIC  9(01).
                05  COA-OBJECT-3RD                   PIC  9(01).
        03  COA-AREA-KEYS REDEFINES CDA-PRIMARY-KEY.
            04  FILLER                               PIC  X(01).
            04  COA-GL-KEY.
                05  FILLER                           PIC  X(01).
                05  COA-CASH-POSTING-NO.
                    06  FILLER                       PIC  X(02).
                    06  COA-DIVISION-KEY.
                        07  COA-EXECUTIVE-CODE       PIC  X(01).
                        07  FILLER                   PIC  X(01).
    02  COA-DIVISION-CODE                            PIC  X(02).
    02  COA-BANK-ACCOUNT-NO                          PIC  X(10).
    02  COA-GENERAL-LEDGER-NO                        PIC  X(05).
    02  COA-ACTIVITY-CODE                            PIC  X(01).
```

Each function will be described with the following format:

NAME OF FUNCTION:

1. Short description.
2. COBOL COPY statement example.
3. EXPANSIONS.
4. Comments.

INTRINSICS:

1. Establish the "DATELINE" AND "WHO" intrinsics.
2. COPY KSAM-CI IN KSAMLIB NOLIST.
3.
```
MOVE ' ' TO CONTROL-Y-SWT FIRST-TIME-SWT.
MOVE 'N' TO TRACE-SWT STDIN-OPEN-SWT STDLIST-OPEN-SWT.
MOVE ZEROS TO RESULT DISPLAY-DECIMAL EXTERNAL-LABEL RETURN-LABEL RECORD-LENGTH
        STACK-SIZE CALL-CONDITION STDIN-FILE-NUMBER STDLIST-FILE-NUMBER
        STDLIST-CONTROL WAIT-ITERATION FIRST-RECORD-NUMBER.
MOVE SPACES TO SAV-ERROR-BYTES.
MOVE CALL-INTRINSICS TO KIO-FUNCTION.
PERFORM CALL-KSAMIO.
```
4. Used as first statement or one of the first statements in the PROCEDURE DIVISION. Provides ability to check LOGON information and date/time related information.

CHRONOLOGICAL READ:

1. Reads KSAM file records in chronological order.
2. COPY KSAM-CR IN KSAMLIB NOLIST REPLACING = =??? = = BY = =COA= =.
3.
```
MOVE CHRONOLOGICAL-READ TO KIO-FUNCTION.
MOVE ???FILE TO KIO-FILE.
PERFORM CALL-KSAMIO.
MOVE KIO-RECORD TO ???-RECORD.
MOVE KIO-FILE TO ???FILE.
```
4. Cuts COBOL SORT extraction read time by ignoring the sequential KEY chained read.

NOTE: KSAMIO copy clauses use the COBOL REPLACING phrase. The "text-to-replace" is represented by ?? or ??? depending upon the function requested. Values for the ?? are EQUAL, GREATER, or EQUAL-GREATER which are used in the pointer read key relationship. The "replace-text" for ??? is used as a prefix for naming the FILE and RECORD COBOL descriptions in KSAMIO.

EXAMPLE: ???FILE AND ???-RECORD use an acronym prefix for the Chart of Accounts application at Amarillo College resulting in a naming scheme of COAFILE AND COA-RECORD.

SEQUENTIAL READ:

1. Reads a KSAM file sequentially starting with first record.
2. COPY KSAM-SR IN KSAMLIB NOLIST REPLACING = =??? = = BY = =COA= =.
3.
```
MOVE SEQUENTIAL-READ TO KIO-FUNCTION.
MOVE ???FILE TO KIO-FILE.
PERFORM CALL-KSAMIO.
MOVE KIO-RECORD TO ???-RECORD.
MOVE KIO-FILE TO ???FILE.
```
4. Records are read in sequential order by key value.

RANDOM READ:

1. Reads a KSAM file randomly.
2. COPY KSAM-RR IN KSAMLIB NOLIST REPLACING = =??? = = BY = =COA= =.
3. MOVE RANDOM-READ TO KIO-FUNCTION.
   MOVE ???FILE TO KIO-FILE.
   PERFORM CALL-KSAMIO.
   MOVE KIO-RECORD TO ???-RECORD.
   MOVE KIO-FILE TO ???FILE.
   IF KIO-ERROR-BYTE-1 NOT = '0' MOVE SPACES TO ???-RECORD.
4. Any key created with KSAMUTIL may be read by changing the KEY LOCATION and setting the LENGTH in the FILE TABLE.

LOCK RANDOM UPDATE:

1. Reads KSAM file records randomly, sets RECORD LOCK CODE, then REWRITES record with lock code.
2. COPY KSAM-LRU IN KSAMLIB NOLIST REPLACING = =??? = = BY = =COA= =.
3. MOVE LOCK-RANDOM-UPDATE TO KIO-FUNCTION.
   MOVE COAFILE TO KIO-FILE.
   PERFORM CALL-KSAMIO.
   MOVE KIO-RECORD TO COA-RECORD.
   MOVE KIO-FILE TO COAFILE.
4. User should remember to "always" REWRITE the record or other users of KSAMIO can not read the record.

POINTER READ:

1. Start reading a KSAM file with the COBOL "CKSTART" PROCEDURE which sets the pointer to the first record to be read using a parameter of equal to, or greater than, or both, and then uses the "CKREAD" PROCEDURE.
2. COPY KSAM-PR IN KSAMLIB NOLIST REPLACING = =??? = = BY = =COA= =
   = =?? = = BY = =GREATER= =.
3. MOVE POINTER-READ TO KIO-FUNCTION.
   MOVE ???FILE TO KIO-FILE.
   MOVE ??-KEY TO KIO-KEY-RELATION.
   PERFORM CALL-KSAMIO.
   MOVE KIO-RECORD TO ???-RECORD.
   MOVE KIO-FILE TO ???FILE.
4. A partial key read is available by setting the key length to the desired value in the FILE TABLE.

LOCK POINTER UPDATE:

1. Start reading a KSAM file with the COBOL "CKSTART" PROCEDURE which sets the pointer to the first record to be read using a parameter of equal to, or greater than, or both and then uses the "CKREAD" PROCEDURE. Also sets the RECORD LOCK CODE by rewriting record.
2. COPY KSAM-LPU IN KSAMLIB NOLIST REPLACING = =??? = = BY = =COA= =
   = =?? = = BY = =GREATER= =.
3. MOVE LOCK-POINTER-UPDATE TO KIO-FUNCTION.
   MOVE COAFILE TO KIO-FILE.
   MOVE GREATER-KEY TO KIO-KEY-RELATION.
   PERFORM CALL-KSAMIO.
   MOVE KIO-RECORD TO COA-RECORD.
   MOVE KIO-FILE TO COAFILE.
4. User should remember to "always" REWRITE the record or other users of KSAMIO can not read the record.

WRITE RECORD:

1. Write a KSAM file record.
2. COPY KSAM-W IN KSAMLIB NOLIST REPLACING = =???= = BY = =COA= =.
3.
```
MOVE WRITE-REC TO KIO-FUNCTION.
MOVE ???-RECORD TO KIO-RECORD.
MOVE ???FILE TO KIO-FILE.
PERFORM CALL-KSAMIO.
MOVE KIO-FILE TO ???FILE.
```
4. Records are written in chronological sequence to the KSAM DATA file while the KSAM KEY file is updated dynamically.
   NOTE: If the type KSAM file is coded "M" then the system returns to the program the "pointer record" which contains the key of the last record written to the KSAM file.

REWRITE RECORD:

1. Rewrite a KSAM file record.
2. COPY KSAM-RW IN KSAMLIB NOLIST REPLACING = =???= = BY = =COA= =.
3.
```
MOVE REWRITE-REC TO KIO-FUNCTION.
MOVE ???-RECORD TO KIO-RECORD.
MOVE ???FILE TO KIO-FILE.
PERFORM CALL-KSAMIO.
MOVE KIO-FILE TO ???FILE.
```
4. Rewrite is preceded by a LOCK and followed by an UNLOCK.

DELETE RECORD:

1. Deletes KSAM file records.
2. COPY KSAM-DLT IN KSAMLIB NOLIST REPLACING = =???= = BY = =COA= =.
3.
```
MOVE DELETE-REC TO KIO-FUNCTION.
MOVE ???FILE TO KIO-FILE.
PERFORM CALL-KSAMIO.
MOVE KIO-FILE TO ???FILE.
```
4. KSAM puts a code of binary 1's in the first two positions of the record. You may want to design your record formats where the first two positions are FILLER in order to recover deleted records.

OPEN FILE:

1. Opens KSAM file.
2. COPY KSAM-OF IN KSAMLIB NOLIST REPLACING = =???= = BY = =COA= =.
3.
```
MOVE CHECK-OPEN-FILE TO KIO-FUNCTION.
MOVE ???FILE TO KIO-FILE.
PERFORM CALL-KSAMIO.
MOVE KIO-FILE TO ???FILE.
```
4. Opening a file checks on the type access and opens with CKOPEN OR CKOPENSHR. A file is automatically opened the first time another function is used against the file.

CLOSE FILE:

1. Closes KSAM file.
2. COPY KSAM-CF IN KSAMLIB NOLIST REPLACING = =???= = BY = =COA= =.
3.
```
MOVE CLOSE-FILE TO KIO-FUNCTION.
MOVE ???FILE TO KIO-FILE.
PERFORM CALL-KSAMIO.
MOVE KIO-FILE TO ???FILE.
```
4. Closing a file with a reopen allows access mode or input/ouput type changes.

LOCK FILE:

1. Locks KSAM file.
2. COPY KSAM-LF IN KSAMLIB NOLIST REPLACING ==???== BY ==COA==.
3.
```
MOVE LOCK-FILE TO KIO-FUNCTION.
MOVE ???FILE TO KIO-FILE.
PERFORM CALL-KSAMIO.
MOVE KIO-FILE TO ???FILE.
```
4. Locking a file is used when file access is shared and the file opened with CKOPENSHR.

UNLOCK FILE:

1. UnLocks KSAM file.
2. COPY KSAM-ULF IN KSAMLIB NOLIST REPLACING ==???== BY ==COA==.
3.
```
MOVE UNLOCK-FILE TO KIO-FUNCTION.
MOVE ???FILE TO KIO-FILE.
PERFORM CALL-KSAMIO.
MOVE KIO-FILE TO ???FILE.
```
4. Unlocking a file is required when file has been locked and is used when file is opened with CKOPENSHR.

TRACE ON:

1. TRACE ON is used to display the executing KSAMIO function.
2. COPY KSAM-TON IN KSAMLIB NOLIST.
3.
```
MOVE TRACE-ON TO KIO-FUNCTION.
PERFORM CALL-KSAMIO.
```
4. Setting "TRACE ON" enables the programmer to identify if and when each KSAMIO function executes. Works in conjunction with the "TRACE OFF" function.

TRACE OFF:

1. Sets "TRACE OFF".
2. COPY KSAM-TOF IN KSAMLIB NOLIST.
3.
```
MOVE TRACE-OFF TO KIO-FUNCTION.
PERFORM CALL-KSAMIO.
```
4. TRACE OFF terminates the "TRACE ON" function. Both may be used anywhere in the PROCEDURE DIVISION.

CONTROL Y:

1. CONTROL Y enables user program termination control.
2. COPY KSAMCTLY IN KSAMLIB NOLIST.
3.
```
MOVE CONTROL-Y TO KIO-FUNCTION.
PERFORM CALL-KSAMIO.
```
4. "CONTROL Y" may be set for program development but deleted for production.

READ TERMINAL:

1. Enables a "FOPEN" AND "FREAD" for "$STDIN".
2. COPY KSAM-TRD IN KSAMLIB NOLIST.
3.
```
MOVE READ-TERMINAL TO KIO-FUNCTION.
PERFORM CALL-KSAMIO.
```
4. "READ TERMINAL" is used to accept input from a terminal.

WRITE TERMINAL:

1. Enables a "FOPEN" AND "FWRITE" for "$STDLIST".
2. COPY KSAM-TWR IN KSAMLIB NOLIST.
3. MOVE WRITE-TERMINAL TO KIO-FUNCTION.
   PERFORM CALL-KSAMIO.
4. Used to write ouput to a terminal.

COBOL II'S INDEXED I/O MODULE coding is shown first.

```
-----------------------------------------------------------
$CONTROL QUOTE=',USLINIT
 IDENTIFICATION DIVISION.
 PROGRAM-ID. TCOA99CS.
 REMARKS. RANDOM READ/REWRITE OF CHART-OF-ACCOUNTS KSAM FILE.
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 SELECT ACCOUNT-FILE ASSIGN TO 'COA99DF'
      ORGANIZATION IS INDEXED
      ACCESS MODE IS DYNAMIC
      RECORD KEY IS COA-PRIMARY-KEY
      FILE STATUS IS KSAM-STATUS.
 DATA DIVISION.
 FILE SECTION.
 FD  ACCOUNT-FILE.
      COPY COA-REC IN RECDLIB NOLIST.
 WORKING-STORAGE SECTION.
 01  KSAM-STATUS.
      02  KSAM-STATUS-1                                   PIC X(01).
      02  KSAM-STATUS-2                                   PIC X(01).
 01  SAV-ACCOUNT-NUMBER.
      02   SAV-SLASH                                      PIC X(01).
      02   FILLER                                         PIC X(05).
 PROCEDURE DIVISION.
 MAINLINE.
      OPEN I-O ACCOUNT-FILE.
      PERFORM GET-ACCOUNT-KEY UNTIL SAV-SLASH = '/'.
      CLOSE ACCOUNT-FILE.
      STOP RUN.
 GET-ACCOUNT-KEY.
      DISPLAY 'ENTER ACCOUNT NO. OR "/" TO END'.
      ACCEPT SAV-ACCOUNT-NUMBER.
      IF SAV-SLASH NOT = '/' PERFORM READ-REWRITE-COA.
 READ-REWRITE-COA.
      MOVE SAV-ACCOUNT-NUMBER TO COA-PRIMARY-KEY.
      READ ACCOUNT-FILE RECORD INVALID KEY
           PERFORM READ-ERROR.
           IF KSAM-STATUS = '00'
           PERFORM REWRITE-COA-RECORD.
 READ-ERROR.
      DISPLAY 'KEY NOT MATCHED'.
 REWRITE-COA-RECORD.
      EXCLUSIVE ACCOUNT-FILE.
      MOVE 'WHATEVER' TO COA-NAME.
      REWRITE COA-RECORD INVALID KEY
           DISPLAY 'REWRITE ERROR-ACCOUNT-FILE'
           STOP RUN.
      UN-EXCLUSIVE ACCOUNT-FILE.
-----------------------------------------------------------
```

KSAMIO coding is shown next.

```
-------------------------------------------------------------
$CONTROL QUOTE=',USLINIT
 IDENTIFICATION DIVISION.
 PROGRAM-ID. TCOA99CS.
 REMARKS. RANDOM READ/REWRITE OF CHART-OF-ACCOUNTS KSAM FILE.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  SAV-ACCOUNT-NUMBER.
     02  SAV-SLASH                                          PIC X(01).
     02  FILLER                                             PIC X(05).
 COPY KSAMLINK IN KSAMLIB NOLIST.
 COPY COAFILE IN FILELIB NOLIST.
 PROCEDURE DIVISION.
 MAINLINE.
     PERFORM GET-ACCOUNT-KEY UNTIL SAV-SLASH = '/'.
     COPY KSAM-CF IN KSAMLIB NOLIST REPLACING ==???== BY ==COA==.
     STOP RUN.
     COPY KSAMCALL IN KSAMLIB NOLIST.
 GET-ACCOUNT-KEY.
     DISPLAY 'ENTER ACCOUNT NO. OR "/" TO END'.
     ACCEPT SAV-ACCOUNT-NUMBER.
     IF SAV-SLASH NOT = '/' PERFORM READ-REWRITE-COA.
 READ-REWRITE-COA.
     MOVE SAV-ACCOUNT-NUMBER TO KIO-KEY.
     COPY KSAM-RR IN KSAMLIB NOLIST REPLACING ==???== BY ==COA==.
     IF KIO-ERROR-BYTE-1 NOT = '0'
         DISPLAY 'KEY NOT MATCHED'
         ELSE
         PERFORM REWRITE-COA-RECORD.
 REWRITE-COA-RECORD.
     MOVE 'WHATEVER' TO COA-NAME.
     COPY KSAM-RW IN KSAMLIB NOLIST REPLACING ==???== BY ==COA==.
     IF KIO-ERROR-BYTE-1 NOT = '0'
         DISPLAY 'REWRITE ERROR-ACCOUNT-FILE'
         STOP RUN.
-------------------------------------------------------------
```

```
$CONTROL QUOTE=',USLINIT
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SAMP01CS.
 REMARKS. CHRONOLOGICAL READ OF A KSAMFILE.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   NO-RECS                                                    PIC 9(5) VALUE ZEROS.
      COPY KSAMLINK IN KSAMLIB NOLIST.
*                                  (ESTABLISHES KSAMIO LINK)
      COPY CORFILE IN FILELIB NOLIST.
*                                  (COPY COR file/record DESCRIPTIONS)
 PROCEDURE DIVISION.
 MAINLINE.
      COPY KSAM-CI IN KSAMLIB NOLIST.
*                                  (INITIALIZE INTRINSICS)
      COPY KSAMCTLY IN KSAMLIB NOLIST.
*                                  (OPTIONAL - SETS CONTROL Y BREAK)
      COPY KSAM-TON IN KSAMLIB NOLIST.
*                                  (OPTIONAL - SETS KSAMIO FUNCTION DISPLAY
      PERFORM READ-COR UNTIL KIO-ERROR-BYTE-1 = '1'.
      DISPLAY 'NO RECORDS = ' NO-RECS.
      COPY KSAM-CF IN KSAMLIB NOLIST REPLACING ==???== BY ==COR==.
*                                  (KSAM FILE CLOSE)
      COPY KSAM-TOF IN KSAMLIB NOLIST.
*                                  (OPTIONAL - TURNS OFF KSAMIO DISPLAY)
      STOP RUN.
      COPY KSAMCALL IN KSAMLIB NOLIST.
*                                  (CALL TO KSAMIO LINK)
 READ-COR.
      COPY KSAM-CR REPLACING ==???== BY ==COR==.
*                                  (CHRONOLOGICAL READ OF KSAM FILE)
      IF KIO-ERROR-BYTE-1 NOT = '1'
*                                  (TEST FOR END OF FILE)
          ADD 1 TO NO-RECS
          DISPLAY COR-NAME ' ' COR-ADDRESS ' ' COR-SOC-SEC-NO.
```

```
 $CONTROL QUOTE=',USLINIT
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SAMP02CS.
 REMARKS. SEQUENTIAL READ OF A KSAM FILE.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   NO-RECS PIC 9(5) VALUE ZEROS.
      COPY KSAMLINK IN KSAMLIB NOLIST.
      COPY ZIPFILE IN FILELIB NOLIST.
 PROCEDURE DIVISION.
 MAINLINE.
      COPY KSAM-CI IN KSAMLIB NOLIST.
      IF LOGON-ACCOUNT-NAME NOT = 'BUSINESS '
          DISPLAY 'LOGON ACCOUNT CAN NOT USE THIS PROGRAM'
          STOP RUN.
      PERFORM READ-ZIP UNTIL KIO-ERROR-BYTE-1 = '1'.
      DISPLAY 'NO OF RECORDS = ' NO-RECS.
      COPY KSAM-CF IN KSAMLIB NOLIST REPLACING ==???== BY ==ZIP==.
      STOP RUN.
      COPY KSAMCALL IN KSAMLIB NOLIST.
 READ-ZIP.
      COPY KSAM-SR IN KSAMLIB NOLIST REPLACING ==???== BY ==ZIP==.
      IF KIO-ERROR-BYTE-1 NOT = '1'
          ADD 1 TO NO-RECS
          DISPLAY ZIP-NAME.
```

```
$CONTROL QUOTE=',USLINIT
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMP03CS.
REMARKS. LOCK RANDOM UPDATE OF COR FILE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  SAV-SOC-SEC-NO.
    02  SAV-SLASH                                          PIC X.
    02  FILLER                                             PIC X(8).
    COPY KSAMLINK IN KSAMLIB NOLIST.
    COPY CORFILE IN FILELIB NOLIST.
PROCEDURE DIVISION.
MAINLINE.
    PERFORM GET-SSN UNTIL SAV-SLASH = '/'.
    COPY KSAM-CF IN KSAMLIB NOLIST REPLACING ==???== BY ==COR==.
    STOP RUN.
    COPY KSAMCALL IN KSAMLIB NOLIST.
GET-SSN.
    DISPLAY 'ENTER SOC.SEC.NO OR "/" TO END'.
    ACCEPT SAV-SOC-SEC-NO.
    MOVE '25' TO KIO-ERROR-BYTES.
    IF SAV-SLASH NOT = '/'
        PERFORM LOCK-RECORD-READ
            UNTIL KIO-ERROR-BYTES NOT = '25'
        PERFORM REWRITE-COR-RECORD.
LOCK-RECORD-READ.
    MOVE SAV-SOC-SEC-NO TO KIO-KEY.
    COPY KSAM-LRU IN KSAMLIB NOLIST REPLACING ==???== BY ==COR==.
REWRITE-COR-RECORD.
    COPY KSAM-RW IN KSAMLIB REPLACING ==???== BY ==COR==.
    IF KIO-ERROR-BYTE-1 NOT = '0'
        DISPLAY 'REWRITE ? CORFILE ' COR-RECORD
        STOP RUN.
```

```
$CONTROL QUOTE=',USLINIT
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMP04CS.
AUTHOR. D.CONGER.
DATE-WRITTEN. APRIL 27, 1989.
REMARKS. WRITE ASC RECORD TO MPL KSAM FILE AND THEN DELETE.  ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  EOF-ASC                                                PIC X(1)  VALUE SPACES.
01  SAV-SOC-SEC-NO.
    02  FIRST-BYTE                                         PIC X(1) VALUE SPACES.
    02  FILLER                                             PIC X(8) VALUE SPACES.
    COPY TERMINAL IN WORKLIB NOLIST.
    COPY KSAMLINK IN KSAMLIB NOLIST.
    COPY CON-TABL IN WORKLIB NOLIST.
    COPY ASCFILE IN FILELIB NOLIST.
    COPY CORFILE IN FILELIB NOLIST.
    COPY MPLFILE IN FILELIB NOLIST.
PROCEDURE DIVISION.
SET-SCREEN.
    COPY KSAM-CI IN KSAMLIB NOLIST.
    DISPLAY CURSOR-HOME-UP.
    DISPLAY CLEAR-DISPLAY.
    PERFORM GET-YEAR-SEMESTER.
    PERFORM GET-SOC-SEC-NO UNTIL FIRST-BYTE = '/'.
    COPY KSAM-CF IN KSAMLIB NOLIST REPLACING ==???== BY ==COR==.
    COPY KSAM-CF IN KSAMLIB NOLIST REPLACING ==???== BY ==ASC==.
    STOP RUN.
    COPY KSAMCALL IN KSAMLIB NOLIST.
```

```
GET-YEAR-SEMESTER.
    DISPLAY HALF-BRIGHT-INVERSE
        '        Enter Enrollment Year and Semester (YYS)'
        TURN-OFF CURSOR-UP.
    ACCEPT SEL-ENROLLMENT-PERIOD.
    IF SEL-ENROLLMENT-PERIOD NOT NUMERIC OR SEL-ENROLL-TERM < 1 OR > 4
        GO TO GET-YEAR-SEMESTER.
    MOVE SEL-ENROLLMENT-PERIOD TO FILE-NAME-YEAR-TERM IN ASCFILE.
GET-SOC-SEC-NO.
    DISPLAY CURSOR-HOME-UP.
    DISPLAY CLEAR-DISPLAY.
    MOVE ZEROS TO ASC-PRIMARY-KEY.
    DISPLAY HALF-BRIGHT-INVERSE
        '        Enter Soc-Sec-No or "/" to exit'
        TURN-OFF CURSOR-UP.
    ACCEPT ASC-SOC-SEC-NO.
    MOVE ASC-SOC-SEC-NO TO SAV-SOC-SEC-NO.
    IF FIRST-BYTE NOT = '/' PERFORM RANDOM-READ-COR.
RANDOM-READ-COR.
    MOVE ASC-SOC-SEC-NO TO KIO-KEY.
    COPY KSAM-RR IN KSAMLIB NOLIST REPLACING ==???== BY ==COR==.
    IF KIO-ERROR-BYTE-1 NOT = '0'
        DISPLAY '?? TO COR FILE ' ASC-SOC-SEC-NO
        ELSE
        DISPLAY SPACES
        DISPLAY COR-NAME
        DISPLAY SPACES
        PERFORM READ-ASC UNTIL EOF-ASC = 'Y'.
READ-ASC.
    MOVE ASC-PRIMARY-KEY TO KIO-KEY.
    COPY KSAM-PR IN KSAMLIB NOLIST REPLACING ==???== BY ==ASC==,
                                              ==??== BY ==GREATER==.
    IF KIO-ERROR-BYTE-1 = '1'
        MOVE 'Y' TO EOF-ASC
        ELSE
        IF ASC-SOC-SEC-NO NOT = SAV-SOC-SEC-NO
            MOVE 'Y' TO EOF-ASC
            ELSE
            PERFORM WRITE-DELETE-ASC-RECORD.
WRITE-DELETE-ASC-RECORD.
    MOVE ASC-RECORD TO MPL-RECORD.
    COPY KSAM-W IN KSAMLIB NOLIST REPLACING ==???== BY ==MPL==.
    IF KIO-ERROR-BYTE-1 NOT = '0'
        DISPLAY 'MPL FILE WRITE ERROR?'
        STOP RUN.
    COPY KSAM-DLT IN KSAMLIB NOLIST REPLACING ==???== BY ==ASC==.
    IF KIO-ERROR-BYTE-1 NOT = '0'
        DISPLAY 'DELETE ERROR ASC99DF KEY = ' KIO-KEY
        STOP RUN.
    DISPLAY HALF-BRIGHT-INVERSE
        ASC-COURSE-ID-NUMBER '=COURSE ID NUMBER DELETED'
        TURN-OFF.
```

LOOPING THROUGH IMAGE DATASETS WITH PROTOS

or

But Mother Always Said "It's Not Polite to Point"

Brian Becker
NOMC, Inc.
P.O.Box 3911
New Orleans, LA 70177-3911
(504) 949-9801

INTRODUCTION

PROTOS' FOR ALL set... and FOR detailset... statements provide powerful programming tools for processing whole datasets, subsets of datasets, detail chains, and qualifying entries along detail chains. These structures can offer the programmer some of the greatest time savings available through the PROTOS language - In short, easy to read statements, PROTOS initializes the dataset or chain; validates entries; tests for end-of-file / end-of-chain; and will even generate the code to sort the entries on up to 10 sort fields. There are some situations, however, in which these loops can skip entries or cause the wrong record to be updated or deleted. This is not due to any 'bugs' in PROTOS, but can occur when the user lacks a complete understanding of the way PROTOS handles IMAGE record pointers.

One of the major features of PROTOS is that it generates structured COBOL code that is accessible to the user, rather than using the 'black-box' approach of some other 4GL's. Even so, some details of these statements are not fully documented; and the functional code, particularly in the 'SORTED' clauses, is written into SL routines, thus cannot be easily ascertained even by analyzing the COBOL code.

There are many PROTOS statements that access IMAGE datasets, but this discourse will be limited to the loops generated by the FOR...set structures. I should note that the FOR ALL... statement also may be used to read KSAM and MPE files, but that is also beyond the scope of this text. For those of you who like to pick nits as much as I do, when I say 'PROTOS does this,' or 'PROTOS generates that,' I am generally referring to the COBOL code which is generated by PROTOS from the user's source code. When I refer to a PROTOS SL routine or the actual PROTOS program (called from the PROBUILD or PROWRITE UDCs) it will be explicitly stated.

There are four basic forms of 'FOR ALL set' and four forms of 'FOR detailset', as follows:

The statements

```
FOR ALL  dataset ...
FOR ALL  dataset   WITH  condition ...
FOR ALL  dataset   SORTED ...              and
FOR ALL  dataset   SORTED ... WITH condition
```

serially read a Master or Detail dataset. (There is one exception, which will be discussed later.) If the 'WITH condition' clause is used, the statements between the 'FOR ALL dataset ...' and its associated 'ENDFOR' are executed against those entries which meet 'condition', otherwise all entries in the set are processed.

The other four forms are similar in purpose, but operate on IMAGE detail chains only:

```
FOR detailset WITH search-item=keyval
FOR detailset WITH search-item=keyval AND condition
FOR detailset SORTED... WITH search-item=keyval
FOR detailset SORTED... WITH search-item=keyval AND condition
```

perform the program statements on the entries along a detail chain. The path is determined by 'search-item', and the chain is established by 'keyval'. If 'search-item' is not specified, the primary path for the set is used by default. If 'keyval' is omitted, PROTOS uses the current value for 'search- item' that is present in the set's buffer.

All eight forms allow database modification - e.g. PUT, UPDATE and DELETE. In all forms, 'condition' can be a simple or compound logical expression. Before delving too deeply into this discussion, the user should also be familiar with PROTOS' IMAGE-AREA buffer and the IMAGE-STATUS items in the ERROR-AREA buffer. Examples are shown below.

FIGURE 1 ; PROTOS IMAGE & ERROR AREAS

```
01  IMAGE-AREA
     05 IMAGE-BASE-NAMES.
        10 dbase-BASE-NAME           PIC X(28).
     05 IMAGE-PARMS.
        10  IMAGE-BASE               PIC X(28).
        10  IMAGE-PASSWORD           PIC X(8)     VALUE ";".
        10  IMAGE-DATASET            PIC X(16).
        10  IMAGE-MODE               PIC S9999    COMP.
        10  IMAGE-NOLOCK             PIC XX       VALUE "DR".
        10  IMAGE-ITEM               PIC X(16).
        10  IMAGE-ARG                PIC X(80).
        10  FILLER       REDEFINES   IMAGE-ARG.
            15  IMAGE-RECORD-NUMBER          PIC S9(9) COMP.
            15  IMAGE-RECORD-NUM     REDEFINES      IMAGE-RECORD-NUMBER
                                     PIC S9(9)    COMP.
            15  FILLER               PIC X(76).


01  ERROR-AREA.
    05  FILLER                       PIC S9999    COMP.
    05  IMAGE-STATUS.
        10  IMAGE-COND               PIC S9999    COMP.
            88  IMAGE-OK                     VALUE ZERO.
            88  IMAGE-BEG-OF-FILE            VALUE 10.
            88  IMAGE-END-OF-FILE            VALUE 11.
            88  IMAGE-DIR-BEG-OF-FILE        VALUE 12.
            88  IMAGE-DIR-END-OF-FILE        VALUE 13.
            88  IMAGE-BEG-OF-CHAIN           VALUE 14.
            88  IMAGE-END-OF-CHAIN           VALUE 15.
            88  IMAGE-NO-ENTRY               VALUE 17.
            88  IMAGE-CRITICAL-UPDATE        VALUE 41.
            88  IMAGE-DUP-VALUE              VALUE 43.
        10  IMAGE-STATUS-2           PIC S9999    COMP.
        10  IMAGE-STATUS-3-4         PIC S9(9)    COMP.
        10  IMAGE-STATUS-5-6         PIC S9(9)    COMP.
        10  IMAGE-CHAIN-LENGTH       REDEFINES IMAGE-STATUS-5-6
                                     PIC S9(9)    COMP.
        10  IMAGE-CHAIN-LEN          REDEFINES IMAGE-STATUS-5-6
                                     PIC S9(9)    COMP.
        10  IMAGE-STATUS-7-8         PIC S9(9)    COMP.
        10  IMAGE-STATUS-9-10        PIC S9(9)    COMP.
        10  IMAGE-FWRD-PTR           REDEFINES IMAGE-STATUS-9-10
                                     PIC S9(9)    COMP.
```

The 'FOR ... set ' structures are discussed below in order of their complexity.

'FOR ALL dataset... [WITH condition]'

The 'FOR ALL dataset...' and 'FOR ALL dataset... WITH condition...' formats are fairly straightforward and are accurately described in the PROTOS manual. The dataset is first rewound with a Mode 3 DBCLOSE (the record pointer is reset to the first record while dataset remains open.) PROTOS then generates a COBOL 'PERFORM ... UNTIL' loop that executes a serial (Mode 2) DBGET on the set. If a 'WITH condition' clause is present, each entry is checked for 'condition' as it is read. The loop executes until end-of-file, or until an 'EXIT' statement is encountered.

There is one exception to the serial read used by 'FOR ALL dataset'. If the dataset is a detail set, 'WITH condition' is used, and 'condition' is a simple search-item=value structure with search-item being a valid IMAGE path to the set, then PROTOS will treat the statement as if it were a 'FOR detailset' structure, and will read the *chain* as described below rather than serially reading the dataset.

'FOR detailset ... [WITH search-item[= keyval] [AND condition]]'

The basic function of the 'FOR detailset [WITH search-item=keyval]' and the 'FOR detailset [WITH search-item=keyval [AND condition]]' statements are also described in the PROTOS manual. The chain head is first established via a call to DBFIND, using the 'search-item' and 'keyval' specified in the 'WITH' clause (or using the defaults if the 'WITH' clause is omitted.) The IMAGE intrinsics actually used to read the chain, however, can only be discerned by reviewing the COBOL code.

When a 'FOR detailset ...' statement is encountered during a PROWRITE, PROTOS generates an independent item in the COBOL program's working storage section called 'setname-ENTRY', where 'setname' is the name of the detail set. For example, if you have a detail set called INVEN, and write the loop

```
        FOR  INVEN  ...
                ...
                program statements
                ...
        ENDFOR
```

PROWRITE will generate the variable

```
   01  INVEN-ENTRY          PIC S9(9).
```

After the initial DBFIND is executed, PROTOS moves the Forward Record Pointer (Image Status word 9-10) into this variable, and enters a PERFORM...UNTIL loop to read the entries and execute your statements. Rather than executing a Mode 5 DBGET to read the entries, however, PROTOS performs a paragraph called 'setname-GET-4'. ('setname' is again the name of the detail dataset.) This paragraph moves 'setname-ENTRY' - the forward record pointer - into the Argument parameter IMAGE-ARG (refer to the IMAGE-AREA buffer in Figure 1,) and does a Directed (Mode 4) DBGET. The new Forward Record Pointer returned from the DBGET is immediately moved to 'setname-ENTRY' in anticipation of the next read. Figure 2 shows extracts from a PROTOS program, and the resulting COBOL code, using 'FOR detailset'.

FIGURE 2 ; EXAMPLE OF PROTOS' 'FOR detailset'

This sample section of PROTOS code

```
PROC  YARDRPT  USING   IMAGE-AREA
                       ERROR-AREA
                       PASS-LINE-CODE :  X(8)

...
< < Report definition statements > >
...

WITH YARD
BEGIN
   FOR  YARD   WITH    SHIPLINE = PASS-LINE-CODE
               AND     IMPORT-EXPORT = "I"
       REPORT 1
   ENDFOR
END
...
< < Report Heading, Detail, Footing paragraphs > >
...
```

Will generate the following COBOL code:

```
  $CONTROL DYNAMIC
   IDENTIFICATION DIVISION.
   ...
   ENVIRONMENT DIVISION.
   ...
   DATA DIVISION.
   ...
   WORKING-STORAGE SECTION.
   ...
   01   YARD-ENTRY                 PIC S9(9) COMP.
   ...
   01 RECORD-AREA.
       05   YARD-RECORD            COPY YARREC01.
            15   < <yard detailset definition from Copylib> >
   ...
   LINKAGE SECTION.
   01  IMAGE-AREA                  COPY IMAGELINK.
   ...
   01  ERROR-AREA                  COPY ERRLINK.
   ...
   01  PASS-LINE-CODE              PIC X(8).
   ...
   PROCEDURE DIVISION USING IMAGE-AREA  ERROR-AREA  PASS-LINE-CODE.
   MAIN-LINE-SECTION SECTION.
   MAIN-LINE.

       OPEN OUTPUT PRINTER-1.
       MOVE PASS-LINE-CODE         TO      IMAGE-ARG-SHIPLINE.
       MOVE  "SHIPLINE"            TO      IMAGE-ITEM.
       PERFORM                             YARD-FIND.
       MOVE    IMAGE-STATUS-9-10   TO      YARD-ENTRY.
       MOVE CONTINUE-VALUE         TO      FOR-RESULT.
       PERFORM                             01-FOR        UNTIL    EXIT-FOR.
       MOVE CONTINUE-VALUE         TO      FOR-RESULT.
       CLOSE PRINTER-1.
   GOBACK.
```

(Continued on next page)

FIGURE 2 - Example of PROTOS 'FOR detailset' (Cont'd)

```
01-FOR.

    PERFORM   YARD-GET-4.

    PERFORM   YARD-GET-4
        UNTIL NOT IMAGE-OK
        OR    (YAR-IMPORT-EXPORT = "I" ).
    IF  IMAGE-OK
        PERFORM REPORT-1-ROUTINE
    ELSE
        MOVE  EXIT-VALUE          TO  FOR-RESULT.

    ...

YARD-FIND.

    MOVE  YARDDB-BASE-NAME        TO  IMAGE-BASE.
    MOVE  "YARD"                  TO  IMAGE-DATASET.
    CALL  "MYDBFIND"              USING  IMAGE-PARMS  ERROR-AREA.

    ...

YARD-GET-4.

    MOVE  YARDDB-BASE-NAME        TO  IMAGE-BASE.
    MOVE  4                       TO  IMAGE-MODE.
    MOVE  "YARD"                  TO  IMAGE-DATASET.
    MOVE  YARD-ENTRY              TO  IMAGE-RECORD-NUMBER.
    CALL  "MYDBGET"               USING  IMAGE-PARMS  ERROR-AREA.
    IF  IMAGE-OK
        MOVE  IMAGE-BUFFER        TO  YARD-RECORD
        MOVE  IMAGE-STATUS-9-10   TO  YARD-ENTRY.

    ...
```

Remember that the purpose of using this loop structure is to process an entire chain in the most expedient manner. This method is quite a bit more efficient than IMAGE's Mode 5 DBGET, especially if your routine is executing, e.g., a series of DELETEs and PUTs down the chain.

Caution must be used, however, if other concurrent processes also happen to be modifying the same chain. For example, say that you are reading down a chain whose key value is 'PRO', and have just read record 12. The forward record pointer, which was moved to 'setname-ENTRY', points to record 18. Since IMAGE utilizes a doubly-linked list structure, record 18's backward record pointer will point to record 12 - your current record. Now suppose that while you are doing your thing to record 12, another process DELETE's record 18 and PUT's a new record, with a key value of 'COB', in its place. You finish with record 12 and proceed to read the next record on your chain. PROTOS moves 'setname-ENTRY', which still points to record 18, into IMAGE-ARG and does its directed DBGET. Now, however, record 18 is on the chain COB, not PRO. With a 'true' chained (Mode 5) read, IMAGE would check record 18's backward pointer, discover that it did not point to record 12, and report a problem (the condition word would reflect DBERROR 18 - Broken Chain.) When you do Directed DBGETs, however, IMAGE does not do any such checking. As a result, in this example, your program will proceed down the wrong chain (COB) with no indication that anything is amiss.

There are some other rare but very possible circumstances in which even a Mode 5 chained DBGET can get thrown off the track. Such situations are inherent to IMAGE, and can occur regardless of the programming language you use. These have been described in detail by other authors more knowledgable than I, so I will only offer some suggested solutions.

The most secure solution is to lock the chain during the execution of your 'FOR detailset...' loop. Even if you are not doing any database modifications, someone else's PUT's and DELETE's could misdirect your reads.

If locking is not practicable, the next best method is to check both the record pointers and the original search-item value. If either is suspect, the application should warn the user, attempt recovery, or simply abort, depending upon how much code you want to write and how critical the application is. To accomplish this, store the current-record number (Status word 3-4) as you DBGET each entry (i.e as one of your first program statements after 'FOR detailset'.) On each subsequent DBGET, check the new entry's backward pointer (Status word 7-8) against the last current-record number stored, AND check the search-item value of the retrieved entry against the value used in the initial DBFIND (the 'keyval', explicit or default, used in the 'WITH search-item=keyval' clause.) This solution requires a bit of additional logic, but in multi-processing environments it is almost mandatory when you cannot lock the chain. Figure 3 shows an example of this checking.

Crucial Note: This method will only work if no 'AND condition...' clause is used. When a condition is in effect, PROTOS will read all records on the chain and advance the forward record pointer for each entry read. Your manually-adjusted current-record pointer, however, will only be advanced when an entry qualifies.

FIGURE 3. - CHECKING PREVIOUS RECORD AND KEY VALUES IN PROTOS.

```
VAR     CHK-PREV-RECORD        9(9) COMP    VALUE ZERO

    ...

WITH  YARD
BEGIN  YRD-RPT-01-MAIN-LOOP

    MOVE  ZERO      TO       CHK-PREV-RECORD

    FOR  YARD       WITH     SHIPLINE = PASS-LINE-CODE

        IF IMAGE-STATUS-7-8 < > CHK-PREV-RECORD
         OR SHIPLINE         < > PASS-LINE-CODE

            PERFORM              999-SL-CHAIN-DISRUPTED
            MOVE    TTRUE TO RESTART-FLG
            EXIT

        ENDIF
        MOVE IMAGE-STATUS-3-4 TO CHK-PREV-RECORD

        additional processing statements

    ENDFOR
END
```

SORTING

Much of any computer's time is spent producing reports, not just entering data. Often, this is simply a matter of reading some records and writing the desired information to the printer. The people reading these reports, however, usually like to see things sorted in some meaningful order. (Note this is *their* version of 'meaningful' - I didn't say anything about *logical*.)

To this end, the 'SORTED' clause introduces a new level of power. One short, readable sentence can relieve the programmer of the task of writing pages of SORT procedure code. When the 'SORTED' clause is specified with any of the 'FOR ... set' constructs, PROTOS takes care of creating a sortfile, loading the desired entries, sorting, and returning the records to your program. Up to ten sort fields may be specified, none of which has to be an IMAGE search or sort item. If the record length is fairly short and/or the volume of records to be sorted is low, the optional 'USING MEMORY' phrase can allow the sort to be performed entirely in memory, reducing execution time dramatically. (I mention the 'USING MEMORY' phrase for the sake of completeness. It can have a significant impact on your program's performance, but otherwise has no effect on the way the SORTED clause functions.)

Unfortunately, the 'SORTED' clause is poorly documented, and the actual sort operations occur within proprietary SL routines so cannot be fully evaluated from the COBOL code. This function also presents a great potential for erroneous reports or logical data corruption due to the way it handles record pointers. Finally, probably in order to follow one of Murphy's laws that nothing be as simple as it seems, the 'SORTED' clause functions differently depending on whether a 'condition' is used to select the desired entries.

SORTS ON ENTIRE DATASETS OR CHAINS

Without a 'condition' clause, the statements

```
  FOR  ALL  dataset   SORTED ...     and
  FOR  detailset   SORTED ... [WITH search-item = keyval]
```

both function in roughly the same manner:

At run time, a temporary sort file (or memory buffer if the 'USING MEMORY' clause is used) is created. On each iteration of the 'FOR ... set ...' loop PROTOS calls its own SL procedure, 'READIMAGESORT'. The *first* time this procedure is called, it reads *every entry* in the set, or *all records* on the detail chain. When 'FOR ALL dataset' is used the dataset is serially read; with 'FOR detailset ...' the chain is read using the directed read method described earlier. For each entry, the IMAGE record pointer and the requested sort field values are released directly to the SORT.

After all the entries are read and sorted, the pointers are returned into the the sort-file or memory buffer. The first (sorted) record pointer is used to do a directed DBGET, returning the database entry to the application. On each subsequent iteration of the loop, the next pointer is returned, and another directed DBGET is performed to re-retrieve its record from the database. The loop is repeated until the sort-file/buffer is exhausted or until a user-declared EXIT is encountered.

SORTING SELECTED RECORDS FROM A SET OR CHAIN

When a 'Condition' is added to the statement, for example

```
        FOR ALL   dataset  SORTED...      WITH   condition
or
        FOR     detailset     SORTED...      WITH   search-item = keyval  AND  condition
```

two PERFORM ... UNTIL loops are used to sort and process the desired entries. The first loop reads the dataset or chain, checking each entry for 'condition' as it is read. If the entry qualifies, the PROTOS SL procedure 'MYDBSORTRELEASE' loads its record pointer and a *full copy* of the entry - not just the sort fields - into a temporary sort file or buffer. The records are not released for sorting until the first loop has completed and the sort-file/buffer is loaded.

When the first loop is finished the second COBOL loop is executed. Another PROTOS SL procedure, 'MYDBSORTRETURN', sorts the records and copies the sorted entries back to the temporary sort-file/buffer. Each iteration of the loop then returns a sorted record pointer which is used to re-retrieve its entry from the database with another directed DBGET.

To avoid the extra overhead of writing and reading this temporary file, the PROTOS manual suggests that if a large percentage of entries in the set or chain will qualify, it is more efficient to write an Unconditional 'FOR ... set' loop and then manually test each record as it is returned to the application for processing.

Another major point not clearly stated in the manual is that the conditional sort uses the entire record rather than just the pointer and sort fields. Processing the extra length may or may not offset any gain which might be realized by sorting fewer records. More importantly, using the entire record may prohibit the use of the USING MEMORY phrase. If you have a dataset with a data length of 250 words (500 bytes), you would not even be able to sort 150 entries in memory: No data segment can exceed 32K words, but 250 w X 150 recs = 37,500 words. If, however, you have a total of maybe 10 or 20 words in sort fields plus the 2-word record pointer, it might be possible to sort several hundred records, possibly even one or two thousand, entirely in memory depending on the size of the rest of your data area, your MAXDATA specifications, etc.

Performance issues aside, the sort function looks good, doesn't it? PROTOS took care of the selection and the sorting, and the loop is now processing your records just as you wanted, right?

Watch it - Here comes Murphy!

In a multi-user environment, almost anything could happen between the time a record is loaded into the sortfile and the time the sorted entry is returned for processing. If another process deletes an entry after it has been loaded into the sortfile, it is not always sufficient to just trap for DBERROR 17 - Missing or Empty Record. It may also be destructive if other users are adding new records. Entries added to the database after the sortfile is loaded may be missed completely, or in some cases may be over-written or deleted.

To offer a real-life example, we have a group of batch programs which access our principle inventory database. Each routine can take from 2 to 25 minutes wall time to process depending on user load and number of records processed. These routines may be run at any time, so they must be able to run concurrently with on-line users. The boss's boss, who is surprisingly literate as far as top management goes, has decreed that the programs cannot lock sets or even chains, since to do so could suspend 10 to 25 percent of our on-line operations while these programs are running.

So What?

Consider the following: What if a record pointer loaded into the sortfile was to a synonym in a Master set, and during the sort procedure another process deleted its primary? Regardless of whether you program in PROTOS/COBOL, SPL, or any other language, a directed read using a pointer that is 5 minutes or 5 seconds old could easily return a DBERROR 17 when in fact the entry is not 'missing' but has simply migrated. Migrating secondaries can cause a potentially more serious problem. Suppose another user has PUT a new entry which hashed to the record occupied by your synonym. IMAGE moves your synonym, but the pointers in your sortfile are not updated to reflect this. The directed DBGET will now return the new primary entry rather than the synonym that was originally retrieved and sorted. You can also retrieve, update, and/or delete the wrong entry in a detail set if another process deletes one of 'your' records and someone puts a new one in its place. (Don't scream and holler and yell at PROTOS - while I was writing this paper, our shop encountered this situation with some QUERY reports *- 3 times in one week!*)

In any of these cases, 'simple' reports can be confusing at best. If your procedure modifies the database, the logical integrity of the data could be destroyed. Note also that in the case of Master datasets there does not even have to be another process involved, and locking may be purely academic. If you do PUT's or DELETE's in your own loop, synonyms can migrate even with the dataset locked or the database opened for exclusive access.

With the 'FOR ... SORTED ...' statements, there is no direct way to test a retrieved record against a known value. Because of this, our shop uses this structure ONLY for the following types of programs, and then only within the noted restrictions:

a) Report routines (no database modifications of any kind,) when the set or chain can be locked, or when the report is not critical and can be easily re-run if anything goes awry;
b) Batch Updates, Deletes or Puts to Detail datasets, but only if the set or chain can be locked or opened exclusively;
c) When the set in question is a Master dataset, the routine performs DBUPDATEs ONLY - No DBPUT's or DBDELETE's; and again only with exclusive access.

Our preferred solution requires additional effort, but we feel that the integrity of our databases is more than enough justification. Whenever batch modifications to the database are involved, we either use an UN-SORTED 'FOR ... set' loop, or construct a loop with the FIND, GET, GET NEXT and/or GET PREVIOUS statements. When a sorted listing of the transactions is required, the pertinent data is written to an MPE file as each entry is processed. After all database modifications are completed, PROTOS' 'SORT' verb is used to sort the transaction file and other appropriate statements are used to produce the report.

SUMMARY

Upon encountering some of the problems which prompted me to put this little opus to paper, I often became frustrated with the 4GL concept in general and PROTOS in particular. On further investigation, however, most of what is really going on came to make sense. And, as in any language, there are often alternative statements or approaches that may be better suited to certain problems.

If there is a flaw in PROTOS' 'FOR... set' statements, it is in the documentation. In my opinion, the manual (including the much-improved October 1989 revision) does not adequately define the true operations involved nor does it provide sufficient warnings of potential problem areas.

On the other hand, when the programmer is armed with an understanding of their intricacies, these structures are quite safe and offer a great deal of flexibility and power.

Advanced CI Programming
The 2.1 Story

Scott Cressler
Jeff Vance
Steve Elmer

Hewlett-Packard
19447 Pruneridge Avenue
Cupertino, CA 95014

OVERVIEW

The goal of this paper is to present changes to the MPE XL Command Interpreter (CI) which will be available in MPE XL, Release 2.1. These changes greatly enhance the usefulness of the CI as a tool for automating tasks.

Many of the examples in this paper are taken from a command file called SPOOK. This command file is presented at the end of the paper for reference.

GENERAL CI PROGRAMMING

Basic programming of the MPE XL Command Interpreter (CI) is covered in the MPE XL Commands Reference Manual and has been addressed in previous papers and talks at BARUG, at Interex, and in Interact magazine. However, to facilitate the understanding of the examples presented in this paper, we will briefly cover some of the techniques used in programming the CI. If you feel comfortable with terms such as "string substitution" and "entry points", please skip ahead to the heading "The Style Of This Paper."

Expressions

One of the first concepts which must be understood about CI programming is that of expression evaluation. We have all understood expression evaluation from our early years at school: 1+1=2. In the CI, an expression can be evaluated to either an integer, string or Boolean value. CI variables can be used in expressions just as variables can be used in expressions in programming languages and algebra. If A=1, then A+1=2. There are also many functions which can be used in CI expressions to obtain and process information. This is similar to the functions in mathematics: ABS(-2)=2. Expressions are evaluated by several of the commands in the CI, namely CALC, ELSEIF, IF, SETVAR, and WHILE. Expressions can also be evaluated and their values inserted into any command using string substitution, discussed below. An example of an expression is:

```
 elseif len(setvar(_spook_rec,ltrim(rtrim(input("> "))))+chr(0) > 1 then
```

We will examine this example in more detail later in the paper.

String Substitution

Parameter Substitution

Another commonly used CI programming technique is called string substitution. This is a feature of the CI which is available on every command line processed. Before any processing is done with a command line, the CI performs string substitution. It is important to remember that string substitution is performed before the command name is determined and before the expression part of a command is evaluated. The most common use of this feature is insertion of the value of a User Command (UDC or Command File) parameter, variable or expression into a command line. Some examples illustrate this concept:

```
  if "!entry" = "get_dfid_or_username" then
```

In this example, the value of the parameter `entry` is substituted in the command line in place of the string `!entry`. For instance, if the value of the parameter `entry` is `main`, the command actually executed by the CI is:

```
  if "main" = "get_dfid_or_username" then
```

Of course, this command evaluates to FALSE.

Variable Substitution

Another type of string substitution which is commonly used is the substitution of the value of a CI variable into a command line. It is **not** a good CI programming practice to use string substitution when using a variable in an expression. Variables in CI expressions have types, and substituting their values into a line defeats the ability of the CI to use these types in evaluating the expression. In addition, substituting the value of a variable into an expression prior to evaluating the expression is extra overhead. Once string substitution has completed, and the insertion of the value has been done, the expression still needs to be evaluated.

The substitution of variable values into command lines is more commonly used to dynamically modify the execution of a command. For example:

```
spoolf !_spook_dfid !_spook_opts ;DELETE
```

If the variable `_spook_dfid` had previously been given the string value "@" and `_spook_opts` had the value ";SELEQ=[OWNER=ME.MYACCT]", the post string substitution command (the command actually executed by the CI) would be:

```
spoolf @ ;SELEQ=[OWNER=ME.MYACCT] ;DELETE
```

This command would purge all output spool files owned by the user ME.MYACCT.

Expression Substitution

The third common type of string substitution is expression substitution. This feature of the CI allows the insertion of the value of an expression into any command line, even those which do not support expressions. For example:

```
echo Eof: !_spook_last.  Created on: &
     ![finfo(_spook_text,"fmtcreated")] at &
     ![finfo(_spook_text,"fmtalloctime")].
```

The ECHO command just writes everything on the command line to $STDLIST. Suppose `_spook_last` has the value 25 and `_spook_text` has the value "O7934.OUT.HPSPOOL". If the file O7934.OUT.HPSPOOL was created on March 27, 1990, at 5:47 PM, the command which would be executed would be:

```
echo Eof: 25.  Created on: TUE, MAR 27, 1990 at 5:47 PM.
```

Resulting in the following being displayed:

```
Eof: 25.  Created on: TUE, MAR 27, 1990 at 5:47 PM.
```

If you look at the SPOOK command file example at the end of this paper, you will notice that string substitution is **only** used to insert parameter values into commands (including commands supporting expressions, such as IF), and to insert variable or expression values into commands which don't support expressions.

Entry Points

The IF command in the above example is the use of the CI programming technique called "entry points". An entry point defines a subtask, or "subroutine", within a User Command. Each subtask has its own entry point. An entry point is normally used when a subtask must be executed several times or must have its input and/or output redirected. To execute an entry point from within a command file, the command file is recursively called, passing the appropriate entry point. For example, the following excerpt is from a command file called SPOOK:

```
  PARM entry=main
  PARM cmd=""
  ANYPARM cmdparms=null
  if "!entry" = "get_dfid_or_username" then
     ...
  elseif "!entry" = "main" then
     ...
     xeq spook entry = "get_dfid_or_username" cmdparms=&
         ![lft(_spook_parms,pos(';',_spook_parms)-1)]
     ...
```

When this command file is first executed, the default value of `entry`, "main", is used. All of the command file is skipped until the entry point for "main" is reached. As part of the `main` entry point, the command file (SPOOK) is executed recursively (using the XEQ command to avoid any UDCs which might be defined with that name). This invocation of the command file uses a different value for `entry`, and so the entry point corresponding to that value of `entry` is executed; the rest of the command file is skipped. Entry points were used in this example because the subtask of parsing input for a Device File Identifier (DFID) or user name, done by the `get_dfid_or_username` entry point, is performed several times within this command file. The commands to perform this parsing could be placed everywhere in the command file they are needed (a maintenance nightmare), or they could be placed in a second command file which would be called from this command file (causing the original command file to depend on other files).

THE STYLE OF THIS PAPER

This paper will consist mostly of examples. Each example will present a problem to be solved using CI programming. A solution will then be presented using the "old" MPE XL CI, followed by the solution using the new MPE XL CI.

NEW FOR MPE XL 2.1

The MPE XL Command Interpreter has been changed dramatically for Release 2.1. Some of the changes are general to the CI. Other changes are new or modified functions available in expressions. Still other changes are new commands and changes to User Commands.

New CI Features

Command Input/Output Redirection (CIOR)

Command Input/Output Redirection (CIOR) redirects the input and/or output of a command to a file and is available on most commands. The file can be almost any MPE XL file or file equation. CIOR processing is performed by the CI on a command line after string substitution but before the command is executed.

CIOR is invoked by including a redirection specification on a command line. A redirection specification is a string of the form `>outfile`, `>>appfile` or `<infile`. ">" tells the CI that the output of the current command is to be sent to the file, `outfile`. ">>" says that the output of the command is to be appended to `appfile`. "<" indicates that input for the command is to be taken from `infile`.

Using CIOR to redirect input and output is a great improvement over the way this was done in previous versions of the CI.

For example, suppose you want to purge all the files in a given file set. You could create the following command file, named WILDPRGE:

```
parm fileset,entry=main
if "!entry" = "main" then
   file outfile;nocctl;rec=-80,,f,ascii;temp
   listf !fileset,6;*outfile
   file outfile,oldtemp
   run ci.pub.sys;info='wildprge !fileset,"process files"'&
                 ;stdin=*outfile&
                 ;stdlist=$NULL;parm=3
   reset outfile
   purge outfile,temp
   deletevar _wildprge_@
elseif "!entry" = "process files" then
   setvar _wildprge_num_records finfo('outfile','eof')
   while _wildprge_num_records > 0 do
     input _wildprge_next_line
     continue
     purge !_wildprge_next_line
     setvar _wildprge_num_records _wildprge_num_records - 1
   endwhile
endif
```

On MPE XL 2.1, this command file is reduced to:

```
parm fileset
comment This command file relies on the fact that each record read
comment from the message file is deleted.
file outfile;msg
listfile !fileset,qualify >*outfile
reset outfile
while finfo('outfile','eof') > 0 do
  input _wildprge_next_line <outfile
  continue
  purge !_wildprge_next_line
endwhile
deletevar _wildprge_@
```

The default type of an output file created by CIOR is a temporary ASCII file with 256-byte, variable records. This was chosen because of its usefulness as input to User Commands, since only the data of a record will be read by an INPUT command.

There may be times when a string will appear in a command line which looks like a redirection specification but which must not be treated as one. For example, the following line appears in the SPOOK command file:

```
echo !<dfid!>  ::= [#O]n or #In or *
```

It is not intended that this command take input from a file called `dfid`. Instead, the goal is to display the following:

```
<dfid>  ::= [#O]n or #In or *
```

This is what is displayed and input is not redirected because the "!" before the "<" and before the ">" indicates to the CI that CIOR is not to be done.

CIOR is not available on the following commands because of the ambiguity of including redirection specifications on their command lines: CALC, ELSEIF, IF, SETVAR, and WHILE. For example, consider:

```
:calc a > b
```

Is this an attempt to calculate the comparison of the variable A to the variable B? Or is it an attempt to calculate the value of the Boolean variable A and output the result of that calculation to a file named B? To remove this ambiguity, CIOR is not allowed on these commands.

CIOR is not performed on the TELL, TELLOP and WARN commands since they can easily contain ">" or "<" characters in existing jobs and User Commands. The REMOTE command is excluded from CIOR since the command in the REMOTE may be one of those excluded (e.g. `:remote if a > b then`), and COMMENT commands are excluded from CIOR to speed their processing.

New CI Functions

Another major area of enhancement in the CI was the addition of several new functions which can be used in expressions.

Numeric(), Alpha(), and Alphanum()

In the old CI, the following commands would check that all characters in a string were numeric (digits '0' through '9'). This might be used if the string was to be substituted as a numeric parameter of a command:

```
  setvar i 1
  while i <= len(str_var) and &
        str(str_var, i, 1) >= '0' and &
        str(str_var, i, 1) <= '9' do
    setvar i i+1
  endwhile
  if i > len(str_var) then
     comment str_var is numeric
     build abc;rec=-!str_var
  endif
```

Using the numeric() function of the new CI, this is reduced to:

```
  if numeric(str_var) then
     comment str_var is numeric
     build abc;rec=-!str_var
  endif
```

Numeric() takes a string expression as a parameter and returns TRUE if all characters in the string are digits ('0' through '9'). Numeric() returns FALSE if any non-digit character is encountered in the string. There are two companion functions to this function, alpha() and alphanum(). Alpha() only returns TRUE if all of the characters in its string parameter are alphabetic. Alphanum() returns TRUE if all of the characters in its string parameter are either alphabetic or digits.

Setvar(), Input(), Rtrim(), and Ltrim()

Several of the new CI expression evaluator functions have already been used in this paper. When put together, they can significantly reduce the number of commands required to accomplish a task, increasing the performance of that task.

The setvar() function allows a variable to be created or have its value modified within an expression. The input() function reads from $STDIN and returns the input received. The rtrim() and ltrim() functions trim blanks (or occurrences of any other string) from the right and left ends of a string, respectively.

As an example of the use of these functions together, consider the command line mentioned in the earlier discussion of expressions:

```
elseif len(setvar(_spook_rec,ltrim(rtrim(input("> ")))+chr(0))) > 1 then
```

This command prompts the user with a "> ", reads input from the user, trims blanks from the start and finish of the input, appends a NULL (`chr(0)`) to the input, sets the variable `_spook_rec` to the value and then tests that it's length is greater than one. On the old CI, this would have been:

```
input _spook_rec, "> "
comment Strip blanks from right and left of input
while rht(_spook_rec,1) = " " do
  setvar _spook_rec lft(_spook_rec,len(_spook_rec)-1)
endwhile
if len(_spook_rec) > 0 then
   while lft(_spook_rec,1) = " " do
     setvar _spook_rec rht(_spook_rec,len(_spook_rec)-1)
   endwhile
endif
setvar _spook_rec _spook_rec+chr(0)
if len(_spook_rec) > 1 then
```

The 2.1 version of this example also uses the new ELSEIF command, which is discussed later in the paper.

Rpt()

Another of the new functions, rpt(), is used in the SPOOK command file to print a line which contains a leading record number. Rpt() returns a string repeated a given number of times. SPOOK uses rpt() to print a variable number of spaces the record number so that the output will look like the following:

```
        9) This is record 9, preceded by 7 spaces.
       10) This is record 10, note that it is preceded by 6 spaces.
```

Rpt() is also used to highlight a word in a line which has been displayed by printing some carets ('^') beneath the word. For example, if the word to be highlighted was "preceded" and the line printed was line 10 above, the following would be output:

```
      10) This is record 10, note that it is preceded by 6 spaces.
                                             ^^^^^^^^
```

To perform these tasks in the old CI, the following commands would be required. In this example, assume the variable **`eighty_blanks`** had previously been set to eighty blanks, **`eighty_carets`** had been set to eighty carets, **`_spook_pattern`** contains the word to be highlighted (e.g. "preceded" in the above example), **`_spook_curr_rec`** contains the integer record number, and **`_spook_rec`** contains the record which is to be printed.

```
setvar leading_blanks lft(eighty_blanks,8-len("!_spook_curr_rec"))
setvar caret_blanks lft(eighty_blanks, 9+pos(_spook_pattern,_spook_rec))
setvar carets lft(eighty_carets,len(_spook_pattern))
echo !leading_blanks!_spook_curr_rec)!_spook_rec
echo !caret_blanks!carets
```

The 2.1 CI solution is:

```
echo ![rpt(" ",8-len("!_spook_curr_rec"))]!_spook_curr_rec) &
     ![rtrim(_spook_rec)]
echo ![rpt(" ",9+pos(_spook_pattern,_spook_rec)) + &
       rpt("^",len(_spook_pattern))]
```

The latter example uses the rpt() function to precede the string substituted integer value of the variable **`_spook_curr_rec`** with the correct number of blanks so that the right parentheses line up, regardless of how many characters are in the integer. It then substitutes the value of **`_spook_rec`**, trimmed of blanks on the right, and echoes this line. The second ECHO command substitutes the number of blanks necessary to reach the position beneath the word to be highlighted and then substitutes a number of carets equal to the length of this word.

Pos() and Finfo()

Some of the changes to the CI for 2.1 are modifications of existing functions. Both the pos() and finfo() functions have been changed. The pos() function has been enhanced to give the CI programmer more control over searching for one string in another. As an example, suppose you have a string variable containing user input which contains a quoted string. If you wanted to test that there are characters between the quotes, you could use the old CI with the following commands:

```
setvar _spook_pos pos('"',_spook_parms)
setvar _spook_i pos('"',rht(_spook_parms,len(_spook_parms)-_spook_pos)
if _spook_i - _spook_pos <= 1 then
```

And with the new, 2.1 CI:

```
if setvar(_spook_i,pos('"',_spook_parms,-1)) - &
   setvar(_spook_pos,pos('"',_spook_parms)) <= 1 then
```

An optional third parameter has been added to pos(). The current behavior of `pos(".","file.pub.acct")` is to return the position of the string "." in the string "file.pub.acct", which is 5 in this case. The third parameter allows the user to specify which occurrence of the first parameter is to be sought, e.g. `pos(".","file.pub.acct",2) = 9`. This third parameter can also be used to specify that the search is to proceed from the right, e.g. `pos(".","file.pub.acct",-1) = 9`.

Finfo() has been modified to make it easier to use and self-documenting. The parameters to finfo() were a string representing the name of a file about which information is required, and an integer option, indicating the type of information. So, for example, if you wanted to know if the file HPBROWSE.PUB.SYS exists on your system, you would (using the old CI) execute the following command:

```
if finfo("hpbrowse.pub.sys",0) then
```

Of course, this requires that you know that an option number of zero will test for existence of the file. On 2.1, however, this command becomes:

```
if finfo("hpbrowse.pub.sys","exists") then
```

The old way is still supported, but the new way makes your command files (which will be used on 2.1) more readable.

Max(), Min() and Odd()

The other functions added to the CI for 2.1 are max(), min() and odd(). Max() returns the maximum of several integer expressions, e.g. `max(2,5+4,-3) = 9`. Min() does the opposite of max(), e.g. `min(2,5+4,-3) = -3`. Odd() returns TRUE if its parameter (which can be any integer expression) is odd and FALSE if it is even.

New Commands

Several commands have been added to the CI for MPE XL 2.1. Four are particularly useful for CI programming: ELSEIF, ERRCLEAR, PAUSE, and LISTFILE.

The ELSEIF command reduces the number of ENDIFs used when performing several consecutive tests. For example, entry points require tests of the form:

```
parm entry=main
if "!entry"="main" then
   comment main part
else
   if "!entry"="get_dfid_or_username" then
      comment get dfid part
   endif
endif
```

With ELSEIF this becomes:

```
parm entry=main
if "!entry"="main" then
   comment main part
elseif "!entry"="get_dfid_or_username" then
   comment get dfid part
endif
```

ERRCLEAR

The ERRCLEAR command is simply a convenient and fast way to set the values of the error-related HP predefined CI variables to zero. These variables are CIERROR and the new variables HPFSERR, HPCIERRCOL, and HPCIERR. The new variables and ERRCLEAR were added to give the CI programmer more control over error display. They can be used to display CI errors in a similar manner to the CI. HPCIERR has the same function as CIERROR, to return the integer value of the last CI error encountered. However, HPCIERR distinguishes between CI errors and warnings, since errors are positive and warnings are negative. HPCIERRCOL contains the position in the command line that the last error occurred. HPFSERR can be dereferenced to determine the last File System error which occurred. The existing HPCIERRMSG variable contains the error message text corresponding to the error number contained by CIERROR.

For example, consider the following excerpt from the SPOOK command file:

```
errclear
...
comment {Otherwise, try to execute an MPE command}
...
setvar hpmsgfence 2
continue
!_spook_rec
setvar hpmsgfence 0

if hpcierr = 975 then
   echo Invalid command name, use HELP.
elseif hpcierr <> 0 then
   if hpcierrcol > 0 then
      echo ![rpt(" ",hpcierrcol+1)]^
   endif
   if hpfserr > 0 then
      echo FILE SYSTEM ERROR NUMBER: !hpfserr
   endif
   echo !hpcierrmsg
endif
```

PAUSE

The PAUSE command is useful when you want to create a command file which will check for some event periodically. For example, you could synchronize two jobs through the existence of a file. The jobs in this example have some processing they can perform concurrently and independently, but JSECOND is dependent on some data produced by JFIRST. JSECOND can perform its initial processing, but then it must wait to continue processing until JFIRST reaches a certain point. The jobs could be created on the old CI as follows:

```
Job stream JFIRST:

!job jfirst,me.myacct
!comment Do some stuff which takes awhile and must be done before
JSECOND
!comment gets to the synchronization point.
!build semafile
!comment Do other stuff
!eoj

Job stream JSECOND:

!job jsecond,me.myacct
!comment Do stuff which can be done before JFIRST gets finished with its
!comment initialization.
!while not finfo("semafile",0) do
!  listf @;$null
!endwhile
!comment Continue part of job which depends on JFIRST.
!eoj
```

The problem with this version of these jobs is that the WHILE loop in JSECOND is wasted processing. On the MPE XL 2.1 CI, the following loop would do a much better job:

```
!while not finfo("semafile",0) do
!  pause 3
!endwhile
```

The PAUSE command pauses for the number of seconds specified by its parameter. The pause is done in such a manner that the process does not consume CPU while paused. This means that JSECOND will check for the file, wait one second, and check for the file again. PAUSE can be terminated by BREAK and can not exceed the time specified by the HPTIMEOUT variable (if it is set).

PAUSE is also useful if you want to display a message for a short period of time prior to entering a program which will clear the screen. One such application of PAUSE occurs in the SPOOK command file (the following is an excerpt):

```
  if not bound(hpeditor) then
     echo HPEDITOR variable not defined, using 'HPEDIT.PUB.SYS'
     pause 2
     setvar hpeditor "HPEDIT.PUB.SYS"
  endif
  ...
```

This CI command file fragment first checks if the variable HPEDITOR has been defined by the user. This variable is used by the SPOOK command file to determine what editor should be used when editing spool files. If the variable doesn't exist, SPOOK displays a message to inform the user, waits two seconds, then sets the variable so HPEDIT.PUB.SYS will be used. When HPEDIT is run, it clears the screen, so the PAUSE allows the message to be read.

LISTFILE

The other new command of interest to CI programmers is LISTFILE. This command is a replacement for LISTF and LISTFTEMP. These old commands will always exist for backward compatibility, but they may not be enhanced. Any future enhancements will be added to LISTFILE. Some of the desired enhancements could not be done within the MPE V-style syntax of LISTF and LISTFTEMP.

LISTFILE combines the features of LISTF and LISTFTEMP in one command. The options ;PERM, ;TEMP and ;PERMTEMP can be used to choose to display information about only temporary, only permanent, or both temporary and permanent files.

LISTFILE allows multiple filesets to be specified. It also uses the same MPE XL wildcarding features provided through the SETVAR, DELETEVAR and Native Mode STORE commands. This wildcarding allows sets of characters and ranges of characters to be given as a match for a character. For example:

```
  listfile [oe-g]@
```

This command will list all files whose name begins with an "O" or an "E", "F" or "G".

LISTFILE uses the MPE XL parser, allowing a more flexible syntax. T the type of information displayed can still be controlled through the old "magic" numbers, e.g. `:listfile @,2` will be equivalent to `:listf @,2`. However, each number has a corresponding alias which is related to the type of information and easier to remember. For example, `:listfile @,3` is equivalent to `:listfile @,detail` because this form of LISTFILE output is more detailed than the `,2` form.

LISTFILE also provides the ability to display information specific to the type of a file. If the FORMAT parameter (the second parameter) of LISTFILE is DATA or UNIQUE, information specific to KSAMXL and SPOOL files will be displayed for each file of that type. The SELEQ parameter also allows selection of the files to be listed by file type. For example:

```
  listfile @,qualify;seleq=[ftype=ksamxl] >myksams
```

This command will list the fully qualified file names (`qualify`) of all files in the current group which are Native Mode KSAM files. The list will be sent to the file MYKSAMS.

New User Command Features

ANYPARM

If you have ever wanted to pass a string to a User Command (UDC or command file) without having it be parsed by the CI, you now have a way. In addition to having the

PARM line on a User Command, there is now a feature called the ANYPARM line. An ANYPARM must be the last parameter specified for a User Command. For example, a UDC to do TELLs in inverse video could be created using the old CI:

```
SAY user,p1="",p2="",p3="",p4="",p5="",p6="",p7="",p8="",p9="",&
    p10="",p11="",p12=""
tell !user;![chr(27)]&dJ !p1 !p2 !p3 !p4 !p5 !p6 !p7 !p8 !p9 &
          !p10 !p11 !p12
```

Suppose this UDC were invoked as follows:

```
:say sally.accting;hi there, let's meet now; I will bring Joe.
```

When the TELL command is executed, it would be:

```
tell sally.accting;hi there let's meet now I will bring Joe.
```

The message part would be in inverse video. Of course, another word in the message would be treated as another parameter, causing an error. The commas and semicolons have also been parsed out of the message, since they were taken as delimiters on the original invocation of the UDC. The same UDC on the 2.1 CI would be:

```
SAY user
ANYPARM message
tell !user;![chr(27)]&dJ !message
```

Given the same invocation of SAY, the TELL command would become:

```
tell sally.accting;hi there, let's meet now; I will bring Joe.
```

Of course, the message part would be in inverse video.

When a User Command has an ANYPARM and the CI detects that it is finished processing the other parameters of the User Command, the rest of the line is passed as the ANYPARM. No parsing is done on this part of the line.

ANYPARM is also useful when you need to pass unparsed user input to an entry point of your own command file. This is the way the SPOOK command file uses ANYPARM. SPOOK takes interactive input from a user. Some of the input, such as DFID specifications, must be decoded by several different parts of the command file. Rather than putting the commands to perform this function every place they are needed, SPOOK uses entry points. When the entry point which converts user input to spool file names is called, it must be passed the user's input in an unparsed form, since there may be an unknown number of parameters specified. This input is passed in an ANYPARM.

The SPOOK Command File

The SPOOK command file was developed by Jeff Vance of the Hewlett Packard MPE Lab. It is intended as a tool for transition to and teaching of the New Commercial Spooler, which is also new with MPE XL 2.1. It uses the same syntax which is used by the SPOOK program on earlier releases of MPE. The SPOOK program is no longer used as of

MPE XL 2.1. Instead, several new and old commands replace its functionality. The SPOOK command file can be used to transition from pre-2.1 releases. If a job exists which used the SPOOK program, it may still work unchanged with the SPOOK command file. This command file can also be used as a tool for learning the new commands, because it displays the commands used to emulate the functionality of each SPOOK command, before executing them. The new commands provide much more power than the old commands or SPOOK, but they are different and must be learned. An example of their power is that one of the earlier CI programming papers which came out of our Lab used a command file as an example which deleted all spool files owned by a given user. The functionality of that command file, which is over 50 lines in length, can be achieved with the following command:

```
:spoolf @;seleq=[owner=user.acct];delete
```

The SPOOLF command performs several operations on spool files, one of which is deleting them. It has extensive selection capabilities. This command specifies to delete all the spool files accessible by the person executing the command which are owned by USER.ACCT. The selection equation (SELEQ) could have included many other criterion for deciding which spool files to delete.

The SPOOK command file uses many features of the new MPE XL 2.1 CI and solves a "real-world" problem, which is why it makes a good example for this paper. It may be used in several ways. By reading through it, you will learn many of the techniques of CI programming which can be used to solve other problems. One of the new techniques presented in this example is the version history at the end, showing how version history can be included in the command file itself. In addition, you can see the use of the new 2.1 features of the CI, which will give you a reason to look forward to MPE XL, Release 2.1.

Here is the text of the SPOOK command file:

```
PARM    entry=main
PARM    cmd=""
ANYPARM cmdparms=null
comment This command file translates the SPOOK command set to the NM Spooler.
comment Written by Jeff Vance (CSY) February 1990.

if "!entry" = "get_dfid_or_username" then
   comment (  Parses dfid or list of dfid's or user[.acct] name            )
   comment (Input:  cmdparms,     contains difd/s or user.acct             )
   comment (        _spook_parms, everything right of command name         )
   comment (Output: _spook_dfid,  list of dfid/s, or "@" if user.acct used )
   comment (        _spook_opts,  owner= seleq if user.acct used, else nil )
   comment (        _spook_parms, _spook_parms without dfids or username   )
   comment
   if numeric(lft("!cmdparms",1)) or lft("!cmdparms",1) = "#" or &
      lft("!cmdparms",1) = "*" then
      comment (Dfid [,dfid,...,dfid] syntax)
      setvar _spook_dfid "(!cmdparms)"
      comment (Change '*' to currently texted dfid)
      if setvar(_spook_pos,pos('*',_spook_dfid)) > 0 and &
         bound(_spook_text) then
         setvar _spook_dfid lft(_spook_dfid,_spook_pos-1)+"#!_spook_textid"+&
                rht(_spook_dfid,len(_spook_dfid)-_spook_pos)
      endif
      comment (Get rid of any extra "*"s)
      while pos('*',_spook_dfid) > 0 do
         setvar _spook_dfid _spook_dfid-"*"
      endwhile
   else
      comment {Parse user[.acct] name}
      if setvar(_spook_opts,"!cmdparms") = "null" then
         setvar _spook_opts "!hpuser.!hpaccount"
      endif
      setvar _spook_opts ";SELEQ=[OWNER=!_spook_opts]"
      setvar _spook_dfid "@"
   endif
   comment {Return _spook_parms without dfid(s) or user.acct name}
   setvar _spook_parms ltrim(_spook_parms-"!cmdparms")
   return

elseif "!entry" = "get_range" then
   comment {  Parses a range into starting and ending record numbers.       }
   comment {  Handles #[/#],"*",+,-,FIRST,LAST.                             }
   comment {Input:  cmdparms,      contains range                           }
   comment {        _spook_curr_rec, record number for current text file    }
   comment {Output: _spook_start, integer starting record number for :print }
   comment {        _spook_end,   integer ending record number for :print   }
   comment
   if "!cmdparms" = "null" then
      comment {No range specified, use default range of current line}
      setvar _spook_start _spook_curr_rec
      setvar _spook_end _spook_start
   elseif "!cmdparms" = "ALL" then
      setvar _spook_start 1
      setvar _spook_end _spook_last

   else
      comment {Parse start [/end] specification}
      if setvar(_spook_pos,pos('/',"!cmdparms")) > 0 then
         setvar _spook_start lft("!cmdparms",_spook_pos-1)
         setvar _spook_end str("!cmdparms",_spook_pos+1,&
                len("!cmdparms")-_spook_pos)
      else
         setvar _spook_start "!cmdparms"
         setvar _spook_end _spook_start
      endif
      comment {Compute the actual start and end (account for "LAST", "*",
      comment  etc)}
      setvar _spook_i 0
      setvar _spook_num _spook_start
      while _spook_i <= 1 do
         comment {Loop twice}
         if setvar(_spook_pos,pos('*',_spook_num)) > 0 then
            setvar _spook_num lft(_spook_num,_spook_pos-1)+&
                              "!_spook_curr_rec"+&
                              rht(_spook_num,len(_spook_num)-_spook_pos)
         endif
         comment {Handle FIRST and LAST}
         if setvar(_spook_pos,pos('FIRST',_spook_num)) > 0 then
            setvar _spook_num lft(_spook_num,_spook_pos-1) + "1" +&
                   rht(_spook_num,len(_spook_num)-_spook_pos-4)
         elseif setvar(_spook_pos,pos('LAST',_spook_num)) > 0 then
```

```
            setvar _spook_num lft(_spook_num,_spook_pos-1) + "!_spook_last" +&
                   rht(_spook_num,len(_spook_num)-_spook_pos-3)
          endif
          comment (Do arithmetic and convert to an integer)
          if _spook_i = 0 then
            setvar _spook_start !_spook_num
            setvar _spook_num _spook_end
            setvar _spook_i 1
          else
            setvar _spook_end !_spook_num
            setvar _spook_i 2
          endif
        endwhile
      endif
      return

    elseif "!entry" = "find" then
      comment {  Finds first occurence of "_spook_pattern" in the _spook_textid }
      comment {  file. Input has been redirected from this file.                }
      comment {Input:  _spook_pattern,  case sensitive string to match in file  }
      comment {        _spook_curr_rec, current record number for _spook_textid }
      comment {Output: _spook_curr_rec, number of next record (or last) after   }
      comment {        match                                                    }
      comment {        _spook_found,    true means pattern was found            }
      comment
      setvar _spook_found false
      while _spook_curr_rec <= _spook_end and not (_spook_found) do
        comment (Read next record and strip 1st byte if CCTL file)
        if len(setvar(_spook_rec,input())) > 0 and _spook_text_cctl then
          setvar _spook_rec ltrim(rht(_spook_rec,len(_spook_rec)-1))
        endif
        comment (Check if this record matches pattern)
        if (_spook_any_match and pos(_spook_pattern,_spook_rec) > 0) or &
          (not _spook_any_match and pos(_spook_pattern,_spook_rec) = 1) then
          comment (Match! Echo record with leading right-justified line number)
          echo ![rpt(" ",8-len("!_spook_curr_rec"))]!_spook_curr_rec) &
               ![rtrim(_spook_rec)]
          comment (Echo carets(^^) under matching word)
          echo ![rpt(" ",9+pos(_spook_pattern,_spook_rec))+&
                 rpt("^",len(_spook_pattern))]
          comment (Stop while loop)
          setvar _spook_found true
        endif
        setvar _spook_curr_rec _spook_curr_rec+1
      endwhile
      if _spook_curr_rec > _spook_last then
        setvar _spook_curr_rec _spook_last
      endif
      return

    elseif "!entry" = "main" then
      comment (Main entry for SPOOK)
      if "!cmd" = "" then
        echo
        echo SPOOK Emulator  Version 2.1
        comment (Initialize user environment)
        setvar _spook_noise 'LOUD'
```

```
endif

setvar _spook_looping true
while _spook_looping do
   setvar _spook_cmd ""
   if "!cmd" <> "" then
      comment (Interface for internally executing spook commands)
      comment (Assumes that "cmd" is upshifted and "cmdparms" is)
      comment (terminated by a chr(0).                         )
      setvar _spook_cmd "!cmd"
      setvar _spook_parms "!cmdparms"
      comment (Keep internal calls quiet)
      setvar _spook_noise_save _spook_noise
      setvar _spook_noise 'QUIET'

   comment (Prompt and read user input)
   elseif len(setvar(_spook_rec,ltrim(rtrim(input("> ")))+chr(0))) > 1 then
      comment (Parse command name)
      setvar _spook_i 1
      while alpha(str(_spook_rec,_spook_i,1)) do
         setvar _spook_i _spook_i+1
      endwhile
      setvar _spook_cmd ups(lft(_spook_rec,_spook_i-1))
      comment (Separate command parameters)
      setvar _spook_parms_raw ltrim(rht(_spook_rec,len(_spook_rec)-&
            _spook_i+1))
      setvar _spook_parms ups(_spook_parms_raw)
   endif

   if len(_spook_rec) > 1 then
      setvar _spook_opts ''
      setvar _spook_dfid ''
      errclear

      comment (Case _spook_cmd of...)
      comment ****** ALTER ******
      if _spook_cmd = "A" or _spook_cmd = "ALT" or _spook_cmd = "ALTER" &
         then
         comment (Convert to SPOOLF...;ALTER)
         if len(_spook_parms) = 1 then
            echo Expected a "user[.acct]" or "dfid" parameter.
         else
            comment (Parse the dfid[s] or user[.acct] name)
            xeq spook entry="get_dfid_or_username" cmdparms=&
                 ![lft(_spook_parms,pos(';',_spook_parms)-1)]
            comment (Remove leading ';' in spook_parms (rtn by
            comment  get_dfid_..))
            setvar _spook_parms _spook_parms-';'
            while len(_spook_parms) > 1 do
               setvar _spook_i pos('=',_spook_parms) + 1
               if setvar(_spook_pos,pos(',',_spook_parms)) = 0 then
                  setvar _spook_pos len(_spook_parms)
               endif
               comment (Look for COPIES= ,DEV=, PRI= options)
               if lft(_spook_parms,1) = "C" then
                  setvar _spook_opts _spook_opts + ";COPIES="
               elseif lft(_spook_parms,1) = "D" then
            setvar _spook_opts _spook_opts + ";DEV="
         elseif lft(_spook_parms,1) = "P" then
            setvar _spook_opts _spook_opts + ";PRI="
         endif
         comment (Append value to above option)
         setvar _spook_opts _spook_opts + &
                str(_spook_parms,_spook_i,_spook_pos-_spook_i)
         comment (Remove this option from _spook_parms)
         setvar _spook_parms ltrim(rht(_spook_parms,&
                len(_spook_parms)-_spook_pos))
      endwhile
      if _spook_noise = 'LOUD' then
         setvar _spook_opts _spook_opts + ';SHOW'
         echo   spoolf !_spook_dfid !_spook_opts ;ALTER
      endif
      continue
      spoolf !_spook_dfid !_spook_opts ;ALTER
   endif


comment ****** BROWSE ******
elseif _spook_cmd = "B" or _spook_cmd = "BROWSE" then
   comment (New SPOOK command: Syntax: browse [dfid])
   comment (Invokes hpbrowse passing dfid or current text file)
   if not finfo("hpbrowse.pub.sys","exists") then
      echo HPBROWSE.PUB.SYS does not exist on this system.
      setvar hpcierr 52
   elseif len(_spook_parms) > 1 and lft(_spook_parms,1) <> "*" then
      comment (Supplied the dfid)
      comment (Cause passed dfid to become the current text file)
      xeq spook cmd="TEXT" cmdparms=!_spook_parms
   elseif not bound(_spook_text) then
      echo No spoolfile specified.
      setvar hpcierr 1
   elseif _spook_noise = 'LOUD' then
      echo Browsing !_spook_text...
      pause 1
   endif
   if hpcierr = 0 then
      comment (Invoke hpbrowse passing the dfid file)
      continue
      xeq hpbrowse.pub.sys "!_spook_text"
   endif


comment ****** EDIT ******
elseif _spook_cmd = "EDIT" then
   comment (New SPOOK command: Syntax: edit [dfid])
   comment (Invokes hpeditor passing dfid or current text file)
   if not bound(hpeditor) then
      if _spook_noise = 'LOUD' then
         echo HPEDITOR variable not defined, using 'HPEDIT.PUB.SYS'
         pause 2
      endif
      setvar hpeditor 'HPEDIT.PUB.SYS'
   endif
   if not finfo(hpeditor,"exists") then
```

```
      echo Non-existent editor program: !hpeditor.
      setvar hpcierr 52
   elseif len(_spook_parms) > 1 and lft(_spook_parms,1) <> "*" then
      comment (Supplied the dfid)
      comment (Cause passed dfid to become the current text file)
      xeq spook cmd="TEXT" cmdparms=!_spook_parms
   elseif not bound(_spook_text) then
      echo No spoolfile specified.
      setvar hpcierr 1
   elseif _spook_noise = 'LOUD' then
      echo Editing !_spook_text...
      pause 1
   endif
   if hpcierr = 0 then
      comment (Invoke the editor passing the dfid file)
      continue
      !hpeditor "!_spook_text"
   endif


comment ****** EXIT / QUIT ******
elseif _spook_cmd = "E"    or _spook_cmd = "Q"    or &
       _spook_cmd = "EXIT" or _spook_cmd = "QUIT" then
   setvar _spook_looping false


comment ****** FIND ******
elseif _spook_cmd = "F" or _spook_cmd = "FIND" then
   if not bound(_spook_text) then
      echo No text file, see TEXT command.
   elseif len(_spook_parms) <= 4 and not bound(_spook_pattern) then
      comment (Handles: F, F", F"", F@" + chr(0))
      echo No search pattern defined.
   else
      comment (Initialize range of text to be searched)
      setvar _spook_start _spook_curr_rec
      setvar _spook_end _spook_last
      if len(_spook_parms) > 4 then
         comment (Compute length of pattern)
         if setvar(_spook_i,pos('"',_spook_parms,-1)) - &
            setvar(_spook_pos,pos('"',_spook_parms)) <= 1 then
            if pos(',',_spook_parms) = 0 then
               comment (No range parameter therefore null pattern)
               echo Null pattern not allowed.
               setvar hpcierr 1
            endif
         else
            comment (Use "raw" buffer to handle lowercase patterns)
            setvar _spook_pattern str(_spook_parms_raw,_spook_pos+1,&
                   _spook_i-_spook_pos-1)
            setvar _spook_any_match lft(_spook_parms,1) = "@"
            comment (Strip pattern from _spook_parms for range check)
            setvar _spook_parms ltrim(rht(_spook_parms,&
                   len(_spook_parms)-_spook_i))
         endif
      endif
```

```
            if hpcierr = 0 and len(_spook_parms) > 2 and &
               lft(_spook_parms,1) = "," then
               comment (Range parameter specified)
               xeq spook entry="get_range" cmdparms=![_spook_parms-chr(0)]
               setvar _spook_curr_rec _spook_start
            endif

            if hpcierr = 0 then
               comment (Copy unsearched portion of dfid file to search)
               setvar hpmsgfence 1
               print !_spook_text;start=!_spook_start;end=!_spook_end;&
                     page=0 >!_spook_textid
               setvar hpmsgfence 0
               comment (Invoke spook to find first occurence in local file)
               xeq spook entry="find" <!_spook_textid
               purge !_spook_textid,temp
               if not _spook_found and _spook_noise = 'LOUD' then
                  echo "!_spook_pattern" not found.
               endif
            endif
         endif


      comment ****** HELP ******
      elseif _spook_cmd = "H" or _spook_cmd = "HELP" then
echo A[LTER]  user[.acct]        [;pri= [,copies=] [,dev=]]  |:SPOOLF ;ALTER
echo          dfid [,dfid [,...]] [;pri= [,copies=] [,dev=]]  |
echo B[ROWSE] [dfid] *New* Invokes HPBROWSE                   |:HPBROWSE Onnn
echo EDIT     [dfid] *New* Invokes !!HPEDITOR                 |(your editor)
echo E[XIT] or Q[UIT]                                         |
echo F[IND]   [[@] "pattern"] [,range]                        |(your editor)
echo I[NPUT]  user[.acct]        ;tapefile                    |:SPFXFER
echo          dfid [,dfid [,...]];tapefile                    |
echo L[IST]   [range]                                         |:PRINT
echo O[UTPUT] user[.acct]        ;tapefile [;PURGE]           |:SPFXFER
echo          dfid [,dfid [,...]];tapefile [;PURGE]           |
echo P[URGE]  user[.acct]                                     |:SPOOLF ;DELETE
echo          dfid [,dfid [,...]]                             |
echo SET      [loud | quiet] *New*                            |
echo S[HOW]   [user[.acct]] [;[@] [I] [O]]                    |:LISTSPF
echo          dfid [,dfid [,...]]                             |
echo T[EXT]   [dfid]                                          |
echo [:]!<MPE XL command!>                                     |
echo _________________________________________________________|
echo !<dfid!>  ::= [#O]n or #In or *
echo !<range!> ::= n [/n] or n+n or n-n or FIRST or LAST or ALL or *
echo !<n!>     ::= one or more digits, 0..9
echo !<*!>     ::= current object (current dfid or current position in file)


      comment ****** INPUT / OUTPUT ******
      elseif _spook_cmd="I" or _spook_cmd="IN" or _spook_cmd="INPUT" or &
             _spook_cmd="O" or _spook_cmd="OUT" or _spook_cmd="OUTPUT" then
         comment (Convert to SPFXFER)
         echo ![_spook_rec-chr(0)] >spfxin
         echo EXIT >>spfxin
         if _spook_noise = 'LOUD' then
      echo   Commands for SPFXFER are:
      print spfxin
      echo
   endif
   continue
   xeq spfxfer.pub.sys <spfxin
   purge spfxin,temp


comment ****** LIST ******
elseif _spook_cmd = "L" or _spook_cmd = "LIST" then
   comment (Convert to PRINT)
   if not bound(_spook_text) then
      echo No text file, see TEXT command.
   else
      comment (Pass list range to spook for parsing)
      xeq spook entry="get_range" cmdparms=![_spook_parms-chr(0)]
      comment (Print the dfid file for specified range)
      setvar hpmsgfence 1
      continue
      print !_spook_text;start=!_spook_start;end=!_spook_end;num
      setvar hpmsgfence 0
      comment (Increment current line count)
      setvar _spook_curr_rec _spook_end+1
      if _spook_curr_rec > _spook_last then
         setvar _spook_curr_rec _spook_last
      endif
   endif


comment ****** PURGE ******
elseif _spook_cmd = "P" or _spook_cmd = "PURGE" then
   comment (Convert to SPOOLF...;DELETE)
   if len(_spook_parms) = 1 then
      echo Expected "user[.acct]" or "dfid" parameter.
   else
      xeq spook entry="get_dfid_or_username" cmdparms=&
                 ![_spook_parms-chr(0)]
      if _spook_noise = 'LOUD' then
         setvar _spook_opts _spook_opts + ';SHOW'
         echo   spoolf !_spook_dfid !_spook_opts ;DELETE
      endif
      continue
      spoolf !_spook_dfid !_spook_opts ;DELETE
      comment (See if we purged the current text file)
      if bound(_spook_text) and not finfo(_spook_text,"exists") then
         deletevar _spook_text@
      endif
   endif


comment ****** SET ******
elseif _spook_cmd = "SET" then
   if len(_spook_parms) = 1 then
      comment (No parm, display current value)
      echo You are currently !_spook_noise.
   else
```

```
      setvar _spook_parms _spook_parms-chr(0)
      if _spook_parms = 'LOUD' or _spook_parms = 'QUIET' then
         setvar_spook_noise _spook_parms
      else
         echo Expected 'LOUD', 'QUIET' (or nothing to see current &
              value).
      endif
   endif


comment ****** SHOW ******
elseif _spook_cmd = "S" or _spook_cmd = "SHOW" then
   comment (Convert to LISTSPF)
   if len(_spook_parms) > 1 then
      comment (Parse the dfid[s] or user[.acct] name)
      if setvar(_spook_pos,pos(';',_spook_parms)) = 0 then
         setvar _spook_pos len(_spook_parms)
      endif
      xeq spook entry="get_dfid_or_username" cmdparms=&
               ![lft(_spook_parms,_spook_pos-1)]
   endif

   if len(_spook_opts) > 0 then
      comment (User[.acct] specified)
      comment (Parse additional SHOW options)
      if pos('@',_spook_parms) > 0
        setvar _spook_opts _spook_opts + ";DETAIL"
      endif
      if pos('O',_spook_parms) > 0 then
         setvar _spook_dfid "O@"
      elseif pos('I',_spook_parms) > 0 then
         setvar _spook_dfid "I@"
      endif
   elseif len(_spook_dfid) > 0 then
      comment (Use ;detail when dfid(s) specified)
      setvar _spook_opts ";DETAIL"
   endif
   if _spook_noise = 'LOUD' then
      echo   listspf !_spook_dfid !_spook_opts
   endif
   continue
   listspf !_spook_dfid !_spook_opts


comment ****** TEXT ******
elseif _spook_cmd = "T" or _spook_cmd = "TEXT" then
   comment (All text-related variable names begin with "_spook_text")
   if len(_spook_parms) = 1 or lft(_spook_parms,1) = "*" then
      if bound(_spook_text) then
         if _spook_noise = 'LOUD' then
            echo File !_spook_text is un-texted.
         endif
         deletevar _spook_text@
      endif
   else
      setvar _spook_textid _spook_parms-chr(0)-"#"
      if numeric(lft(_spook_textid,1)) then
```

```
            setvar _spook_textid "0" + _spook_textid
         endif
         setvar _spook_text _spook_textid + ".OUT.HPSPOOL"
         if finfo(_spook_text,"exists") then
            comment (Initialze starting and ending record numbers)
            setvar _spook_curr_rec 1
            setvar _spook_last finfo(_spook_text,"eof")
            setvar _spook_text_cctl &
                   pos('NOCCTL',finfo(_spook_text,"fmtfopt")) = 0
            if _spook_noise = 'LOUD' then
               echo Text file is: !_spook_text.
               echo Eof: !_spook_last. Created on: &
                    ![finfo(_spook_text,"fmtcreated")] at &
                    ![finfo(_spook_text,"fmtalloctime")].
            endif
         else
            comment (Dfid does not exist, cleanup)
            echo Non-existent device file (!_spook_text).
            setvar hpcierr 52
         endif
      endif


   comment ****** MPE COMMAND ******
   else
      comment (Otherwise, try to execute an MPE command)
      setvar _spook_rec _spook_rec - chr(0)
      if lft(_spook_rec,1) = ":" then
         setvar _spook_rec _spook_rec - ":"
      endif

      setvar hpmsgfence 2
      continue
      !_spook_rec
      setvar hpmsgfence 0

      if hpcierr = 975 then
         echo Invalid command name, use HELP.
      elseif hpcierr <> 0 then
         if hpcierrcol > 0 then
            echo ![rpt(" ",hpcierrcol+1)]^
         endif
         if hpfserr > 0 then
            echo FILE SYSTEM ERROR NUMBER: !hpfserr
         endif
         echo !hpcierrmsg
      endif
   endif
endif

comment (Return if using internal spook command interface)
if "!cmd" <> "" then
   comment (Restore noise level)
   setvar _spook_noise _spook_noise_save
   deletevar _spook_noise_save
   return
endif
```

```
   endwhile
   comment (Cleanup!)
   deletevar _spook_@
endif
return
comment End of SPOOK!
                ************ Version History *************

1.0   Initial (limited) release supporting alter,edit,exit,find,help,list,
      purge,show,text, and an MPE interface. SPOOK is ONE file with NO
      PROCESS CREATES so far...
1.1   Fixed text bug where finfo was called to get _spook_last even if
      the dfid did not exist in .out.hpspool.
      Re-wrote list to use the new "get_range" entry point. This will later
      be used by find.
      Fixed bug where we strip the cctl char on nocctl files.
1.2   Added ",range" to find.
1.3   Some minor cleanup based on Steve Elmer's suggestions. Added test for
      existence of hpeditor and changed ";I@"and ";O@" logic into elseif. I
      also added some help text.
1.4   Fixed bug where :cmds were not executed since len(_spook_cmd) = 0.
1.5   Added browse,input,output commands. Changed get_dfid_or_username to
      use the "cmdparms" parameter as input.
1.6   Fixed bug when range is used with find. Changed get_range entry to not
      expect chr(0) in cmdparms.
1.7   Fixed bug where spfxfer was spelled spxfer. Added logic to purge to
      delete all text variables if _spook_text was one of the files purged.
1.8   Added set [loud|quiet] command, and incorporated this into alter,input,
      output,purge,text.
1.9   Fixed bug where 'exit' was append to the wrong file for in/out cmds.
1.10  In 'loud' mode the creation date is displayed when a file is texted.
2.0   Official BARUG version. Cleaned up some flaws Scott Cressler found.
2.1   Very minor quotes and indentation cleanup, added a few comments, echo
      the ;show option when loud, echo the dfid in browse and edit when not
      passed and loud. Also added _spook_noise_save so that internal calls
      don't echo when the outer level is loud.
```

Maintaining the Integrity of your Software

Alan Padula

Hewlett-Packard Company
Santa Clara Information Systems Division
Santa Clara, California

May 1, 1990

"SCM is not just for the BIG!"

INTRODUCTION

The premise of this paper is that any size programming shop can improve the integrity of their software by adopting an appropriate level of Software Configuration Management (SCM) tools and procedures for their needs. Formal SCM practices have traditionally and primarily been used in very large projects. But SCM fundamentals can be incorporated into "normal" sized commercial shops too. The key lies in selectively choosing the appropriate level of both its tools and processes for the particular environment. This paper focuses on the tool aspect of SCM and more specifically, the main classes of tools a commercial shop should consider to improve the integrity of their software.

A more formal overview of SCM is presented and the characteristics of the very large environments it has traditionally thrived in are examined first. This provides insight into why SCM is sometimes complex and how it initially gained a discouraging reputation for requiring a significant overhead. Additional common objections from the commercial user, and perhaps some that you may have, are also presented.

An alternative, practical model is then proposed that simplifies SCM into four main tool classes that better addresses the problems & needs of the typical, smaller commercial development shop. Those classes are:

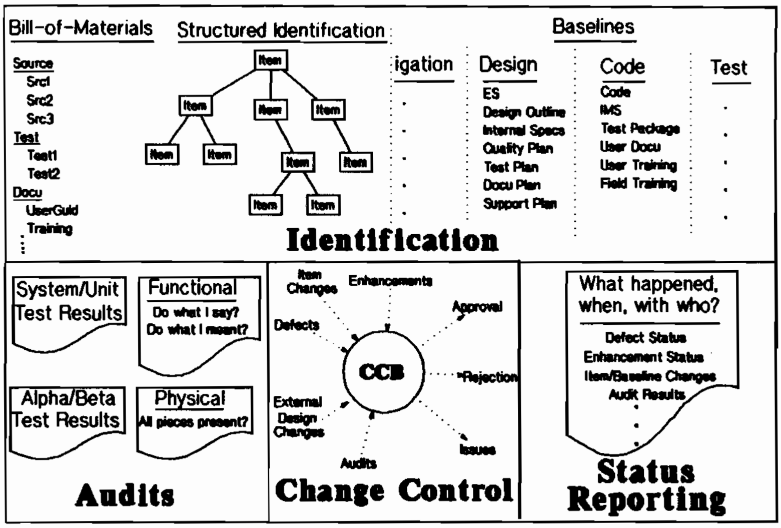
- **Configuration Management Specification & Reporting,**
- **Automated Build Management,**
- **Version Control,**
- **Change Control.**

A description of each of these classes, some of the classic commercial user problems associated with them, and the corresponding standard features that are fundamental to solving them is revealed. These classes do not mean to constitute the set of *all* tools used in SCM. They do provide a good starting base for the commercial user to consider when SCM tools are being sought. It is then shown how the commercial user, by choosing the proper degree of SCM tools and processes (instead of the same breadth and depth needed by the original, very large target customer) can also reap the benefits of:

- **Integrity of their system,**
- **Productivity of their developers,**
- **Control of their process.**

This paper ends with a short potpourri of advanced, "sizzle" functionality to generate further interest.

SCM MODEL FOR LARGE CUSTOMERS



The Traditional SCM Model

So what is the formal industry definition of SCM? If you removed the enormous amount of detail* it could be defined as a discipline of tools and processes used to control the development and maintenance of software. SCM includes the organization, management, and tracking of software configuration items as a single unit. Those items include specification documents, designs, test plans, source files, program files, etc. and their transformation through time. Of premiere importance is the systematic control of change...either planned or crisis management.

Identification

Configuration Identification refers to the naming of all the items that make up a system and their relationship with one another. This means that if a programming module is made up of many different files, then those files might be hiearchically linked as "children" (see Structured Identification in figure). Another example of this simple concept is the linking of chapters in a book, paragraphs in a chapter, sentences in a paragraph, etc. Configuration Identification is key in providing other "documentation type" links such as connecting a paragraph number in a requirements document to the designed feature in an external specifications document to the actual piece of code that is the implementation of that external specification. All of these "pieces" are assigned unique configuration item names and are linked together in a logical fashion. All of the configuration items that make up or are used in the construction of a particular version of an application should be identified in a list structure (see Bill-of-Materials in figure). These are all important in providing traceability over time and establishing the definition of different baselines.

*Please see the Bibliography for more in-depth "textbook" definitions of SCM.

A **baseline** is a significant, formally approved, time-oriented snapshot of a software configuration, including the status of its components, the project plan, and the status of its change requests (bugs and/or enhancements) from which change control can be based. There are many baselines throughout a product's lifecycle. Examples of baselines include those defined as the state of the system at the completion of the requirements definition phase, the external specifications phase, the coding phase (and maybe several intermediate ones representing progressively greater states of functionality), etc. Baselines are necessary to perform the key function of change control. Without a stable, well-defined plan of the system, it is impossible to declare what changes have been approved to that plan.

Change Control

Change Management is the formal decision process for approving or disapproving proposed changes to a baseline. If a bug fix, enhancement, or change to the configuration (as already defined in a baseline) is to be made, a formal approval must first occur. This change to the baseline is then documented so everyone knows about it. This could be critical if someone else is affected by the change. (Before a change is approved, an impact analysis is generally performed. This identifies the magnitude of the change and the scope of other people affected.) The Configuration Control Board (CCB) is the body of people who formally approve or disapprove of proposed changes.

At HP, the CCB is often what we call a Product Team. It is a group of people gathered from the different functional areas necessary to produce a product. This includes at least one representative from the Research & Development Lab, Product Marketing, Learning Products (manuals and training), Quality Assurance, Online Support, Manufacturing, and so on. Other organizations may have a slightly different mix of people.

Audits

There are two fundamental types of SCM configuration audits, one to verify that all components are physically present and another to validate that the product performs as specified.

Physical Configuration Audits (PCA) are performed to check to see that all the pieces of a system are physically there. As virtually everything that makes up the system has been assigned a configuration item name, it is a rather straightforward (albeit tedious) task. Note that everything from the software to the user's guide should have a unique configuration item name. This compilation, sometimes referred to as a software Bill-Of-Materials, is checked during a PCA.

Functional Configuration Audits (FCA) are performed to validate that the product performs as specified and expected. Results of the unit and system tests are analyzed. Alpha and Beta test sites (internal and external customers) are polled to see if the product meets their original needs and meets company quality standards. This is done by mapping the documented *expected functionality* to what actually happens during testing, and then performing a cross-verification to see if that *expected functionality* actually meets the customer's original need and intent. The external specification for a product can be followed and the software can perform precisely as specified in it; however, if the external specification is initially flawed and the specification does not actually meet the original customer requirement, the end result is a product that performs flawlessly in the execution of the wrong purpose. In other words:

- **Does the product perform as defined in the external specifications document?**
- **Does the product meet the original customer requirements and needs?**

This auditing is facilitated by the "traceability" that SCM provides through uniquely named configuration items linked in its configuration structures. Logical links of a feature specified in a paragraph of a requirement's document point to the physical source files that have been coded to perform it.

Status Reporting

Status reporting refers to the production of reports to establish the current state of the project and its components. This includes defect report status, enhancement report status, configuration item and baseline changes, audit test results, etc. Timely access to this kind of information is essential to managing a large project, especially if it spans organizations.

It is obvious that generating reports on large configurations, its items, and its changes is greatly facilitated when that information is entered and tracked through a database. It is generally accepted today that it requires a database to effectively practice SCM in *large* projects. This is not a requirement for smaller projects though.

So... what kind of a customer would need mechanisms as formal and all-encompassing as just described?

SOME CHARACTERISTICS OF LARGE SCM CUSTOMERS

The target customer's needs that evolved into what is now called Software Configuration Management are quite interesting. An understanding of them gives insight into why some organizations truly need formal SCM procedures employed throughout the software development lifecycle. Knowing the needs which some of these procedures were created to satisfy, definitely helps in your own appropriate selection of tools and procedures for your project. Below is a potpourri of some of the more interesting characteristics of those large customers:

- The size of project teams can be anywhere from two hundred to four hundred programmers. The lines of code can range from four to six million lines!
- A project's software development lifecycle may span from five to ten years!
- Traceability is essential. It is not uncommon for the person who develops the initial specifications and code to be different from the person who completes it ten years later.
- Productivity means being able to complete an incremental compile (i.e. re-compiling only what is absolutely necessary, not the whole system) in a day rather that having to rebuild the entire system which may take from one to two weeks!
- The larger the project teams, the more likely it is to have a mix of different hardware. This places an even stronger motivation to use and develop software (especially SCM tools) that is portable across different hardware platforms.
- Informal methods and tools do not work for managing parallel development and resolving file access conflicts with projects that have huge programmer pools or operate in different geographic locations.
- Project management and planning require the generation of timely, up-to-date status reports. Projects must base their plans on the status of a multitude of other projects and dependencies that are sometime geographically dispersed. In addition, typical contractor programming requires frequent and in-depth status reports to be delivered to the contracting employer.
- Following standards whenever possible is considered to be a matter of survival.
- "Mission Critical" functionality takes on a whole new light as a traceability requirement when viewed in large military and aerospace applications. SCM in these huge projects require that a logical link of a paragraph of a requirements document be established to the physical source files. The assurance of full and proper testing and linkage of the functionality is of serious consequence. For example, the flip of a switch may literally mean the difference between life and death as software is invoked to fire retrorockets of a re-entering spacecraft. Whereas for commercial applications, the granularity of the level of linking may not need to be as fine nor considered quite as "critical" ; the need for traceability and functional configuration audits is still readily apparent when applied to payroll or personnel systems.

So... is the disparity between the original target user & typical commercial shops the only reason for reluctance in the adoption of SCM?

COMMON OBJECTIONS FOR NOT FULLY ADOPTING SCM

Although we have seen that there is a very formal definition of Software Configuration Management and users who rely on it daily, it is often difficult to get the same description from the proverbial "person off the street". In a sense, it is akin to trying to define Computer Aided Software Engineering (CASE). There are numerous different interpretations by many different people, and there are also many different implementations. This lack of clarity to the general commercial market has contributed to many unfounded (albeit not always completely!) "reasons" why SCM is not more fully embraced. Hopefully the following will identify and dispel some of the myths and encourage the adoption of SCM (to the correct degree for your project):

"I Don't have the Software Tools required to practice SCM."

SCM is embodied in a set of processes that one uses to maintain their software. Those processes can be either manual or automated. In the beginning, they were mostly manual. As technology has advanced, they have become more automated. Supporting software tools have improved productivity of those people managing that process. Now many people think that in order to practice SCM, they must have this large, all-encompassing set of software tools. If they do not have the tools, then they must not be practicing it. This is not true. Anyone who is managing a software project is practicing SCM even if it is with NO software tools. **By managing a software project, SCM is inherently practiced through processes even without the help that software tools can provide.**

"Access to the Library of Congress is Necessary to Practice SCM."

Hopefully, this humorous exaggeration helps make the point that people often feel overwhelmed by the amount of detail and documentation surrounding SCM. Voluminous amounts of data & huge government documents have been written that cover in minute detail exactly what needs to happen, when it should happen, what forms are required to be completed, who must approve what, what *t's* must be crossed and what *i's* must be dotted. That level of detail is part of the problem --- it is just too much to get your arms around in order to understand. **One can succeed with SCM by scoping their project's practice of formal SCM procedures to the areas where the most control is desired.**

"SCM encompasses all Software Disciplines."

"SCM includes EVERYTHING" is a common refrain one hears when asking people about the breadth of SCM. SCM is often confused with many of the tools that surround it and are many times integrated with it. Take for example the topic of testing. Testing is key to any successful software project and indeed, testing requirements are called for in the practice of SCM. However, SCM focuses more on the "up-front" specification of the requirements that must be met and the subsequent examination of the results of those tests. For example, a requirement may be set that calls for a path flow coverage of 85% of the software code. However, the tools and methods to achieve that goal may not be an SCM issue. (Conversely, in some very formally run projects, the actual measurement tool may be called out as a requirement. It is purely a matter of degree of how much SCM regimentation is imposed and desired.) The main SCM thrust is the specification and the end results of the requirement; not the testing. While testing was singled out in this example, many other software disciplines interact with SCM, but are not actually a core part of it. **Whereas SCM does span the entire software lifecycle, it does not require every software tool and process known to man.**

"The Practice of SCM is Going to Slow the Project Down."

It can take more time to do a job right. The long term benefits such as *not* losing files, meeting quality standards, and having the product work to specification become productivity gains when an additional round of enhancements and fixes are not required. In addition, faster incremental compiles (next section) and pulling status reports together are facilitated.

One of the keys here is summed up in the adage, "Everything in moderation." Frankly, too much SCM in a project may indeed be a waste of time and energy. A small project with resource requirements of one programmer and 50 files may not be the wisest choice for adoption of strict SCM rules. Still there are basic SCM principles that should be followed to successfully complete the project. (This will happen even if people do not know they are doing so.) For every example where SCM might slow a project down there is a counter example of how it can speed a project up. For example, team members may have to go through some extra steps in order to adhere to their standard SCM processes. However when adding or replacing a programmer on the team, those same processes will facilitate a faster learning curve and productivity of that programmer on the team. **The time invested in the practice of an appropriate level of SCM when evaluated over a product's full lifecycle is paid back in quality of the software and productivity of the programmers.**

"I Can't Afford A Separate Configuration Manager."

Generally speaking very large SCM projects do require a separate SCM manager. However, in a smaller project it is worth while for a single person to share dual responsibilities that include SCM management. For example, even relatively small projects frequently have a Project Librarian. That same person could additionally manage many of the SCM responsibilities that have been scoped down to fit the smaller project. Another option is to have the Project Manager or Leader fill that role. **A full-time Configuration Manager is not necessary for projects that have scoped-down SCM requirements.**

"SCM is only for Contract-Driven Organizations."

Much of SCM is a result of the need for managing the development of software by contractual agreement. Oftentimes a baseline is established as a point in time where status of the progress can be measured. At that time, the schedule, the actual deliverables, the quality of what has been produced, etc. are reviewed. **Many of the SCM principles that contractors and their employers share in the successful development and review of their projects are directly applicable by "in-house" programming teams and their management.**

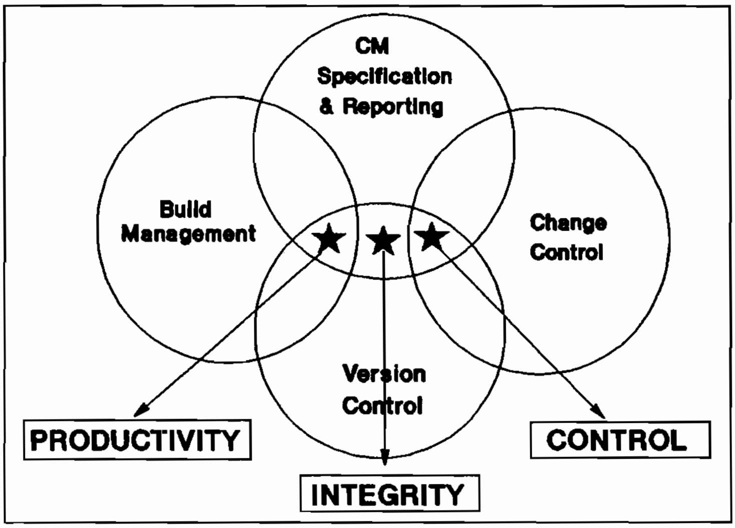
"Only Huge Military & Aerospace Industry Projects use SCM, not ME!"

SCM has traditionally been aimed at the military-aerospace industry. Their development products are huge and the need for SCM was born out of necessity in developing those programs. Further, one does not find a lot of 2 year business schools teaching Software Configuration Management for the COBOL programmer! The point of this paper is to dispel the myth that SCM is only for large sized projects. **The key to beneficial SCM practices in a small to medium-sized project is selecting only those that are needed and to the proper degree.**

So... is there another way to define SCM that the commercial user can better identify with?

AN ALTERNATIVE SCM MODEL FOR MEDIUM-SIZED COMMERCIAL CUSTOMERS

It should be clear now that the primary customer needs of the huge projects found in the military and aerospace industry significantly contrast with those of small to medium-sized commercial customers. After examining the needs of many commercial customers, it was not surprising to find that all commercial customers demanded integrity of their software too. In addition, a large number of those customers placed a definite emphasis on programmer productivity. Other customers placed emphasis on in-depth control of the entire process. In the Software & Methods Lab of Hewlett-Packard, this led to the development of at least one alternative, practical, SCM model based on tools and the needs of the commercial customer:



An Alternative Commercial SCM Model

For each one of the above four classes of tools, there follows:

- **A brief DEFINITION,**
- **A description of "everyday" PROBLEMS found in the commercial market which it helps solve,**
- **A short outline of some desirable FEATURES to look for when addressing it. ***

Following the description of the four tool classes, the synergistic benefits of Productivity, Integrity, and Control as seen at the various intersections in the figure above will be explained.

*This is in no way intended to be a complete list of all features for a tool class.

CM SPECIFICATION & REPORTING

This is similar to Configuration Identification which has already been defined in the traditional SCM model. It is the ability to determine *what is* and *what is not* a part of the software product. This allows the software to be treated as a logical and physical unit. Links between components are recorded. Through those links, the build dependencies to create a system, or any part of the system, is fully described. Also, the customer requirement document specifications are linked with the user manuals as well as the actual code modules that satisfy those initial specifications. The capability is provided to access and fully utilize this information in actual development and reporting of status information for project management.

PROBLEMS

- **Determining how one release is DIFFERENT from the previous one**
  When a new product or application is released, people want to know how it is different from the previous release. They want to know information such as: what bugs have been fixed, what are the new features (if any), what new dependencies have been introduced such as special run-time libraries, what documentation is provided, etc.

- **Determining the correct files & versions to reconstruct a release in order to duplicate and fix a bug**
  When a bug is reported, it is necessary to first duplicate the bug. This requires recreating the environment that the defect was reported against by retrieving the correct release of the program file, message catalog, help catalog, data base, etc. that make up the product or application (not to mention the same Operating System release). After the bug is duplicated, the proper versions of the source files must be retrieved. After modification, the compile and link take place; however, the correct versions of these *transformers* must also be taken into consideration. Finally, regression tests (i.e. tests to determine if any new problem was introduced by the changes made) should be run and of course the correct version of those test scripts is warranted. If an *older* release is being fixed, then the current test scripts may contain tests for features and bugs that did not exist in the *older* release. Then of course, there is the issue of the proper release of the compile time libraries and the run time libraries to ensure identical build and test environments.

- **Determining where a variable or string is used**
  Whether fixing a bug or adding a new feature, it is often necessary to determine everyplace that a variable or string is referenced. This includes both the code (source files, message catalogs, forms files, etc.) and the documentation (external specifications, training manual, user's guide, etc.). If a change is going to be made in the software, it is necessary for the programmer to ensure that the change will not have adverse side effects elsewhere in the code. Before a change is approved, an assessment of the impact on the code and documentation should be made. Many times, the greater investment for making a change lies in the realm of documentation and not the code! Based on this data, a simple change to the software code may be disapproved because of the additional work required in other areas.

FEATURES

- A friendly user interface to define and update configurations, sub-configurations, and their versions. The implementation of this may be based on a data base.

- A configuration load utility to aid in migrating existing applications. The luxury of starting a brand new project with a brand new tool is an infrequent one. A utility to help load an existing program with its compile jobs and dependencies is a serious consideration.

- Administrative capability and security definition.

- A reporting interface and structure for traceability, auditability, and change history reporting. Reports on the configuration and the status of it is critical to planning & assessing progress.

- A Software Bill-of-Materials for an application. This lists all the items and their versions that make up an application including source files, message catalogs, forms files, documentation, compilers, libraries, operating system, etc. It should be available for the generation of reports and for programmatic access from other tools. Those tools might include project management tools, version control systems, release management systems, etc.

- The structure for "where-used" cross-reference capability. This is important enough to list independently although the fundamental requirement for accessibility to this type of data has already been stated.

BUILD MANAGEMENT

Build Management provides the capability to create program files from source code (compile, link, etc.) without rebuilding unchanged components, thereby saving time. This ensures the correct versions of the source are integrated into the product, while maintaining programmer productivity.

PROBLEMS

- **Wasting time in recompiling EVERYTHING to ensure all changes are included**
  Rather than risk having a change missed during development, programmers will often opt for recompiling everything. Unaffected USLs, RLs, SLs, etc. are all rebuilt. When a subprogram compile is done instead, other problems of manually tracking what has changed and exactly what files are necessary for a compile to succeed occur.

- **Rebuilding an entire system for a tiny but significant change while a deadline is eminent**
  It seems inevitable that a programmer will need to make one more small but significant change in a release after it has been built but not yet distributed. If the full build is done then issues sometimes crop up such as: missing files, the right file but the wrong version, unapproved files in the official source file group, etc.

- **Building a product with the right files and right versions**
  Maintaining software with more than one release inevitably means that compiling with the "right" files is only half the problem. The other half is compiling with the correct versions of those files.

- **Keeping track of the destination targets for compiles and links**
  This might seem fairly straight-forward but there are situations where it becomes more complicated. Consider the example where a person makes a change to a source file. They search the application's object libraries to see where it should be compiled into and find the object module name in two different libraries. They then recompile the source into those two libraries. It is only later discovered that the object modules coincidentally had the SAME name but in fact, were originally compiled from DIFFERENT sources for DIFFERENT purposes.

FEATURES

- Automatic builds with a single command. This is basically akin to doing a BUILD ALL. A configuration or sub-configuration file is requested to be built and the system does so by the user typing a single command. The command invokes a build process that may include the execution of compiles, links, tests, cross-reference utilities, or any of a variety of user specified operations.

- Incremental builds. The ability to only rebuild what is necessary based on what has changed and its dependencies is a major factor in gaining productivity. Many build management systems depend on the "date modified" file attribute in determining this.

- A Configuration/Version Control Interface. The capability to maintain different versions of configurations and sub-configurations requires the build management system to interface and "understand" the version control system.

- A Dependency Tree Builder. A "dependency tree builder" essentially takes an existing system and aids in the construction and maintenance of the changing dependencies in it. This may be through a tool that is invoked manually to create and maintain a dependency file, or automatically whenever a build is executed.

VERSION CONTROL

Version Control includes the processes and tools used to identify the version of a component or product, the details of the changes between the versions, the reasons for the changes, and the individual who made the changes. In addition it provides: the capability to store and recreate versions of a program and its associated files, controls to prevent users from changing the same files at the same time and thereby writing over each other's changes, the capability to support parallel development, and security features to control who may access files.

PROBLEMS

- **Two programmers writing to the same file & destroying each other's changes**
  This is a classic version control problem that begins with a programmer copying a file to their own group from the official source group where the current, stable source files reside. This is followed by a second programmer copying that same file to their group, making changes, and then copying it back to the official source group. Later, the first programmer copies their file back into the official source group on top of the other programmer's. This destroys the second programmer's changes.

- **Retrieving the correct release of a program and its source files**
  A very common problem is the need to retrieve an older program file (to reproduce a bug reported by a user), or recreate the correct version of the source files (in order to perform a patch).

- **Identifying "who changed what" while debugging a failed team compile**
  A common team development problem occurs when a programmer feels they do not need to follow the project's rules for making changes. They simply modify files from directly inside the official source group without documenting their changes. All the other team members follow project procedures including copying files into their own group and back into the official source group. A week or so later, an integrated compile is done and tests are run that subsequently fail. At this point, no audit trail exists to identify what was changed, who made the changes, when the changes were made, and for what purpose. The changes made in the official group were done in earnest by non-malicious programmers in the name of "productivity"! In reality, they sacrificed the team's productivity for their own.

- **Integrating changes by programming teams working on separate copies of the same source**
  This situation of managing parallel development can be time consuming in ensuring that all changes made by one team are incorporated into their source and vice versa.

FEATURES

- Check in and Check out of files. Upon check out, a version control system will copy out a particular revision of a file and apply what is know as a "lock". This lock prevents another programmer from checking out that same revision. An "escape valve" is usually provided so a user with the proper security can override the lock if a *real world* problem makes it necessary. (For example, if a *hot* bug fix needs to be made while another programmer is modifying the file, or if a programmer leaves the company.) A Check In permanently logs the file into the version control system with the new changes added.

- The capability to reproduce versions of files and configurations. This includes the ability to reference a configuration by name such that all files and their correct versions are copied out of the version control system. Manually keeping track of what versions of the files of a configuration make up a particular release is too difficult. Naturally, the security to control who can perform these types of operation are a built in part of the system. The functionality to version the configuration definition itself is also important as it may undergo significant changes over time.

- Logging of author, date, revision, and purpose of changes. When changes are made to a configuration or file, the system should log who made the change, when the change was made, what revision number or name was assigned to it, and the purpose of the change. The first three items are generally done automatically by the system. The purpose is usually supplied by the programmer.

- Differencing of versions of files or configurations. The capability to *difference* or list the line-by-line changes made between two revisions is inherent to most version control systems.

- Branching & Merging. The ability to create a branch & then merge that branch and its changes back into a single revision is fundamental to supporting parallel development. The merging system will interleave the changes properly whenever possible. Merging, when coupled with intelligent software development methodologies, solves the problem of fixing bugs on an old version and having those fixes propogated forward while simultaneously implementing major enhancements.

- Reporting. Audit trails to report what was logged will aid in determining who has changed what or who has performed what operation on a configuration is another key feature.

CHANGE CONTROL

Change control remains the same as in the standard SCM model. A sizable portion of commercial customers are satisfied with their current processes and tools to handle change. Some do require more sophisticated change control tools though.

PROBLEMS

- **Finding out the status of a bug report**
  A common situation occurs when a person submits a bug report and then a week later, is unable to find out what has happened with it. The submitter can not find out if the bug has been received, if it was approved or disapproved (and why), if there is a "workaround" for it, etc.

- **Communicating software interface changes to other departments**
  This occurs when software interfaces are changed and the documentation department, marketing department, or some other essential functional area is left uninformed. This can be costly to fix if the lack of communication is not discovered until just before release. It is even more costly if the product or application has already been released! This is also a common problem between programming teams working with shared, common code.

- **Assessing whether an application or product is ready for release**
  Before a release is fully approved and distributed, a certain amount of information is warranted. This includes: test coverage of the software, test results, programming standards followed, conformance to original specifications, authorizations from different departments, etc.

- **Determining why a project is over budget and behind schedule**
  Changing requirements and unknown dependencies occur during the course of a project. When projects do not meet budget and schedule constraints, management wants to know why!

- **Standardizing on a procedure for making changes**
  A very common problem is that programmers and users do not know what the proper procedure is to make a change. The results are that the needed change is not included or the change is included "illegally" with sometimes costly results.

FEATURES

- A Defect Tracking System. A defect tracking system that assigns a unique number or name to a bug or enhancement request is a good start. This system should provide a documented trail of what has happened to the request and what is planned to happen to it.

- A Baseline Change Control & Approval System. In order for an established baseline to be changed, there needs to be a formal process in place. An actual system provides a common structure for people to standardize on although the actual approvals must still be done by the *equivalent* of a Configuration Control Board (CCB).

- A Phase Promotion and Software Release Management System. For software to be formally approved and officially moved to other systems, a software release management system is often employed.

- Support for the management of projects and the Project Life Cycle. Whatever system is adopted should support a standard software life cycle methodology and actually aid in the management of the projects, not get in the way. Reports should be easily obtainable.

THE BENEFITS: INTEGRITY, PRODUCTIVITY, AND CONTROL

The synergy generated from the intersection of the different tool classes results in further benefits of integrity, productivity, and control (see previous figure).

Integrity is the most important element of SCM. In order to consistently and reliably get a working final product, developers must be confident that all components of the software product are identified (where and what), and they must be able to determine how those versions of the components fit together. For that reason, the intersection of **CM Specification & Reporting** with **Version Control** yields **Integrity**. Without product integrity, there is no point in trying to increase productivity (all gains would be lost while reworking the final product); and without product integrity, there is no basis from which to exert control. Conversely, increased productivity and control add to the integrity through an automated, reproducible build process and improved control over the content of the product.

Productivity is increased by adding **Build Management** to the combination of **CM Specification & Reporting** and **Version Control**. With Build Management, builds can be done automatically, reliably, and efficiently! The build dependencies are known, and build time is reduced by reconstructing only the units which have changed. Since all components of the software product are known, along with the versions of each component belonging to any specific product release, any release can be built/recreated with minimal programmer interaction. This allows programmers to spend time on content rather than the administrative details of construction.

Control over the exact content of a product, during development and at release, is achieved by adding **Change Control** to the combination of **CM Specification & Reporting** and **Version Control**. Several ways are provided that make it possible to tightly verify that all intended changes are incorporated into the product: 1) authorizations, change request definitions, and checkpoints, via **Change Control**; 2) version change history, and 3) identification of all components (including the specification documents) of a product, via **CM Specification & Reporting**. All of these also ensures that no one slips in any unauthorized changes.

A WORD ABOUT SCM PROCESSES

Although this paper is focused primarily on SCM tools, it should be reiterated that the original foundation of SCM is embodied in processes. Everything mentioned thus far can be implemented manually, without using any automated tools. Tools can improve productivity and reliability; but without the appropriate processes, even the best SCM tools will fail. Therefore, when improving the configuration management of a project, the processes employed should be carefully studied. Once the processes have been analyzed and decided on, appropriate tools should be chosen to facilitate the processes that are most important to the project and that result in the greatest gain in productivity and integrity.

So... besides the above non-exhaustive list of basic tool categories, what other exciting features are there?

ADVANCED FUNCTIONALITY

This paper's main focus was on basic practical SCM and functionality that would be useful in the majority of the commercial developer's world. What follows is a potpourri of some of the more interesting and advanced "wish-list" SCM functionality. This includes:

- a system that automatically notifies (perhaps via electronic mail) of changes to the baseline. This would provide automated communication across functional areas (marketing, documentation, lab, etc.) of change approvals that might effect other departments and other significant baseline milestones achieved for planning purposes.

- an automated distributed build manager that selects the machine in the network that is best suited to do the job. The build could be fully done on one machine or incrementally distributed. Machine characteristics such as CPU speed, current utilization, memory and disk space available, etc. may all be taken into account during the selection process.

- automatic builds based on events in the SCM system. For example, when a Check In of a configuration or sub-configuration occurs in the version control system, a corresponding build would automatically be invoked. In addition, the build system might have its "dependency tree" updated as part of the event.

- versioning of variants such that multiple target machines are supported. This would mean that software applications with only slight target machine-oriented differences in them could transparently be versioned and automatically built on the right machines.

- transparent access to distributed configurations and files across a network.

- integration of Structured Analysis/Structured Design type tools with configuration identification. This would include a windowed, graphical environment for maintaining the configuration structure and the facilitated capability to zoom in and out of sub-configurations.

On a final note, much of the functionality discussed above is currently available in the technical and other markets.

SUMMARY

Software Configuration Management (SCM) has been formally defined through a description of the four activities of Configuration Identification, Change Control, Audits, and Status Reporting. Nonetheless, SCM is a topic that is surrounded with misconception.

It is often erroneously thought of as a discipline that is used and is applicable to only very large software projects. This is understandable though, as the formal definition of SCM was born out of the needs and problems of huge projects often associated with the military or aerospace industry. This includes projects with hundreds of programmers, millions of lines of code, project life cycles that span five to ten years, and "mission critical" functionality and testing requirements that could literally spell the difference between life and death. Needless to say, most commercial programming shops do not identify themselves with projects of that scale!

Compounding the problem of disassociation with the original target customer are other common myths. Feelings that the adoption of SCM requires a great number of software tools, a large overhead in detailed paperwork or bureaucracy, a decrease in programmer productivity, an investment in a separate Software Configuration Manager, etc. are typical.

In fact, SCM is a common, everyday practice in projects of all sizes even though in some cases, the programming staff may be unaware that they are doing so. A proposal is made that simplifies SCM into a **practical** model for the "normal" sized commercial shop. This model combines Configuration Management Specification & Reporting, Automated Build Management, Version Control, and Change Control to better address the commercial customer's SCM problems and needs. **Configuration Management Specification & Reporting** provides the ability to determine *what is* and *what is not* a part of the software product as well as the capability to report status information for project management purposes. Automated **Build Management** provides the capability to create program files from source code (compile, link, etc.) without rebuilding unchanged components. **Version Control** provides the capability to identify and retrieve the version of a component or product, the details of the changes between the versions, the reasons for the changes, and the individual who made the change. **Change Control** provides the framework for approval or disapproval of proposed changes to an established standard specification or baseline of a project.

Finally, by adopting the **appropriate level** of tools and processes targeted for the unique needs and methodology of the typical commercial software shop, SCM can offer greater **integrity** to the software system, increased **productivity** for its developers, and desirable **control** over the software processes.

BIBLIOGRAPHY

Babich, Wayne A., *Software Configuration Management*, Reading, Massachusetts: Addison-Wesley Publishing Company, 1986

Bersoff, Edward H. & Henderson, Vilas D. & Siegel, Stanley G., *Software Configuration Management*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1980

Buckle, J. K., *Software Configuration Management*, Houndmills, Basingstoke, Hampshire: Macmillan Education Ltd, 1982

Davey, Mark & Padula, Alan, "Software Configuration Management: An Introduction", *SML News*, April 1989, pp. 1-3

DOD-STD-2167A Military Standard Defense System Software Development, Washington D.C.: Department of Defense, February, 1988

DOD-STD-2168 Military Standard Defense System Software Quality Program, Washington D.C.: Department of Defense

Konstantinow, George, "Standards for Software Configuration Management", *Computer Society Press Reprint*, IEEE Computer Society Press, 1986, pp. 101-105

Leblang, David B. & Chase, Robert P. Jr., & McLean, Gordon D. Jr., "The DOMAIN Software Engineering Environment for Large Scale Software Development Efforts", *Computer Society Press Reprint*, IEEE Computer Society Press, 1985, pp. 260-280

Singleton, Margaret E., *Automating Code and Documentation Management (CDM)*, Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1987

NO TIME, NO MONEY AND THE SYSTEM NEEDS A COMPLETE REWRITE

SALVAGING A MORIBUND SYSTEM WITH SPEEDWARE

John S. Fusek
& Serena R. Fusek
Virginia International Terminals
P.O. Box 1387
Norfolk, Virginia 23501
(804) 440-7023

INTRODUCTION

Virginia International Terminals is the operating company for the Virginia Port Authority. VIT runs three ocean-going, multi-shipline terminals in southern Virginia and is in a race with the port of Baltimore to be the top terminal on the Chesapeake Bay.

VIT's cargo tracking system is computerized. The Virginia ports have grown rapidly in the last five years and so has VIT's computer system, which is run on an HP 3000. (see: "Developing A System to Optimize and Control Containerized Cargo Movements" in the 1989 INTEREX PROCEEDINGS).

Among our operations systems, most of which were written in the last three years, we have one ancient system that we still use every day. Documentation on this system is almost nonexistent but as far as can be determined, the design was started in 1972 and implemented in COBOL with TOTAL and CICS on an IBM 370-115. In April 1977, the system was converted over to the HP 3000 in COBOL with IMAGE and DEL. This was done to decrease expense and increase reliability. After that the system remained unaltered until fall, 1989.

Because this is our Container Control System, which handles 90% of our business, Its inherited inflexibility and inefficiency had become a real problem. The transportation industry was one of the last to automate on a large scale. Today, however, computerization has taken hold and the demand for data has increased 100 fold, increasing our headaches.

Actual day to day port operations had not changed and the old system remained functional but it was not up to the demands newer additional systems were putting on it.

In 1986, the VPA decided on Speedware as the Fourth Generation Language it would use. VIT followed suit. Many new sub-systems, systems and applications have been developed in Speedware. We in VIT's DP shop have grown addicted to the ease of developing and modifying applications in Speedware. Maintaining the Container Control System in COBOL grew distasteful and was too slow to easily keep up with the increasing work load.

Our DP shop is small and our workload large. Therefore, we asked an outside consulting firm to give an estimate to rewrite the CCS. After several months, they returned with a 7 digit amount and a two year time frame. This was unacceptable. Our budget would not extend that far and we needed the system updated now, not two years down the road, if we expected to stay competitive.

Rather than rewrite the system from scratch, VIT's programmers and analysts believed they could convert it in-house to Speedware with considerable savings in time and money. We gave ourselves three months to complete the conversion. This paper documents the conversion's course.

Problems in the Container Control System: Screens and Modules

The Container Control System was written mostly in COBOL68 with many subroutines in SPL. The main screen handler was DEL with extensive modification in SPL. There were many small routines in an RL. We were not sure what language the RL routines were written in.

The main logic of the system revolved around the main areas of container movement.

We had a screen and a module for container arrivals in the gate (via truck or rail), and one for ship arrivals. The same held true for departures except in rail where a new subsystem allowed us to depart containers by railcar number. There were screens for container movements, entering customer and delivery order information and a screen to allow a full record modify. Other screens allowed for the loading and unloading of containers from trucks or ships and moving them for customs inspections.

As the years passed, these screens acquired their own lives. A lot of operator logic grew around each screen, making it difficult for new personnel to use without a long training period. Documentation did not cover all the details the new operator needed to know.

We will use gate arrivals as an example to illustrate the different transactions that were handled with a single screen:

1. The most common transaction is a loaded container being exported on a vessel. Most of the fields on this screen were used for this transaction.

2. An empty container being sent to the terminal for storage and dispatch to a customer for loading required the entry clerk to tab over many fields to enter the information.

3. Import containers are brought in by truck or rail for several reasons. The two most common reasons are to be put on another vessel going to the correct U.S. port or for customs clearances to be completed. Because the container comes in the gate, the old CCS required it to be entered as a gate transaction. These transactions required different data to be placed in fields than is given for exports. Then operators had to go to another screen to complete keying in data that had no equivalent on the gate arrival screen.

```
                         ADD CONTAINER GATE                             ADCG <2.0>
CONTAINER   [              ]             VESSEL/VOY [        ] [    ] SHIPL [  ]
ARRIVAL DT  [           ]                CARRIER    [    ]  TRK/RL [          ]
PRO NO      [              ]             BOOKING NO [          ]
COMM DESC   [   -                      ] COMM TYPE  [   ]   HAZARD [ ]
SCALE WGHT  [       ] [ ]                CONT TARE  [       ] [ ]
TRACT TARE  [       ] [ ]  CHASS TARE [       ] [ ]  MAX GROSS [       ] [ ]
CONT SZ/TY  [      ] CHAS# [             ] SZ[ ]  TYPE[   ] STATUS [   ] RAIL [     ]
SEAL NO     [             ]  PORT [     ] SHIPPER    [                   ]
SHIPPER AD  [                    ]       CONSIGNEE  [                   ]
CONS ADDR   [                    ]       LOCATION   [         ]
ST/DIR/OKM  [  ] [ ] [ ]                 BROKER     [    ]  S/S DT [        ]
REMARKS     [                                        ]

  CHASSIS TYPES: RG IF BLANK          CHASSIS STATUS: LC IF BLANK AND NOT EC/DC ST
    RG - REGULAR   GO - GOOSE NECK      LC - LOADED     EC - EMPTY
    RF - REEFER    FB - FLATBED         DC - DAMAGED    RC - REPAIR
    EP - 8 PIN     CC - CARRIER'S       BC*- BUNDLED (REQUIRES 2 CHASS-UP TO 5)
    MC*- MARRIED (REQUIRES 2 CHASS-NO MORE-CHAS# IS 1ST,CONSIGNEE IS 2ND)
  * ADDITIONAL CHASSIS NOS.-SHIPPER IS 1ST,SHIPPER AD IS 2ND,CONSIGNEE IS 3RD
    AND CONS ADDR IS 4TH-----IN THAT ORDER.


1   f1   2    f2   3    f3   4    f4      2   13 5    f5    6    f6    7    f7    8    f8
```

Old Gate Arrival Screen

System needs a Complete Rewrite

4. Empty chassis come in the gate. These are frames with wheels and lights that carry the container over the road. Chassis can come in singularly, in matched pairs or bundles of four or more.

As can be seen, a single screen had many functions for which most of the fields were mislabeled.

After the data was entered, the problem did not improve.

The system was designed one module for one screen, so there were more than a dozen modules to handle all the screens. In many of these modules billing transactions were generated. Billing records were written in SPL subroutines, with much logic in the COBOL modules deciding whether to call these subroutines. As shown above, many different transactions could be generated for a given module. Several of these were generated in more than one module. Also, VIT is a customer-driven company. Our salesmen are happy to arrange special billing, which the old CCS handled in the data entry modules. Finally, billing changes on a yearly cycle when tariffs are negotiated port wide.

This means that each month there is some change to the billing process that used to stretch across several modules. Because these modules had been in existence a long time, they had undergone a lot of modification by quite a few different people. For many years VIT's DP shop was very small with a large workload, so documentation was given its usual treatment: that is, never updated.

This was the basic on-line data entry system. When we got into the nightly batch processing, things just got worse. But it worked; we were moving and tracking containers with it. Modifying it for even the simplest change, however, took weeks.

STRATEGY

We developed a simple strategy for the conversion project that would allow us to achieve our objectives rapidly. We would develop a speedware screen for each transaction type that came through the system. The screens would be grouped by function into menus called by a main menu. All updates to the main data base would be written to a separate data base, with the second data base mirroring the record's condition at the time of transaction. The mirror data base would generate the billing and history.

We designed this approach to give us some enhancements while preserving the basic functionality of the system. We did not want to change the definition of the main data base. If we retained that definition, we would not have to worry about a data conversion effort. Also if we developed a major problem, we could return to the old CCS easily.

Some enhancements we decided we could add with little impact on the system would create more comprehensive editing and generation of history information. The old system did very little editing. When the system was written in the early 1970's, 5,000 containers a month passed through VIT and a clerk scanning the reports could find any errors. Today the volume is closer to 40,00 containers/transactions a month. Additional systems had been developed around the CCS. Eyeball scanning no longer provided sufficient editing. The data had to be correct to move containers rapidly. By using speedware to generate tables for look-ups, we would be able to rapidly add editing to many fields.

Another enhancement that the CCS needed in the conversion was the development of transaction history. As was shown, the generation of billing was scattered across many modules. The conversion would centralize the billing logic. It would also allow the addition of audit trails and the start of a extensive management reporting system.

Basic tactics were to isolate a transaction and design a screen around for it using the input documents. The screen would then update the main data base and write out one or more billing and/or history records. A separate billing program was then written to handle all the billable transactions. After billing was complete all records were then transferred to history for reporting and inquiry.

```
Speedware               Container Arrival System 05-11-90                05/12/90
                   Add Empty        - Gate              (ADCG) (A)

  1 CARRIER     : BLGR  BLUE_&_GRAY_TRANSPORTATION_CO,
  2 CONTAINER #: ADCG0001010     DATE: 05/12/90 08:00        chassis information
  5 SZ/HGT/TYPE: 486BX BOX_________________                  ------------------
  6 SHIPLINE    : EV    EVERGREEN_MARINE_CORP_(NY)_LTD     7 CHAS #1: ADCZ0001010_
                                                           8 TYPE   : RG
                                                           9 SIZE   : 40
                                                          10 TARE   : _3400
                                                          11 CHAS #2: ____________
 14 COMM TYPE   : 001   GENERAL_CARGO_________________
 15 HAZARD CODE: __    ______________________________
 16 CONT. TARE : __4200
 17 LOCATION    : T3____
 18 SLOT        : 2345
    HOW ARRIVED: G
 20 CONT.STATUS: EM
 21 REMARKS     : ________________________________________


 _____________________________________________________________________________
 21 REMARKS     :                                         .
 ADD LOAD ADD LOAD ADD        PRINT     23   17    CONT      HELP     ACCEPT   STOP
 EXPORT   IMPORT   CHASSIS    SCREEN    C          LOOKUP    FIELD    RECORD   REJECT
```

New Gate Arrival Empty Screen

System needs a Complete Rewrite 3091-7

IMPLEMENTING THE CONVERSION AND PROBLEMS ENCOUNTERED

During implementation of the conversion project, we were pleasantly surprised. There were no major technical problems.

The problems that delayed the project were all either personnel or organizational problems. Family problems forced a crucial individual to leave VIT as the project was nearing the testing stage. An especially virulent flu swept through the office, then returned just as personnel were starting feel well again. It also seemed that there were more than the usual number of personal emergencies during this period.

The worst problem we encountered was the unexpected re-organization that shook VIT's Container Operating Department. Key personnel left Operations and new people were rushed into the empty positions. Three new supervisors entered the department and two switched positions. We spent a month, for which we had not planned, training these people for their new jobs.

Technical problems were small and easily solved. VIT has been using Speedware since 1986 and we have acquired considerable proficiency in it.

Most of the complicated SPL sub-routines were found to be part of the special screen-handling procedures set up to duplicate CICS screens from the IBM 370: Speedware did away with all this.

Speedware has evolved so that we could choose between several options whenever we reached a sticky point in the conversion. In a few cases, however, we dropped into COBOLII to write some sub-routines. The COBOLII sub-routines were used mainly for speed and will be rewritten later.

COBOL as a SPL Substitute

As this project developed, we found we could replace SPL routines with ones written in COBOLII. For several earlier projects we had developed bit manipulation routines for COBOL. By combining these with the extended CALL command, we were able to duplicate functions of all the SPL routines that were not eliminated. We had only one programmer on our staff with any SPL experience and that was minimal. Therefore, although using COBOLII decreased machine efficiency, it increased our ability to maintain the system. As we are migrating to precision architecture, replacing SPL with COBOL will prove a plus.

TESTING

Program and quality assurance testing was carried out by our MIS coordinators. These people joined our project when VIT's Assistant General Manager saw we would need more application knowledge before we started. The individuals who worked with us had been chief clerks in the operating divisions affected by the conversion.

The coordinators went through the basic Speedware course with the programmers, so were able to do the on-line documentation themselves. This documentation was done from the notes they took while certifying that the system was ready. It included help screens designed to guide operations clerks through any procedures that might seem difficult.

RESULTS

After the conversion was completed, we realized we should have implemented it a lot sooner. By converting the out-of-date Container Control System to Speedware, we were able to clear up a lot of outstanding work requests. Some of these requests had waited years for fulfillment. The time we would have needed to to satisfy them was at least equal to the amount we used for the conversion.

The new features we added to the system gave us powerful tools to generate history and productivity reports.

In addition, by converting a system with which we were familiar, we preserved all the experience we had gained working with it for fifteen years. If we had redesigned the system, it would have been necessary to learn the new system from the beginning. Commerce through VIT is too heavy to accommodate the kind of snags and snafus a brand new system can create during that learning period.

The conversion period covered five months, instead of the projected three. The need for the increased time was caused more by the personnel and organizational problems mentioned, rather than any technical glitches. The amount of work time put into the project was the projected three months.

Outside contractors had estimated a price of 2 million dollars and 18 to 20 months to redesign the CCS. We finished the conversion in 5 months at about a cost of $100,000. Speedware will carry the Container Control System, originally designed in the early 1970's, into the 21st century.

CONCLUSION

Conversion of the Container Control System to Speedware fulfilled all our expectations. The new, speedier CCS will allow VIT to manage a mushrooming transportation industry into the twenty-first century.

For what kind of system can you expect similar results?

Any working COBOL system can benefit by being converted to Speedware. The key word here is "working". Speedware cannot fix dysfunctional or marginally functional systems, but it will boost a worn out system. The CCS was still fulfilling its tasks but had lost its efficiency. To get the maximum return, the system you want to convert should have been operational for two or more years and have a large maintenance backlog.

How long a conversion will take depends on your system and the people working on it. Those people should have both a strong working knowledge of your operation and proficiency in 4GL languages. If your crew meets this criteria, you should be able to convert a system in approximately the amount of time the analysis phase of a traditional 3GL project would take.

OMNIDEX/IMSAM AS A DATABASE DESIGN TOOL

Katherine G. Newcombe
Systems Consultant
58 Old Kings Highway
Hampton, CT 06247-9712
Phone: (203) 455-9864
FAX: (203) 455-0490

OMNIDEX and IMSAM, products of Dynamic Information Systems Corporation, under the umbrella of the OMNIDEX Information Management System, give the data base designer great flexibility in the organization of an IMAGE/3000 data base.

OMNIDEX keyword retrieval allows the user the ability to qualify or disqualify data set entries based on criteria specified by the user. This can be especially useful in cases where a count of qualified entries may be all the information a user may need to reclaim from a data base (for instance, how many mailings would be needed to reach all manufacturing companies in the state of Connecticut with at least 25 employees). If the user desires the actual data to be retrieved, OMNIDEX intrinsics also allow for speedy access of the qualified entries' search item values, which can then be used to retrieve the entries via IMAGE calls.

IMSAM keyed sequential access provides the user with the ability to access data by any type or number of keys. This data can be sorted in user-defined sequence for ease of retrieval and reporting. Any data set can be keyed not only by its IMAGE search item, but by almost any other item, combination of items, or subsets of items. Retrieval can be based upon an exact match of a full or partial key value, or on sorted ascending or descending sequence, starting with the full or partial key value specified.

Based upon these abilities, the data base designer can organize a data base by functionality rather than by the number or types of keys needed to access the data. The data base have its preliminary design based far less on the consideration of the ease of data retrieval because of the accessing capabilities of IMSAM. Logical master data sets can be made into physical master sets, which can still be accessed by more than one key. Logical detail data sets can be made into physical master sets, which can still be accessed by more than one key (the IMAGE search item). Logical detail data sets can be made into physical detail sets, while retaining direct read capability without consideration for record addresses. The data base can be made to work specifically for the user's application by creative use of the tools provided by the OMNIDEX IMS.

There are several advantages to designing a data base with the OMNIDEX IMS, rather than using the OMNIDEX key structures as an overlay on an already existing database. Design can be simplified by incorporating the use of OMNIDEX keys and IMSAM B-tree structures.

- o Direct access capability to detail data sets can be maintained without the use of multiple IMAGE automatic masters and relatively inefficient IMAGE find-and-chained-get retrieval. The use of sorted paths within data sets and external sorts during data retrieval can be virtually eliminated. Instead, the IMSAM sorted-sequential access method can be utilized to retrieve data by any IMSAM key or composite key within a data set or even across data sets.

- o Logical masters can remain physical masters and still allow keyword retrieval by several items within the data set via OMNIDEX intrinsics or access by items other than the IMAGE search item via IMSAM keyed access.

- o The physical organization of the database can be determined before system design begins, because the method by which the data is to be retrieved is much less of a physical database design factor. OMNIDEX and IMSAM keys can be created during the system design phase, as the need arises for the keys as determined by the system flow and design specifications. With an IMAGE only database, physical database structure is almost totally dependent upon system design, which determines the way in which the data must be captured and retrieved.

- o Data sets within a database designed with the OMNIDEX IMS can be contain items specific to a particular function rather than items combined because of data access considerations (i.e., find-and-chained-get retrievals on long chains). Enhancements or modifications to a database designed in this fashion can be simpler, since the only data sets and modules affected are those particular to the functions being modified.

- The use of many complex and advanced features of the OMNIDEX IMS can be avoided by carefully using OMNIDEX and IMSAM tools during the design phase. Individual detail record indexing (DR option), which allows the OMNIDEX intrinsics to qualify individual detail data set entries instead of master data set entries, can be difficult to use if qualification and/or retrieval across data sets is required. Using record specific keyword indexing (RS option) on a detail data set requires several conditions to be met, including a limitation on the number of entries in a chain and the ordering of the detail data set in primary IMAGE path sequence. These tools are no doubt useful if the OMNIDEX IMS umbrella must be placed over an already existing database which can not be modified (as part of third-party vendor packages) or which is difficult to modify (such as converting a detail data set to a master). However, they are not the easiest features to understand or implement, and can be avoided with the proper data base design.

- IMSAM eliminates the need for extra fields required by the limitations of IMAGE in order to retrieve data by a combination of keys. Composite keys or subsets of keys can be created invisibly to IMAGE, but yet increase the retrieval power of the data base.

Following are some suggestions for the use of the OMNIDEX and IMSAM intrinsics in application programs, specifically using COBOL, which I have discovered via a combination of exhaustive review of the OMNIDEX IMS manual and trial-and-error programming.

- Qualification of OMNIDEX ID/SI's must be performed by inclusion of desired keywords before exclusion of ineligible entries; in other words, the AND (,) and OR (+) keyword operators, along with any keywords desired, must be specified before the AND NOT (,-) keyword operator and any keyword(s) of entries to be eliminated.

- "No Parse" keywords or keywords containing blanks must each be delimited by not only the keyword operators but also by open and close parentheses (i.e.,"NEW MEXICO"+"NEW HAMPSHIRE"+"SOUTH DAKOTA").

- OMNIDEX fields in detail data sets which are used together to qualify master entries should be combined into OMNIDEX composite keys; otherwise, the entries disqualified by the use of one field may be included by qualification on another field.

- Access via an IMSAM composite key can be tricky if retrieval using the key is designed to return the desired data in a specific sequence. Using a composite key made up of BALANCE and EFFECTIVE-DATE, one cannot retrieve entries with a BALANCE greater than zero in descending sequence by EFFECTIVE-DATE, since the retrieval would involve a simultaneous "greater than" mode (to retrieve only entries with a non-zero BALANCE) and a "less than" mode (to retrieve the desired entries in descending sequence).

- A check must be done after each next or previous sequential mode (90 or 91) retrieval to be sure that the desired key was returned by IMSAM. The initial IMSAM retrieval, using a directed mode (equal (100), greater than (200), less than (400), etc.), operates similarly to a generic start - it finds the first entry (if one exists) which qualifies; each subsequent retrieval, in "next" or "previous" sequence, returns the next or previous entry in the IMSAM chain, whether or not it still satisfies the initial retrieval request.

- Since access is more likely to be along an IMSAM chain for all types of data processing in a data base designed with the OMNIDEX IMS, not only data retrieval for reporting as in an overlay of the OMNIDEX IMS on an existing database, updating of entries can be more complicated. This is true if selective updating (i.e., updating only certain entries in an IMSAM chain) is being performed on any part of an IMSAM key or composite key against which the entries are being accessed. When an entry is updated, modifying all or part of the IMSAM key by which it was retrieved, its position in the IMSAM chain is changed in the IMSAM B-tree to reflect the updated IMSAM key. Therefore, the next or previous entry which would be retrieved by a subsequent IMSAM call would be the next or previous entry according to the updated position in the IMSAM chain. In order to retrieve the next or previous entry desired when using an initial directed base retrieval mode of 'greater than' (200),'greater than or equal' (300), 'less than' (400), or 'less than or equal' (500), the full IMSAM key value of the updated entry, before modification, must be saved and used along with the original directed retrieval mode to get the next desired entry after the update. In 'equal to' mode (100), the retrieval of the next logical entry in the IMSAM chain is much more complicated. After it has been determined that an entry (and the IMSAM key by which it was retrieved) must be updated, a discrete mode retrieval must be performed to access the IMSAM key value and the IMAGE search item value (if retrieval is being performed on a master) or the relative record address (if retrieval is being performed on a detail) of the next entry desired; these values must be saved. Next, the entry to be updated must be re-retrieved and updated to its new position in the IMSAM chain. Then, using the saved values, an IMSAM retrieval must be performed using the initial directed mode to access the next logical entry in the IMSAM chain.

Even with some of the complicated means of retrieval and update using the OMNIDEX and IMSAM intrinsics, there is no denying the fact that the OMNIDEX IMS offers the database designer tools which greatly enhance the ability to order and retrieve desired data much more rapidly than with an IMAGE data base alone.

ONE COMPANY'S SUCCESSFUL IMPLEMENTATION OF SOFTWARE GUIDELINES

Daniel Marcotte, Manager, Systems and Programming

United Liquors Ltd.
One United Drive
West Bridgewater, MA 02379
(508) 588-2300

United Liquors Ltd. is a privately-owned liquor distributor operating form three sites in Massachusetts. The company, founded in 1933, has grown consistantly from one truck and one salesman to the 100 trucks and 600 employees of today. It was not until 1984 that United Liquors entered the world of data processing. The story of ULL's computer system development was told in an article appropriately titled System Development Disaster in the October 1987 issue of "HP Professional." The ULL Management Information Services department has since overcome its notorious beginnings, and has helped the company reach the $200 million-dollar plateau in gross sales.

The article mentioned above left an indelible impression that programs were written in an "Install or be Killed" type of environment. Pressure from upper management can often affect the deadline of a project causing both testing and documentation to suffer dramatically. This pressure combined with quick fixes to production problems left the ULL system in chaos. Some of the common problems that had to be addressed included missing source code, hard-coded values in many of the production programs and little or no documentation of jobstreams, programs or systems. The problem was addressed from two angles. The first was to ensure that documentation and compliance with established standards was held as a high priority when developing new programs or when maintaining older ones, and the second was to document existing systems for both the technical staff and from a user's point of view. Another problem that plagued the fairly young MIS department was a lack of project management. Some programmers maintained lists of requested projects that they would prioritize themselves, and other programmers used the classic project tracking technique of post-it notes stuck to their terminals, desks and walls. The last in first out method was often used, or, just as commonly, the user that yelled the loudest received very prompt attention.

THE STANDARDS MANUAL

The process of developing a standards manual can be a tedious one. If the standards are dictated from above with an iron fist, they will not be as readily accepted as if they are decided upon by those intended to follow the standards. The ULL standards manual was the result of many discussions and compromises between a group of knowledgeable, hardworking programmers, analysts and managers. The end result was a good guideline of requirements for the ULL development environment. The following is a summarized view of the standards manual so that you can envision building one of your own.

I) Production Accounts

Production accounts are the frontline of the ULL system. They contain all applications released to the user community. These accounts are protected against all testing and development so as to maintain the integrity of the data and programs. Each production account is assigned to a functional analyst who, with their backup, act as librarian, groundskeeper and guard dog for all activity within the account. The ULL system at this time contains five functional areas. They are: Financials, Inventory, Order Entry, Purchasing and Sales. Each of the production accounts have identical structures with groups for source code, object code, job streams, documentation and databases and data files. Vendor accounts are also maintained for all applications purchased by ULL. Any software developed to enhance the vendor's product is stored with the most applicable ULL application.

II) Naming Conventions

Naming conventions can vary greatly from organization to organization. ULL adopted a simple plan that seems to be very effective. Each application has a two-letter mnemonic associated with it (i.e., "PO" = Purchasing). This mnemonic combined with a three-digit number make up the program descriptor part of the name. The remaining three characters describe the type of file using the following conventions.

CO#	- COBOL Object Code	VF	- VPLUS Fast Forms File
CS#	- COBOL Source Code	VS	- VPLUS Source Code
QU#	- QUIZ Use File	SF	- Sort File
RO#	- QUIZ Object Code	KK	- KSAM Key File
RS#	- QUIZ Source Code	KS	- KSAM Data File
PO#	- QTP Object Code	CR	- Console Restartable Job
PS3	- QTP Source Code	RP	- Job that Requires Recovery
SO#	- Quick Object Code	RS	- Recovery Job
SS#	- Quick Source Code	JOB	- Misc. Jobstream
SU#	- Cognos Subfile		

Databases are named under a slightly different convention in that the mnemonic is combined with "DB" for the database, and with "DBSC" for the schema file. UDC's are also named differently following these conventions:

UDC + "SYS"	for System Users
UDC + "OPR"	for Operator Users
UDC + "ACCT"	for Account Users
UDC + "?????"	for Other UDC's

These conventions allow the file to be easily identified as to application and owner.

III) Database Guidelines

ULL standards pertaining to databases are somewhat relaxed because all database changes and designs must be approved by the staff database administrator. It is the DBA's responsibility to review design considerations and modify databases accordingly. This quality assurance function allows analysts to stretch their imaginations when doing design work. There are some guidelines that are used for conformity reasons. These include:

Field Size	Dates will be stored in X8 format CCYYMMDD. Codes or Numbers (i.e., Customer #) will be stored in z or x format. Costs and Price elements will have four decimal places, and be stored in packed decimal format.
Dataset Names	Names are restricted to 15 characters. The only special character allowed is the dash.
Security	A simple two password system is used. One for write access, another for read only. No item level security is used.
Locking	Databases are opened mode 1 for updating, and mode 5 for read access. The opens will be conditional. Locks should be done at the lowest possible level. Locks should be applied immediately before execution of an add, update or delete function and released immediately after.
Design	The capacity of a master set should be a prime number. Plan for 80% occupancy for all master sets. Use a master set to store data associated with a unique key field. Avoid sorted paths. Avoid more than two paths into a detail dataset.

IV) View/3000 Screens

ULL makes extensive use of View in most complicated on-line applications. The combination of multiple conditional expressions and heavy dataset contention from other programs has made some rewrites from 4GL languages necessary. New development of on-line programs is done primarily in Cobol/View. The ULL standards manual discusses View in great detail in order to present screens in a uniform professional manner to the user. Some of the considerations are as follows:

Organization

Screens should be organized to present a logical path through an application. An inverted tree structure best illustrates this idea. All major tasks performed by a program should be listed in the main menu, and presented to the user first. A good design practice is that each screen performs one and only one function.

Format

Screens will conform to the same basic layout with screen name, application name and screen title along with time and date as the headings. The screen data will be arranged to logically read left to right, and top to bottom.

All enterable fields will be bound by visible brackets, or highlighted. All data fields must be labeled. This label can either be above or to the left of the data field.

Prompt message should be written so that the user clearly understands what is expected of him/her.

Error messages should identify the field in error, and explain the error if possible.

Function key assignments should remain uniform throughout an application, and the program should always be prepared for use of non-labeled function keys. Function keys should, whenever possible, be assigned so as to conform to other ULL programs.

Screens expecting input from a specific document should conform as much as possible to that document.

View error handling sequences and the VPLUS process specifications should be used.

V) Report Standards

Report standards are also important at ULL. Operations in any given month will produce 300,000 sheets of paper from over 250 report programs. It is important to present the user with a product that looks professional in every way. Although on-line systems are used extensively by ULL users, reports are still the most common method of data exchange. The following standards have been adopted to produce uniform reports.

Report headings conform to the same format independent of report type or page width. The heading includes report number, report title, application name, series name, date time and page number. There are also lines for report qualification to further explain the function of the report if necessary.

The detail of the report will be preceded by column titles to explain each data field. The labels can be abbreviated in accordance with established acronyms. A dashed line separates the data from the labels.

Summary lines are distinguished by preceding asterisks. The higher the summary break, the more asterisks are used to identify it. Summary lines will contain a label that clearly defines the type of summary (i.e., average, total, count).

Page footings are reserved for the use of security messages (i.e., CONFIDENTIAL), or attention messages (i.e., ATTENTION BRAND MANAGER).

The end of the report will be clearly marked after a page break. If a report selects zero records, an appropriate message will be displayed in place of the report data.

Arrangement of data on each line should be logically from left to right, starting with the most general and moving toward the most specific. Sort fields should be displayed in the same order as the sort. Any elements used for selection should appear on the report. Page breaks should be used to separate major divisions in the sort sequence.

VII) Programming Standards

Structured Code

ULL programmers attempt to follow a structured type of programming. Thus, the following rules apply:

Programs should be composed of a series of processes which perform one and only one function.

There should be only one entry point and one exit point from a process.

Processes should be clearly identifiable in a program.

A process should only execute code located below it in the source file.

GOTO statements should only be used carefully within a paragraph, or to recover from an error condition.

Code should be written so that someone else can maintain it.

A program should not allow control to fall through one process into another.

Good code is self-documenting at the statement level.

Naming

Naming conventions within a program should be as meaningful as possible, but not more than fifteen characters in length. Hyphenation should be used to add meaning. When abbreviation is necessary, consider dropping vowels first starting from left to right, then consonants.

Format

Format of code should include proper identation of nested statements from the conditional verb. Statements should be formatted for easy reading, and if they can't be placed on one line they should be broken so that it is obvious that the statement is continued. Blank lines should be used to segregate logical blocks of code and enhance readability. Statement labels should consist of a unique three- or four-digit prefix followed by a hyphenated set of words which describe the function of succeeding lines of code.

Documentation

The following will be provided at the beginning of each source code file:

Program Name
Description of What the Program Does
Author and Creation Date
Language

A narrative description of any restrictions or conditions that might preclude general use or execution of the program.

Lists of all input, output and update files used by the program.

Maintenance Log, describing all changes that were made to the program, when they were made and by whom.

It is important to use comments within the program to explain what the code is doing. Comments are most useful at the beginning of each paragraph to explain the code within the paragraph.

Some ULL/COBOL program standards include the use of copylib members whenever applicable, but not for database buffers. The use of the "@" list is not allowed. Programs that encounter an abort condition will call the "Quit" intrinsic instead of the STOP RUN statement. Additionally, a call to "DBEXPLAIN" should be used if appropriate if an Image call caused the abort. Compiles will be done in a batch environment whenever posible.

Summary

The above standards topics represent a summarized cross section of the ULL manual. The manual itself is far more detailed in some areas, but it always leaves room for individual preferences. The programmers are not expected to memorize the standards, but rather to use them as a guide. ULL does not, as of yet, have any type of formal program review procedure. Formal reviews can often have a negative impact on programmer morale, and morale is always proportional to production. Each programmer is expeccted to combine their educational background with their creative abilities to produce a product that will make themselves and everyone at ULL proud. The standards manual was produced by programmers, for programmers, to serve as a tool in producing a quality product.

Project Management

As discussed earlier, the standards manual makes up only half of the ULL software guidelines. The ULL project management system is also a home grown set of procedures. The project tracking system is used to maintain control of projects from their inception through their completion. It is also used to provide management with project status, and to determine if a project is on schedule or not. The system is maintained by all members of the MIS department.

Report Requests

The initial phase of a project is the request. The backlog of MIS can at times be substantial so that maintaining a current queue of requests becomes very important. The user request is completed by submitting a project request form to MIS management. Each department in the ULL organization has a designated Project Request Manager who authorizes projects. It is this person's responsibility to ensure that the project does not conflict with operations in other departments. MIS will confirm this communication in order to prevent conflict at the time of implementation.

The completed form contains all of the who's and what's associated with the request, and, most importantly, the why's. The benefits of a particular project help determine the appropriate priority, and thus the completion date. The request form is assigned a control # dependent upon the functional area to which the project is assigned. The analyst will then assign an estimated time of completion, in hours. This is a ballpark guess that will help the user decide whether or not a manual work around is feasible, and also will serve as a monetary guide when MIS begins charging back its services. If the requesting department decides the cost of the request if not justifiable, they will withdraw it.

All projects are prioritized by ULL management, and reported in prioritized order on a weekly project listing by functional area. These listings are distributed to the functional analysts for status updates. At any given time a complete update of projects can be generated with updated status and completion date. This allows management to visually see the impact of "I need this now" requests.

Project listings also allow MIS management to assign or reassign analysts and programmers to projects according to the priority of the request. The project listings separate projects into basic groups depending upon the status. There are eight status codes for ULL projects. They are:

0 = Backlogged
1 = Design/Analysis
2 = Coding
3 = Unit Testing
4 = Systems Test
5 = Awaiting Signoff
6 = Complete
7 = Withdrawn

These codes along with the request priority determine where on the listing the request will print. Throughout the project cycle, the status of a project will be changed appropriately by the responsible analyst.

Illustration #1 represents a project listing for one of our functional areas.

Project Summary

The project summary sheet is prepared by the analyst assigned to a given project. The summary sheet is used for projects that will take more than 40 hours of total time. The summary as it appears in Illustration 2 lists subtasks of a project, who's assigned to the task, and the time to complete each task. This summary is completed at the end of each week by the assigned analyst after they receive the number of hours worked from the programmers assigned to each task. The summary details work loads within the project, and allows management to use personnel more efficiently within ongoing projects. At any given time a summary will show where on the time line a project is, and also serves as a good prioritizing tool for the analyst. When using the summary it is important to remember that not all analysts and programmers perform at the same level so times may have to be adjusted. Another rule of thumb is to take into account how many hours an analyst will be able to devote to the project when calculating dates. If day-to-day concerns draw an analyst away from projects ten hours a week, remember to adjust the dates accordingly. There should also be some time allotted for miscellaneous problems that arise during the project. This miscellaneous time has usually been exhausted by the end of the project.

Program Authorization

An analyst responsible for a production account has control of all source code in that account. In order for a programmer to work on a program for any given account, the analyst must provide the programmer with a Program Authorization sheet. The authorization sheet details what program is to be worked, what project the programmer's time will be billed against and also serves as a signoff sheet for completion of the program. The authorization sheet's description area can be used to explain the necessary changes to the program, and serves as a supplement to the detailed specs that accompany the authorization sheet.

Time Sheets

Time sheets are completed by all MIS personnel in order to help track time used against project areas. Time sheets also serve as a tool to keep management informed of programmers' time spent dealing with day-to-day concerns. The time sheets are also used as mentioned above to update the project summaries.

Physical Project Cycle

The authorized MIS representative will pass to MIS management a completed request form. The manager after assigning the analyst and associated information will pass the request to the MIS support secretary for assignment of a control number and entry into the project tracking system. The request is forwarded to the asigned analyst who fills out a program authorization sheet, the project summary and any detailed specifications necessary. The package is forwarded to the MIS manager who assigns the program to a programmer. Often the analysts will have to do the programming themselves. The paperwork, however, is still required. The programmer assigned will complete the program, documentation, testing and the program authorization and return the package to the analyst for verification and implementation. The requester of the project then signs off, and the completion date is logged into the tracking system.

Future Goals

The future goals of our software guidelines are very exciting, and some of them are progressing very well. Our first goal is to start billing back departments for MIS services. We are currently tracking the time spent by personnel on each project, and are developing departmental budgets. Once the budgets are decided upon, MIS can create bill rates for each level of the staff, and begin billing. Another goal of MIS is to move the project tracking system, project summaries and time sheet entry from PC-based to HP-based. This goal will allow personnel to access each of these valuable tools without swapping paperwork. The third of our goals at this time is to automate our system documentation. We are currently reviewing a software package that will enable us to have on-line access to documentation to the variable level. These goals, when attained, will allow the MIS department to spend more time on projects and less time on paperwork.

Conclusion

Our software guidelines have made our lives significantly easier, and could help your organization as well. This system is not as detailed as some, but it's a good beginning and it leaves plenty of room to grow. This type of tracking system does not require much setup time, nor does it require a large monetary expenditure. The benefits are realized very quickly, and the results are clearly visible to upper management. It's nice to know where you've been, but it's nicer to know where you're going.

ILLUSTRATION #1

```
PROJECT LISTING AS OF:  05/14/90
          CHANGE FROM:  05/10/90
```

FINANCIALS

```
                                                      PROJECTED
                                                TYPE  DUE DATE

ACTIVE

F0002     Salesman Cash Collection Flag                 16 Hours
05/10/90  Due/Overdue During Upcoming Week
          (A.A.T.)
          Active:  05/11/90
          Closed:
          Hours :

CODING

BACKLOGGED

F0075     Breakup Element in G/L                  P
08/19/88  (Cardinal)
          (L. Koltov)
          Active:

F0020     Combined ATB                                  80 Hours
02/11/88  (P. Russell)
          Active:

CLOSED:

F0191     Cash Discount Report
02/14/90  (J. Silva)
          Active:  05/07/90
          Closed:  05/11/90
          Hours :  16.25

WITHDRAWN

F0076     Create New A/R Users                         40 Hours
08/15/88  Security
          (H. Parlon)
          Withdrawn:  05/11/90
```

ILLUSTRATION #2

UNITED LIQUORS, LTD

MANAGEMENT INFORMATION SERVICES

PROJECT SUMMARY

PAGE 1 **OF** 1

WEEK ENDING: 05/12/90

PROJECT WEEK: 01

PROJECT NUMBER: P0033 **PROJECT NAME:** Auto Generation of Receipt Number **PREPARED BY:** D. Marcotte

TASK (Activity or Prog.#)	PERSON ASSIGNED	PLANNED PROJ. WEEK NOS.	START DATE		COMP. DATE		MAN - HOURS			
			SCHED.	ACTUAL	SCHED.	ACTUAL	PLANNED	ACTUAL	REMAINING	OVER/UNDE
Initial System Study	D.M.	1	05/07	05/07	05/07	05/07	8	8	0	0
General Specifications	D.M.	1	05/08	05/08	05/10	05/11	24	30	0	+6
Menu Screen Design	P1	1	05/11	05/11	05/11	05/14	16	14	0	-2
Dataset Design	D.M.	2	05/14	05/15	05/15	05/16	8	4	4	0
PO111CS1 Detail Specs	D.M.	2	05/16		05/17		16		16	
User's Manual Changes	P1	2-3	05/18		05/21		16		16	
PO111CS1 Changes	P2	3-4	05/22		05/29		40		40	
Unit Test	P2	4	05/30		06/01		24		24	
System Test	D.M.	5	06/04		06/05		24		24	
User Training	D.M.	5	06/04		06/06		8		8	
		Cumulative totals for all tasks started this project week and prior.					184	56	132	+4

*** Not on Project Line**

ONE COMPANY'S SUCCESSFUL IMPLEMENTATION OF
SOFTWARE GUIDELINES
3093-14

Look, No Secondaries! (Optimum Master Set Capacities – From an Art to a Science)

David B. Wiseman
Proactive Systems
4 Main Street, Suite 101
Los Altos, CA 94022
(415) 949-9100

Firstly, I must come clean about the title of this paper. What I really should have said is "Look, a reasonable number of secondaries (given the blocking factor) which will yield good performance without wasting disc space!" – but that doesn't trip off the tongue easily hence the poetic license.

The subject of Secondaries in IMAGE Master Sets has received a lot of coverage in the past because they have a large impact on data base performance. Previously nobody provided a sure-fire method of predicting the number of Secondaries for any Master Set. Rules of thumb and guidelines abound – some of them certainly erroneous. The purpose of this talk is to take the subject of Secondary Estimation from the realms of an Art into the realms of an exact science.

Simply, my claim is that the number of Secondaries which contribute to poor performance is mathematically calculable.

Definition of terms

We must first clarify some of the terms we will be using (bear with me all of you who have seen this before!). IMAGE locates Master Set entries into a Set by applying an algorithm to the Key Value of the entry which is to be located. This algorithm is designed so as to minimize the number of times that two different Key Values will produce the same location in the Set. It is not possible, however, to fully eliminate this occurrence. Whenever this happens, IMAGE will locate the secondary entry into a nearby empty position within the Set. In certain circumstances this arrangement can contribute to poor performance.

A few secondaries are, in themselves, no problem. A large number of secondaries, however, can badly affect a database's performance when adding and deleting entries. If a set is exhibiting this behavior, one must be able to accurately calculate the correct parameters to change so as to reduce this bad behavior to an acceptable level.

The Number of Secondaries

There exists a formula for calculating how the actual number of secondaries, which contribute to poor performance, can be mathematically calculated. This formula is:

$$S = 1 - (1 - e^{-F}) \div F$$

where, S = Number of Secondaries ÷ Number of Entries
F = Number of Entries ÷ Capacity of the Set
and e = Base of Natural Logarithms (2.7183)

This theoretical formula states that the Number of Secondaries only depends upon the Capacity of the Set and the Number of Entries within the Set. In nonmathematical terms what this means is that the quantity of entries which can contribute to poor performance increases, in an exponential fashion, as the set gets fuller.

In developing an algorithm to calculate a good capacity to recommend, we began to question the validity of this existing theory but we needed a large amount of empirical data in order to test its accuracy.

We first wrote a routine which would do the same job as IMAGE's algorithm (sometimes called IMAGE's Hashing algorithm). Having done that, we could take a Master set and apply this routine in turn to each entry to calculate the Primary position within the set – noting as we went along how many Secondaries occurred. It was now possible to do this for any range of capacities and key values that we wanted – the results are shown in graphical form in Figures A thru C for three different key types and sizes.

We plotted the actual observed number of Secondaries against the capacity and also the theoretical Number of Secondaries using the formula above. As can be seen form Figure A, the actual trend matches well to the theoretical, but the individual number of secondaries varies widely about the theoretical. Figure B shows even more wild behaviour with some capacities being particularly poor. The wildest behaviour is shown in Figure C. Here we see a definite segregation of capacities into good, poor and downright awful.

Prime Numbers, Good or Bad?

Now is the time to lay a ghost down for good. It was originally suggested by some IMAGE "experts" that capacities which are Prime numbers will tend to give better results. A look at figure D will will show that there is no truth to this mischievous rumor. Each Prime number capacity has been marked with an "*" on the graph and there are just as many "*"'s on the peaks as on the troughs. So here is real empirical evidence to confirm that Prime Numbers are a Red Herring.

However, there is a class of capacities which should definitely be avoided at all costs. Research has shown that any capacity which is a multiple of 64 should be avoided. This would appear, from the evidence, to be a general result for all Character Keys (X, P, U and Z types) to a greater or lesser degree. Numeric Keys (I, J, K and R types) also exhibit this problem but tend to be even worse. Figure E shows that every single capacity which was a multiple of gave at least twice as many Secondaries as other capacities. Looking back at Figure C (an extreme case) the highest peaks are multiple of 64, the next highest are multiples of 32 and so on. The lowest values are from Capacities which are simply odd numbers (not Primes or anything else special).

Extended Theory

What can be said about the existing theoretical secondaries formula? The number of secondaries is obviously a function not only of the capacity and the number of entries, but also of the characteristics of the particular key value distribution. That is, for any capacity and number of entries there could be a different range of key values, and the number of secondaries will vary with this different distribution. But by how much? Well, the majority of cases seem to match the theoretical formula fairly well. However, certain cases, typically numeric or 2-word keys with poor distributions of key values, give results which on average parallel the theoretical curve, but with a constant offset (either higher, or even lower in some cases). Figure F shows two examples, the majority of examples matched those in Figure A – i.e. the average number of Secondaries tended to equal the existing theories predictions however certain cases, typically as said numeric or two word keys show the constant offset behaviour.

This leads to the first mathematical result:

> The number of secondaries, averaged about each capacity, tends towards the formula:
>
> $$S = 1 - (1 - e^{-F}) \div F + k$$
>
> where k is the "Key Distribution Offset."

The "Key Distribution Offset" is the constant offset mentioned above, and represents an improvement, or degradation, to the theoretical quantity of secondaries due to the specific spread of key values actually in the set. Positive values of k indicate that the key values in the set tend to cluster together more. Negative values indicate a better spread of values. In many cases the value of k will be zero, but n the second graph in Figure F it is equal to 0.11 (or 11%). The above formula can be used to estimate the average number of secondaries at any capacity, but only by examining the current state of affairs to measure k. More usefully one can calculate the approximate capacity which would yield a manageable number of secondaries (say between 15% to 25% of all entries).

It is important to note that this does not guarantee that the capacity so calculated will definitely yield this exact number of secondaries. Because of the erratic behavior of any single capacity one must perform a sampling test for a range of capacities about the chosen capacity to select the capacity which gives the lowest number of secondaries.

The Effect of Blocking Factors

When attempting to locate a secondary, IMAGE will first try the same block of entries in which the primary entry is located. Now, if a secondary is successfully located in the same block as the primary this does not lead, in itself, to a large degradation of performance. When adding, deleting or locating the secondary, IMAGE must do a little more – two attempts instead of one – but this is ideally all done in memory without any need for swapping of buffers or extra disc I/Os (if you're short on memory though, this will not help!). Measurements indicate that the time taken to access a secondary within the same block as its primary is approximately 15% to 20% of the time taken to read that block in from disc (although this will vary with processor speed – these figures are for a midrange classic 3000 machine). In terms of elapsed time, reading a secondary within the same block as the primary takes about 1.2 times the time to read in the primary itself. Taking disc caching or XL mapped file access into account this figure actually rises to about 50% (because the time to originally read the primary decreases with caching enabled).

Three things, however, really do hurt performance:

a) Large numbers of secondaries from the same primary location (i.e., long secondary chain),

b) Lots of full blocks (high clustering), and

c) Large numbers of secondaries in blocks other than the primary location's block (inefficient secondaries).

Inefficient Secondaries

The term "Percentage Inefficient Secondaries" is an important statistic for an IMAGE Master Set, and is defined as the number of secondaries which lie outside of their primary block, as a percentage of the number of entries. The fact that another IMAGE Block must be read in from disc, or memory, to locate the secondary entry results in a larger overhead than that previously discussed. With or without caching the time taken to read in an inefficient secondary would be a minimum of twice the time taken to read a primary (assuming the primary was in cache to start with). If the primary was not in cache the situation improves because the inefficient secondary will be – leading to a factor of 1.5 rather than 2. One or two of these inefficient secondaries is not going to upset the apple cart very much, but when the percentage of inefficient secondaries reaches approximately 23% or more one can expect to see performance begin to degrade – not only on adding and deleting but also on calculated master reads.

To see how this value varied, a large number of experiments to load up master sets at various capacities and blocking factors, key types, number of entries etc, were performed.. The results were recorded and analyzed, and an empirical mathematical formula deduced which accurately matched the data. The formula is:

$$I = e^{x} \text{ where } x = \ln(S) - V(B-1)$$

where, S = No of Secondaries ÷ No of Entries,
B = Blocking Factor,
I = Number of Inefficient Secondaries,
and V = "Key Value Randomity" (see below).

Again, in nonmathematical terms, this says that the quantity of inefficiently located entries in a set decreases, exponentially, as the blocking factor increases. The rate of decrease is determined by the factor we have called the "Key Value Randomity."

Figure G illustrates the close fit of this formula against the actual data. For each graph the parameter V was established using the above formula. We have given this factor the name of "Key Value Randomity" because it depends upon the actual spread of key values as well as the distribution of those keys throughout the set due to IMAGE's hashing algorithm. The purpose of the hashing algorithm is to enable a random distribution of primary positions. The parameter V measures just how successful IMAGE has been in this endeavor. Typically, values of V seem to range from 0.001 (very poor randomity) to 0.700 (very good randomity).

Optimum Capacity Selection Procedures

The above empirical theory gives us a good basis upon which to calculate the optimum capacity for any master dataset. Some general rules are:

1) Avoid Capacities which are multiples of 64.
   Powers of two are always awful, and don't select a Capacity of 1,024 under any circumstances!

2) For Numeric keys avoid Capacities which are even numbers.
   In many cases only odd Capacities gave reduced Secondaries for Numeric Key Types (I, J, K & R). This is not, however, a hard and fast rule as it depends entirely on the range of Key Values within the set. By selecting a Capacity above the highest Key Value in the set it is even possible to have absolutely zero Secondaries. So, this is simply a guideline.

3) Do not select a Capacity just because it's a Prime Number.

4) Test the actual key values against any Capacity selected.
   Experiment shows that varying the capacity up and down by 4 or 5 will enable a capacity to be selected which results in less

secondaries than any of the others. The individual variability of specific capacities must be examined before selecting the capacity, so as to weed out the bad performers.

5) Take the Blocking Factor into account.
The most important statistic of a Master Dataset is the percentage of inefficient secondaries, not the percentage of secondaries themselves. It is this figure which quantifies how inefficient in terms of performance these secondaries are – and this figure, in turn, depends upon the blocking factor.

The method of "Good Capacity Selection" first requires the current values of the two key distribution parameters k and V to be determined. In doing this we are establishing how the current spread of key values within the set affect the number of inefficient secondaries generated by IMAGE. From these two values the approximate location of a capacity which will give, say, a percentage inefficiency of 15% or less can be established (given the current blocking factor). An examination of the capacities either side of this capacity against the actual key values will lead to a capacity which will, ideally, give the smallest number of secondaries. (In the real world absolutes are unobtainable, but we can get close!)

A better alternative, given the resource of enough Blockmax, would be to increase the blocking factor (either as well as, or instead of, increasing the capacity). An appropriate blocking factor may also be estimated from the above empirical mathematical formulae. However, too high a Blockmax reduces the number of IMAGE buffers available. Current thoughts indicate that in most cases the Blockmax should not be raised above 1024 or 1536 words (due to the potential restriction this places upon the number of buffers available). However, more empirical work needs to be done to determine the exact relationship between number of buffers, block sizes and performance.

How Many Secondaries are Too Many?

All the above theory may be academic interest, but what practical use does it really have? Well, a reduced amount of secondaries will definitely improve DBPUT and DBDELETE performance. If an application uses DBPUTs and DBDELETEs intensively this can become crucial to application run times. Also, excessively long secondary chains can affect DBGET and DBPUT performance if IMAGE has to traverse these chains each time a mode 7 DBGET or DBPUT is required. If a significant proportion of these secondaries lie outside of the primary's block (i.e., inefficient secondaries) then performance is further degraded.

As an illustration of how these factors actually affect performance, compare the following run times. The following timings were done on a range of datasets, varying the number of secondaries and inefficient secondaries in each set. Three operations were timed on a midrange classic 3000 machine, single user in batch mode, with disc caching (similar results were obtained without caching). These operations were:

A) a complete load of the set using DBPUTs.
B) a serial DBGET & DBDELETE of the entire set.
C) a serial DBGET of the entire set.

By subtracting the serial DBGET time from the DBGET/DBDELETE time we get the time taken to solely perform the DBDELETEs. Adding this to the DBPUT time and dividing by the number of entries in the particular set we get the average time taken to perform one DBPUT and one DBDELETE.

%age	Blocking	Secondaries		Elapsed Seconds (per Entry)			CPU Seconds (per Entry)		
Full	Factor	%age	Ineff	PUT	DELETE	Total	PUT	DELETE	Total
58%	11	24.3%	1.4%	0.048	0.045	0.093	0.033	0.031	0.064
58%	10	24.3%	4.2%	0.048	0.046	0.094	0.033	0.031	0.064
67%	12	43.0%	9.8%	0.049	0.067	0.096	0.035	0.032	0.067
58%	2	24.3%	14.0%	0.055	0.048	0.103	0.039	0.032	0.071
80%	3	31.7%	19.1%	0.062	0.052	0.113	0.042	0.031	0.073
64%	1	25.9%	25.9%	0.092	0.066	0.158	0.048	0.029	0.077
99%	4	37.0%	30.2%	0.184	0.054	0.238	0.164	0.034	0.198
99%	2	37.0%	34.2%	0.323	0.057	0.380	0.299	0.034	0.333

These are just some of an extensive range of timings, from which one can accurately state just how many Inefficient Secondaries are too many. The difference in average timings highlights the excessive run times beyond approximately 23% Inefficient Secondaries (the "knee in the curve" occurs at just beyond 23%). In fact performance begins to drop at around the 16% level, and really falls off at 23%. The above theory can be used to establish a correct capacity which will provide this acceptable level of inefficiency. The DIAGNOSE module of FLEXIBASE has been modifed in line with this theory so as to recommend appropriate capacities for the spread of key values actually in the set.

Other Constraints

Additional constraints on the selection of a Master Set Capacity are those of providing sufficient room for expansion (i.e., not too low a capacity) and remaining within acceptable serial DBGET limits (i.e., not too high a capacity). Available disc space and time must, of course, also be taken into consideration.

Something like 10% to 15% of the Set should be available purely for new entries, otherwise one would be increasing the capacity far too often. Some Sets, of course, do not change at all, with the other extreme being sets which are continually filling up and emptying out.

Any application which serially reads a Master Set is going to have its run times determined by the capacity. IMAGE will read each potential entry in the set, whether in use or not, until it reaches the last position in the set. Thus, if one selected a capacity which gave 0% inefficient secondaries at the expense of being only 15% full then serial DBGETs would really suffer.

The other area for consideration is the average length of the secondary chains within the set. It is possible to have a small number of inefficient secondaries but for them all to be on one or two long secondary chains. In this circumstance performance is degraded when accessing those secondaries. However, this is a refinement to the basic principle and is, luckily, not a common occurrence in sets with less than 30% secondaries.

Conclusion

I have tried to show the selection of an appropriate Capacity is no longer a matter of guesswork or even an arcane incantation, but a predictable Science which is subject to the laws of Mathematics and Statistics. In summary:

a) There is a demonstrable relationship between the capacity of an IMAGE Master Dataset and the number of Secondaries in that set.

b) More importantly there is a relationship between the Capacity, Blocking Factor and the Number of Secondaries in that set.

c) Prime Number Capacities are not useful in determining an appropriate Capacity for an IMAGE Master set.

d) The actual key values in the dataset are vital in determining an appropriate Capacity.

e) There is a measurable performance degradation in applications which PUT and DELETE entries into an IMAGE Master set which has a high number of Inefficient Secondaries.

f) Finally, the above factors can be quantified and expressed mathematically thus an appropriate Capacity and Blocking Factor may be calculated given the current entries in a Set.

To show that this final point is really possible, here are two sample runs if a Diagnostic module in the FLEXIBASE software product which puts all of the above theory into practice. It does all the calculations for you so as to give suitable recommendations for improving performance.

Run 1 - Poor Secondary Inefficiency:

```
1) Set (Manual)    MANUAL:
Search Item - NOMINAL-CODE              Type X12
Number of Items                    2    Entry Length (words)          120
Number of Paths                    0    Blocking Factor                 2
Capacity                         248    Block Length (words)          251
Number of Entries                214    Total Number of Blocks        124
Percentage Full                86.3%    Clustering Factor             .05
Percentage Secondaries         37.9%    Average Secondary Chain      1.61
Percentage Inefficient         29.4%    Ldev Num of First Extent        2
Key Value Randomity             .251
   RECOMMEND - Increase BLOCKING FACTOR to   4 to optimize buffer usage
   RECOMMEND - To reduce Secondary Inefficiency change CAPACITY to    324
```

Run 2 - Same Set after implementing the above recommendations:

```
1) Set (Manual)    MANUAL:
Search Item - NOMINAL-CODE              Type X12
Number of Items                    2    Entry Length (words)          120
Number of Paths                    0    Blocking Factor                 4
Capacity                         324    Block Length (words)          501
Number of Entries                214    Total Number of Blocks         81
Percentage Full                66.0%    Clustering Factor             .02
Percentage Secondaries         25.2%    Average Secondary Chain      1.34
Percentage Inefficient          8.4%    Ldev Num of First Extent        3
Key Value Randomity             .366
   RECOMMEND - No action at this time
```

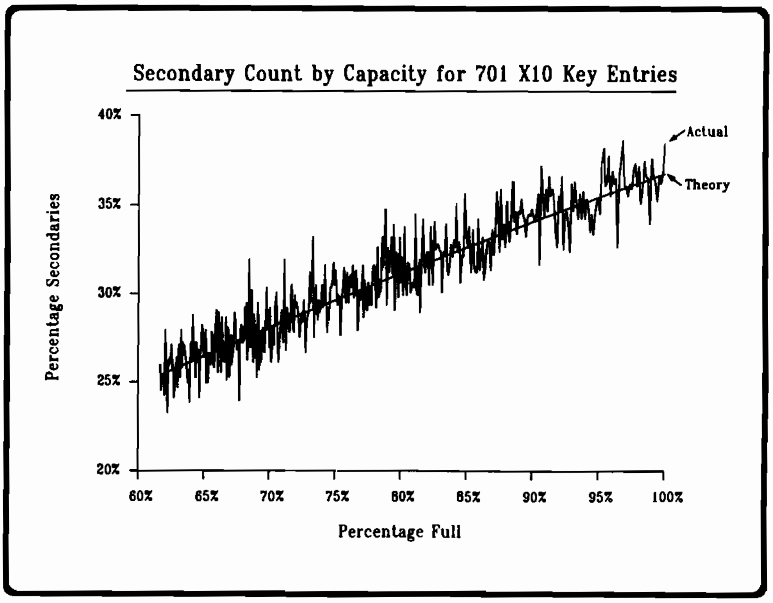
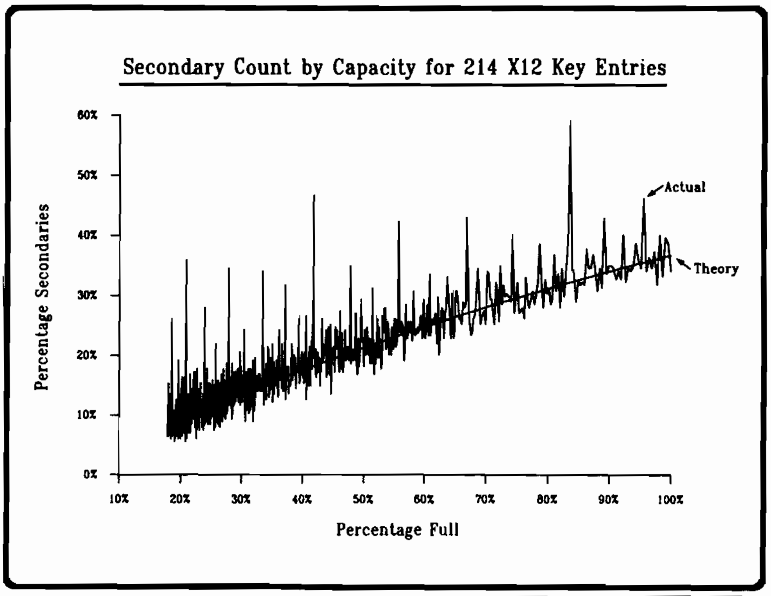
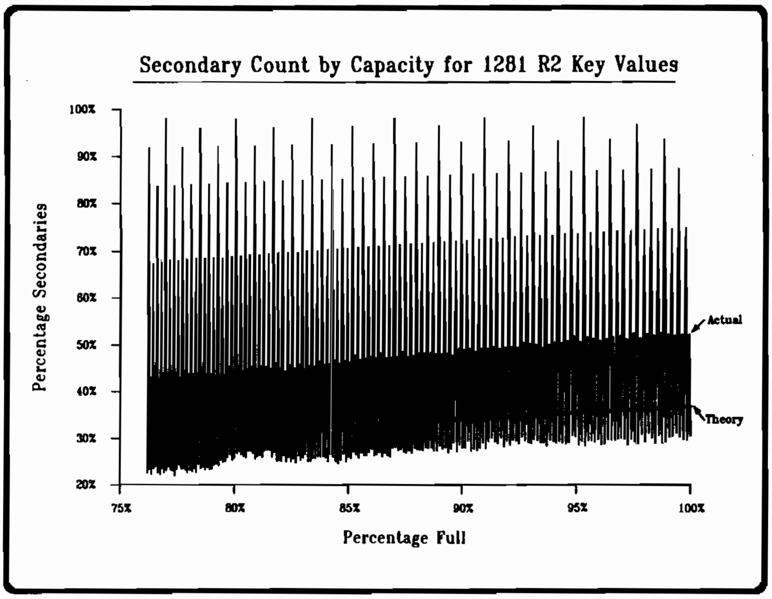

Secondary Count by Capacity for 701 X10 Key Entries
Percentage Secondaries
40%
35%
30%
25%
20%
60%
65%
70%
75%
80%
85%
90%
95%
100%
Percentage Full
Actual
Theory

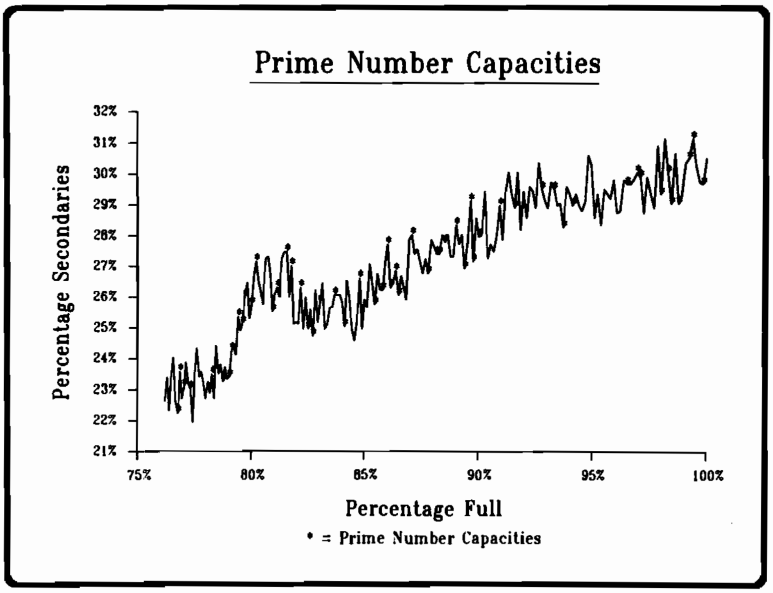
FIGURE A


Secondary Count by Capacity for 214 X12 Key Entries
Percentage Secondaries
60%
50%
40%
30%
20%
10%
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
Percentage Full
Actual
Theory



Secondary Count by Capacity for 1281 R2 Key Values
100%
90%
80%
70%
60%
50%
40%
30%
20%
Percentage Secondaries
Actual
Theory
75%
80%
85%
90%
95%
100%
Percentage Full

FIGURE C



Prime Number Capacities
32%
31%
30%
29%
28%
27%
26%
25%
24%
23%
22%
21%
Percentage Secondaries
75%
80%
85%
90%
95%
100%
Percentage Full
* = Prime Number Capacities

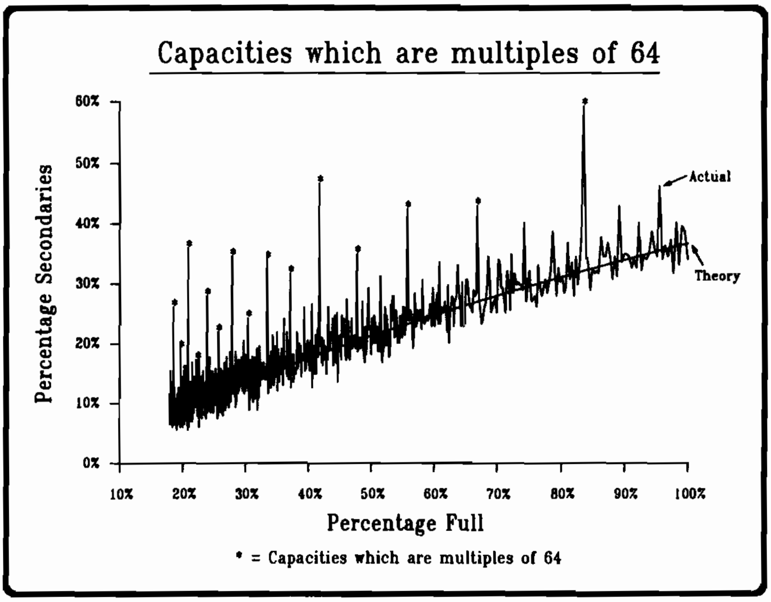
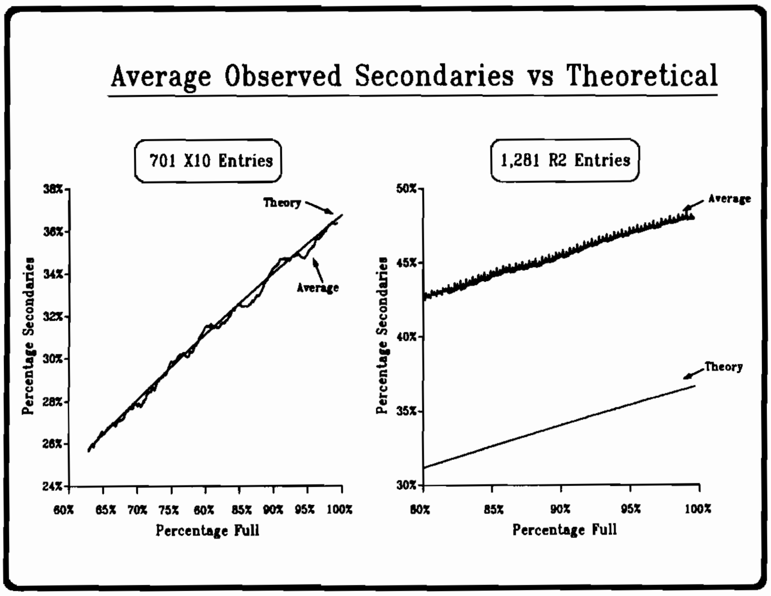
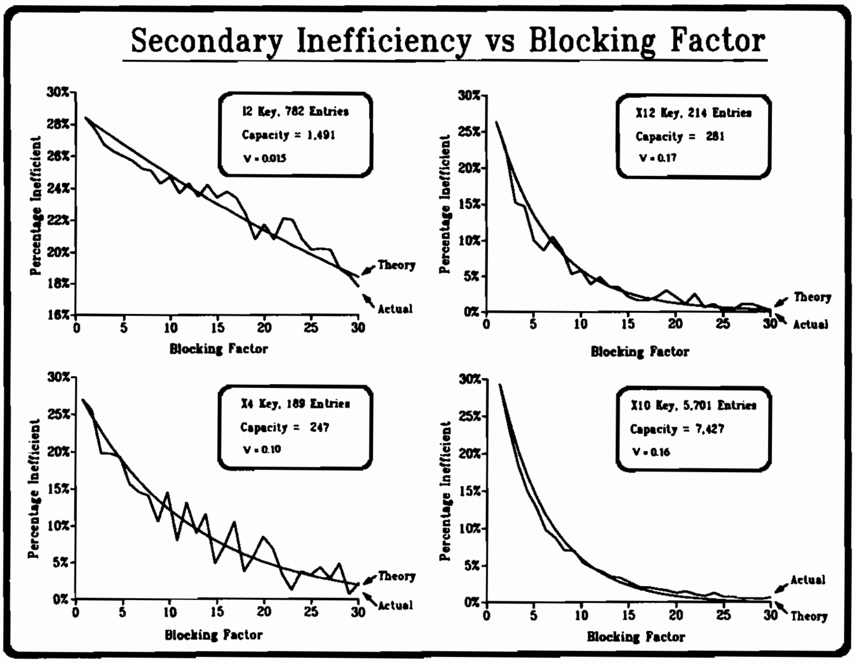

Capacities which are multiples of 64
Percentage Secondaries
60%
50%
40%
30%
20%
10%
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
Percentage Full
Actual
Theory
* = Capacities which are multiples of 64

FIGURE E


Average Observed Secondaries vs Theoretical
701 X10 Entries
1,281 R2 Entries
Percentage Secondaries
38%
36%
34%
32%
30%
28%
26%
24%
60%
65%
70%
75%
80%
85%
90%
95%
100%
Percentage Full
Theory
Average
50%
45%
40%
35%
30%
80%
85%
90%
95%
100%



Secondary Inefficiency vs Blocking Factor
I2 Key, 782 Entries
Capacity = 1,491
V = 0.015
X12 Key, 214 Entries
Capacity = 281
V = 0.17
X4 Key, 189 Entries
Capacity = 247
V = 0.10
X10 Key, 5,701 Entries
Capacity = 7,427
V = 0.16
Percentage Inefficient
Blocking Factor
Theory
Actual

FIGURE G

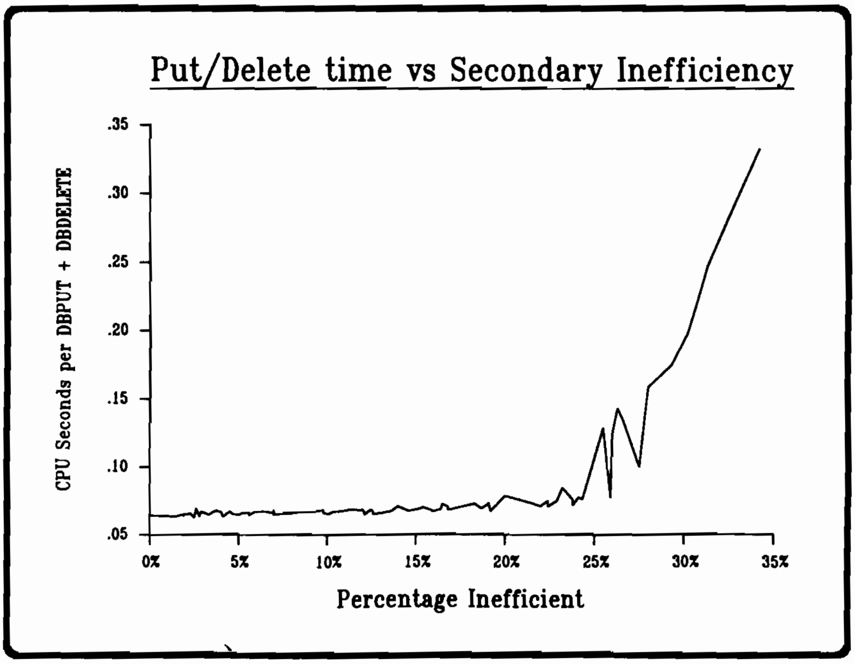

Put/Delete time vs Secondary Inefficiency
.35
.30
.25
.20
.15
.10
.05
CPU Seconds per DBPUT + DBDELETE
0%
5%
10%
15%
20%
25%
30%
35%
Percentage Inefficient

FIGURE H

Performance in Relational Databases

Eric Savage
Dynamic Information Systems Corporation
652 Bair Island Rd., Suite 101
Redwood City, California 94063
(415) 367-9696

There is little doubt that relational databases are the "way of the future." Enormous sums of money are being invested in these database management systems, both by software manufacturers and by users. As promised, relational databases provide flexibility that has been needed for some time.

The most important consideration governing the success of a relational system will be performance. If a system does not perform sufficiently, then it loses its value. Flexibility and ease of use no longer matter.

The first glimpses of these relational systems show that they have performance problems, especially with complex searches and large databases. This paper will explain these performance pitfalls and discuss possible solutions.

What distinguishes a relational database?

By now, you must have read at least forty definitions of a relational database. Most of them are about the same. This paper will not add a forty-first, other than to give the most simplistic definition:

A database is relational when certain conditions are met: their rows and columns must not be in any specific order; each row must be unique; and there cannot be any physical links between tables.

This is a shorthand version on the rules that define a relational database. But these rules don't reveal the benefits of a relational database. The primary benefits are:

A relational database allows you to link or join a table to any other table that shares a common item. It is not required that this common item be a key. You may change the physical structure of the database without requiring everyone else to stop using the database. You can create indexes as you need them, without requiring exclusive access to the database. Relational databases are accessible through the language SQL, a straightforward, simple language that can be used in both 3rd and 4th GL's. And because of their unrestricted structure, most relational databases are portable across more than one hardware platform. These flexibilities distinguish the relational database to the programmer and the user. These are the "flexibilities" that make relational databases the "way of the future."

If Only Life Were That Easy . . .

Relational databases are powerful, but they are not fully developed. Much effort has been spent making the data structure flexible. Indexing has been made straightforward. The table joining and selection process in SQL is uncomplicated. But the data structure is made flexible at the cost of control. The indexing structures are easy to install, but they are not powerful and flexible. And retrievals are made straightforward, but often they are not fast.

Most companies that sell relational databases ask the customer to rely on the ever-increasing speed of computers to resolve these drawbacks. This is a dangerous approach. We have learned from previous generations of database management systems that throwing more power at an incomplete design does not make an effective system.

There is a lot of emphasis in today's market to switch to relational databases. As a result, most executives are adopting a strategy of developing new applications using relational databases, while leaving existing applications alone. The applications being designed for the 1990's will be based in relational technology. Attention needs to be paid to the weaknesses of today's relational databases. If this is not done, then a different price will be paid. Systems may not perform as expected, and the customer service may not be provided as promised.

The Cost of Flexibility

The structure of relational databases is indeed flexible. Unlike previous generations of databases, relational databases allow items, tables, and indexes to be added at any time. Tables and indexes can be dropped, and capacities can be increased at any time. Survivors of previous generations of databases celebrate these improvements.

These improvements, though, were not free. They present serious problems in locking, especially in databases with many concurrent users. Previous generations of databases had relatively simple locking issues. Locks were placed when data was added, deleted, or updated. Relational databases add a new level of complexity. Parts of a table are now locked when reading the database, allowing others users to read at the same time, but not to write. And entire tables are locked exclusively while any restructuring occurs.

If many people are using the database at the same time, this can be catastrophic to performance. If restructuring occurs during production hours, everyone will wait until it completes. While relational databases do not require everyone to disconnect from the database while restructuring occurs, they do require that their work be suspended. In this way, it is impractical to consider the database to be highly dynamic if you have multiple users.

The locks issued when reading the database do not cause problems to other readers. These locks generally allow others to read the same data at the same time. However, updating data requires exclusive access to that portion of data. Updates must wait until other users have finished reading the data before they can complete. Consider a person who retrieves a record, expecting to make some sort of change to the record. Relational databases provide that the record will be locked so that others can read it, but not update it. If the person is delayed in updating because of research, or because it is time for lunch, the lock persists, and no change can be made to this data.

This scenario is made even worse. Most relational databases lock at either a page level or a table level, but do not allow locking a single record. This means that surrounding records that are unrelated are also locked, even though they are 'innocent bystanders'.

These approaches to locking reduce the throughput that can be achieved by relational databases. This will be especially true for online applications where much reading and updating occurs, such as order entry and manufacturing.

Relational databases emphasize a flexible data structure. The issues of locking determine how much of this flexibility is useable. A database with many users will find this flexibility expensive, and perhaps impractical if performance is to be maintained.

Retrieving Data in a Relational Database

Relational databases simplify the retrieval of data. The programmer is no longer required to determine the approach to a search, or the order to access the tables. This allows programs to be greatly simplified, and makes the work of the programmer much easier. The relational database analyzes the search, and picks the best approach. This analysis relies on two factors. First, the database considers the composition of the search, including which operators are used, which tables are involved, and the size of the tables. Secondly, it considers what indexes are available.

The determination of how to approach the search is the trade secret of each relational database company. It is often their biggest source of pride. Unfortunately, their effectiveness is limited by the type of indexing structures that are available. This is analogous to the carpenter -- a master carpenter cannot be effective without a good set of tools, no matter how great his skill.

The indexing structures available in relational databases are simple. They are powerful only in the simplest of searches. And in today's business world, the simplest searches often do not satisfy the needs of the company.

The indexing structures available in relational databases need to be improved. Until then, it is critical that the developer understand what is missing.

Database Indexing Structures

To be powerful and flexible, a database must be able to index a variety of situations. Indexes should be able to locate unique records, to locate multiple records that share a single value, and to locate multiple records that share multiple values. Additionally, the indexes should facilitate viewing records in a variety of orders.

There is no single method of indexing today that meets all these requirements. This is analogous to the carpenter's toolkit -- there is no tool that does everything a carpenter needs to build a house. It is necessary to use a variety of indexing structures.

There are four common methods of indexing to choose from. Some database systems use a method called hashing. Hashing uses an algorithm to relate a value of an item with a location in a file. This method is effective for locating single records that have unique keys (eg. locating a customer by customer number). Another method, doubly-linked lists, maintains pointers to associated records within each individual record. An example of this is a human phone tree, where one person calls another, who then calls another, and so on. Eventually everyone is called even though no one person had the entire list. This approach is useful for retrieving many records that share the same value (eg. locating all customers in a particular state).

The third method of indexing, B-Tree's, is the most common indexing methods in the computer industry. They are the predominant method of indexing used in relational databases. A B-Tree maintains records (or pointers) in sorted order. Using a technique called a binary search, a single record can be quickly located. A common example of a B-Tree is the telephone directory. A binary search is the same technique you use to locate someone in the telephone directory -- you keep narrowing the pages until you locate the page containing the person who want to call. And like the phone book, records can then be viewed in sorted order.

B-Tree's also provide the general capabilities of hashing and doubly-linked lists. They allow you to locate a unique record quickly. Additionally, all records sharing the same value will be next to each other, since they are maintained in sorted order. Therefore, retrieving all records that share the same value is fast.

A drawback to these three methods of indexing is that they are only powerful in simple searches. They are not powerful for multiple criteria searches. These three methods of indexing require that you use only one key for your search. Even if your search references a second item that has also been indexed, the index will not be used.

Relational databases today have performance problems with complex searches, especially with large databases. The primary reason for this is the lack of the fourth indexing structure.

The Fourth Indexing Structure

Back in the Dark Ages of Databases, we required everyone to know their customer number, the product number, the order number, the PO number, etc. But today, we are in a Renaissance of Customer Service. We are aiming to please the customer. Therefore, we expect to find records based on last names, first names, nicknames, cities, addresses, product descriptions, order dates, etc.

A classic example of this is the need to 'recognize' your customers (meaning retrieve their record) based on anything except their voice print. Imagine calling a major computer company and saying, "Hello. My name is Eric Savage. I purchased a computer last summer. I would like to place another order for the same thing." Unless you have a very personal sales representative, a large company using one of today's relational database management systems would not be able to answer effectively. And as more emphasis is placed on service, and as our applications become more sophisticated, these searches become more integral to our success.

The relational database made it easy to store the data. It made it easy to index various items. It made it easy to word the search. But on a large database, the search will not be fast enough to respond effectively. Ideally, the sales representative would answer, "Hello, Mr. Savage. I see you purchased a Portable 286 Laptop Computer last June. How did that work out for you? And are you still working for Dynamic Information Systems Corporation?". But this answer is only possible if the relational database presents this information in a matter of seconds. On most large relational databases, this will not happen. Your company must answer, "Hello Mr. Savage. I'm sorry, but do you remember the specific model of computer that you purchased? And which company do you work for?"

This is an example of a multiple criteria search. Multiple criteria searches are best serviced by the fourth indexing structure -- inverted lists. As of this writing, this type of indexing is only available on one of the major relational database systems, which runs primarily on the IBM Mainframe platform.

An inverted list index is one of the most powerful indexing structures available. While few database management systems employ it directly, this method of indexing is commonly used in everyday life. The indexes found at the back of a book the most common example of an inverted list. This index reverses the organization of important words from the text of the book. These words are organized alphabetically, rather than in the order they occur in the text. This allows you to quickly find information on something that is important to you.

This method is effective for multiple criteria searches. If you needed to locate all entries related to SQL and Indexes, you could look up both words in the back of the book, and note which page numbers are common. Using this same approach, you can make the search as complex as you like.

The speed of computers can make this approach quite powerful. The complexity of multiple criteria searches can be easily managed in a well designed index. Using this index, a complex, online retrieval can be performed in fractions of seconds. And most importantly, the most precious resource on all hardware platforms -- physical I/O -- is conserved.

Exceptions to the Rule

There are always exceptions to the rules. This is true with the need for inverted lists. Small databases can be scanned with today's fast computers without need for indexes. This is because today's computers can load most, if not all, of a small database directly into memory. As the database grows, though, this advantage is lost. As the database becomes larger than the available memory, inverted lists prove their worth.

Another exception exists with batch reporting where a large portion of the table is evaluated. A powerful technique can be used that applies selection criteria to each table, and then merges the result. Most relational database systems employ this technique. While the issues of locking still apply in this situation, batch performance can be satisfactory.

How to Approach Relational Databases Today

Relational databases are not fully developed. The locking strategies need to be improved, and more sophisticated indexing structure must be provided. Until this is done, relational databases will not fully capture the market. As the size of a relational database increases, and as the number of concurrent users increases, performance will falter.

At the same time, relational database represent a technological breakthrough. The flexibilities provided by relational databases permanently change the role of the application programmer.

While I recommend caution in migrating to a relational database, I also endorse their strengths. Applications that are relatively small and straightforward can benefit greatly from relational technology. Applications that are do not require online retrieval and online updates will perform satisfactorily. In the future, the performance issues with relational databases may be resolved, allowing the benefits to occur on the larger, more complex databases.

The relational and non-relational databases that exist today should both be components of the developer's toolkit. Care should be taken to insure that the right tool is used for the task at hand. With the right tools in hand, companies can be successful in building their future.

David Robinson
PowerSpec International
403 Cross Lake Drive
Fuguay-Varina, NC 27526

QUIZ/QTP Performance: Problem Areas

POWERSPEC Inc.
The PowerHouse® Specialists

Interex '90

QUIZ & QTP PERFORMANCE

PROBLEM AREAS
Version II

®

StarBase is a registered trademark of Cognos, Inc.

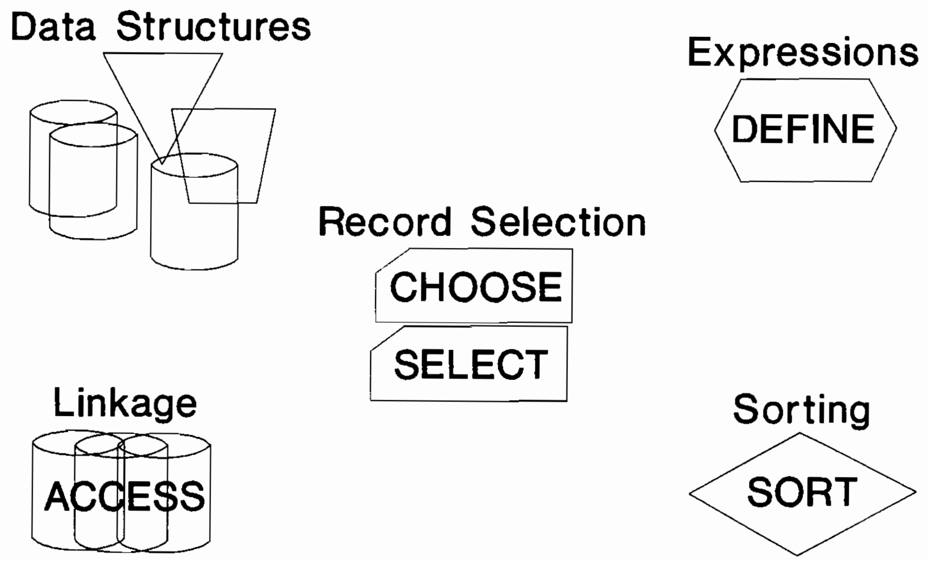
QUIZ, QTP, and QDESIGN are trademarks of Cognos, Inc.

All other marks mentioned are the property of the respective trademark holders

David G. Robinson

PowerSpec Inc.

The Problem Areas



Data Structures

Linkage

Record Selection

Expressions

Sorting

The Data Structures

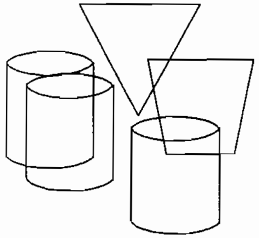
Logical Data Base Design

Normalization

Mapping Entity Relationships

Analyze Frequency Ratios

●●●



The Data Structures

Why Normalize ?

●●●●●●●●●●

Flexibility

Reduction of Redundancy

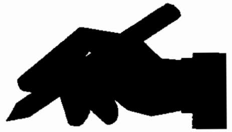
Elimination of Duplicate Data

Less Maintenance

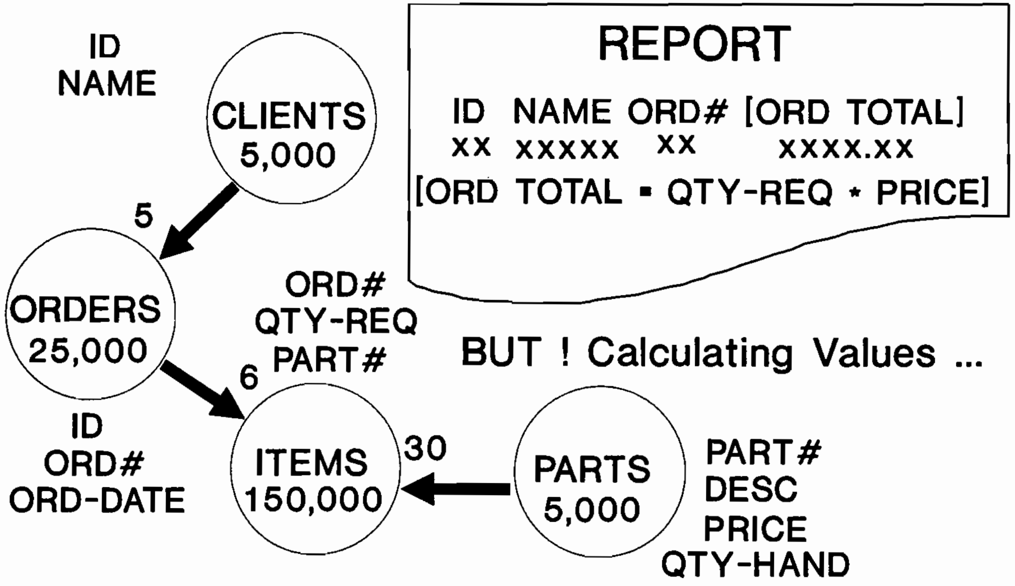
PowerHouse Performance

But !

Calculating Values ...



The Data Structures

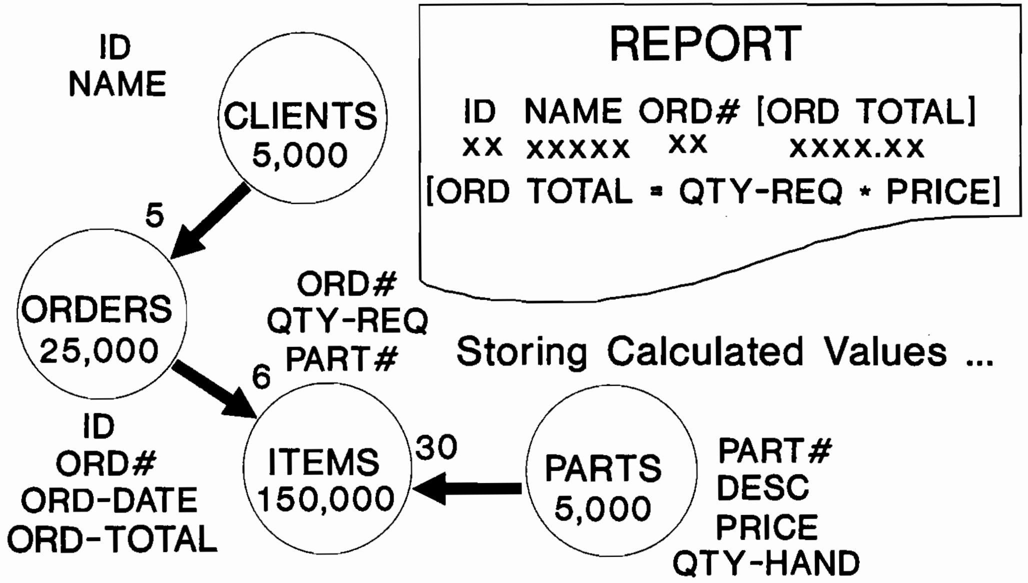


REPORT

```
ID  NAME ORD# [ORD TOTAL]
XX  XXXXX  XX     XXXX.XX
[ORD TOTAL = QTY-REQ * PRICE]
```

BUT ! Calculating Values ...

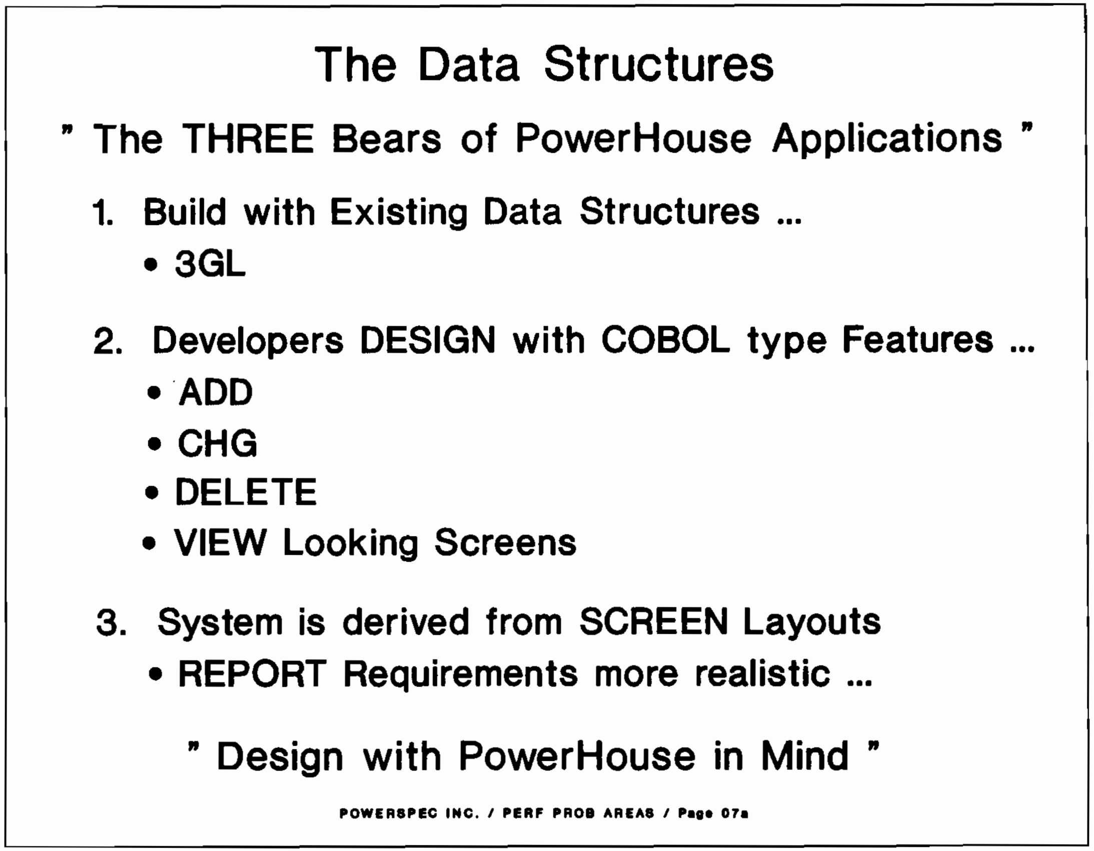
The Data Structures



REPORT

```
ID  NAME ORD# [ORD TOTAL]
XX  XXXXX  XX     XXXX.XX
[ORD TOTAL = QTY-REQ * PRICE]
```

Storing Calculated Values ...



The Data Structures

" The THREE Bears of PowerHouse Applications "

1. Build with Existing Data Structures ...
   - 3GL
2. Developers DESIGN with COBOL type Features ...
   - ADD
   - CHG
   - DELETE
   - VIEW Looking Screens
3. System is derived from SCREEN Layouts
   - REPORT Requirements more realistic ...

" Design with PowerHouse in Mind "

Data Structures

➡ Linkage

Record Selection

Expressions

Sorting

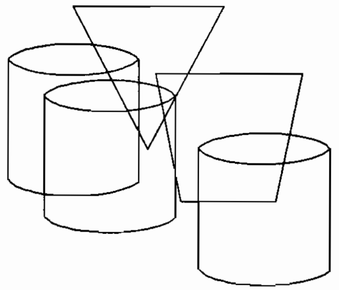
The Data Structures

PHYSICAL DATA BASE DESIGN

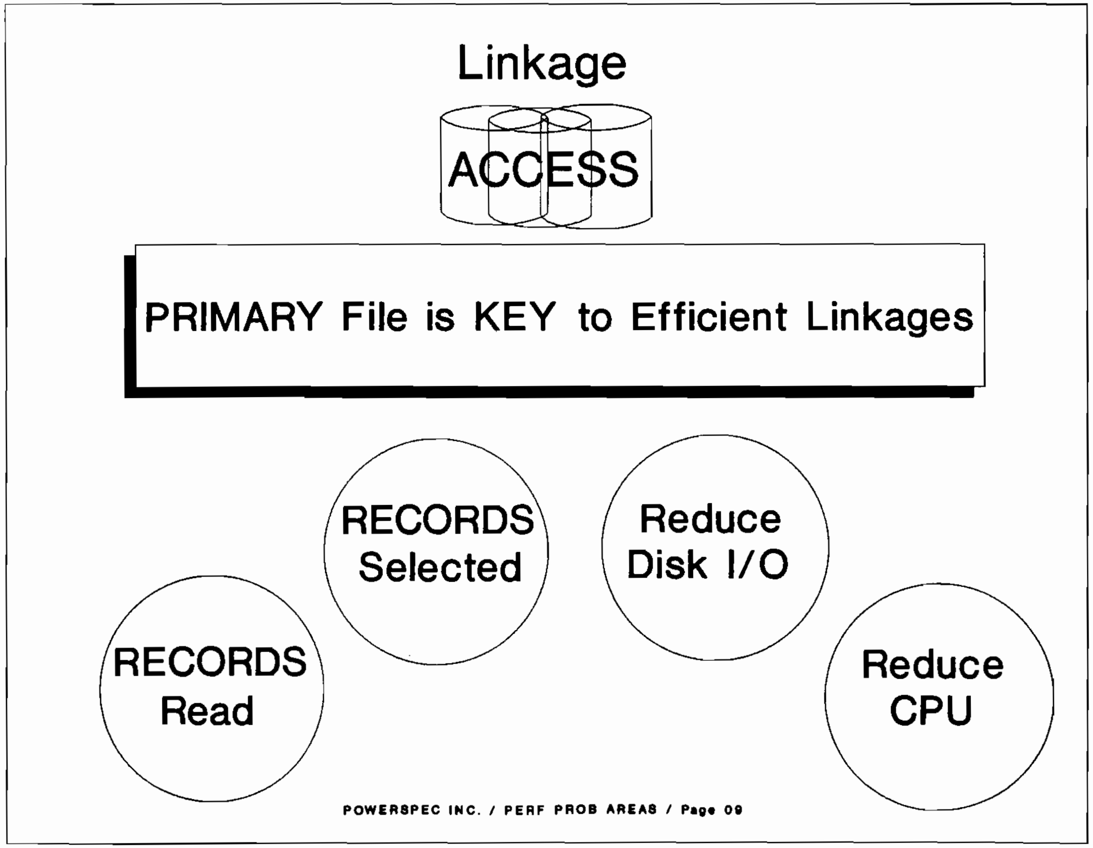
Record Blocking

Buffer Specifications

Keys



Data Structures: A Key to Performance



Linkage

ACCESS

PRIMARY File is KEY to Efficient Linkages

RECORDS Read

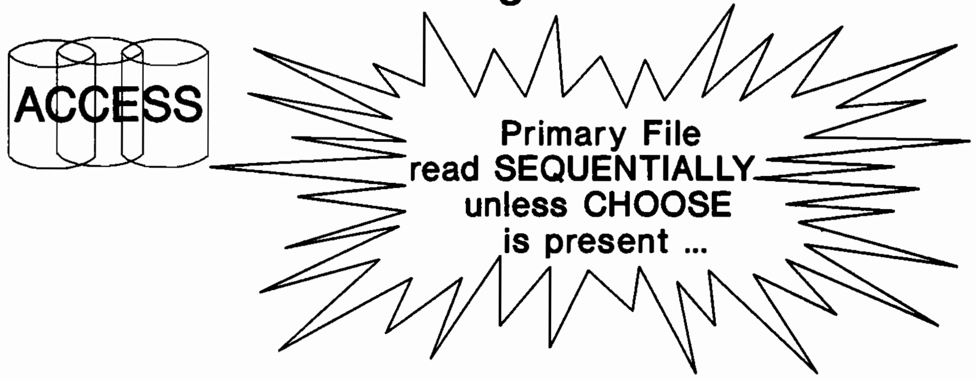
RECORDS Selected

Reduce Disk I/O

Reduce CPU

POWERSPEC INC. / PERF PROB AREAS / Page 09

Linkage



Primary File ➡

- ✓ Selection Criteria
- ✓ Capacity of File(s)
- ✓ Report Specifications
- ✓ Sequence of Report

ACCESS

Linkage BENCHMARK

SELECT all payments of pay-type ="G"

CLAIMS

CLAIM#
NAME
45,000

```
;Inefficient Specification
>ACCESS claims LINK TO payments
>SELECT IF pay-type of payments = "G"
>.....
```

CPU/386   Elapsed MIN/4

PAYMENTS

CLAIM#
PAY-TYPE
7,000

```
;Efficient Specification
>ACCESS payments LINK TO claims
>SELECT payments IF pay-type = "G"
>.....
```

CPU/18   Elapsed MIN/1

Linkage

- Understand the Types of Linkages
  - HIERARCHICAL Linkage
  - PARALLEL Linkage
  - RELATIONAL Linkage
- Linkage Options
  - OPTIONAL
  - IF [NOT] RECORD file EXISTS

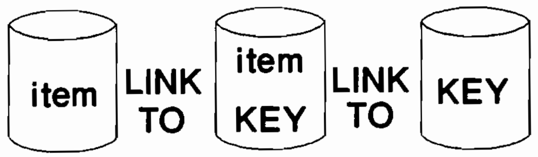
ACCESS

Linkage

- ■ HIERARCHICAL Linkage

›ACCESS file1 LINK item TO key OF file2 ...
- Implied relationship to preceeding File

›ACCESS file1 LINK (item1,item2 ...) TO item1, item2 ...
- Segmented Keys

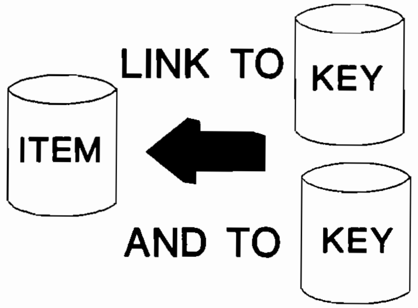


Record Complex/Transaction
Record from each File

Linkage

■ PARALLEL Linkage

›ACCESS file1 LINK item TO key OF file2 AND TO ...
- Linkage is in Parallel
- Implied OPTIONAL in Record Complex/Transaction

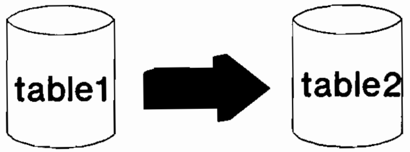


Linkage

- RELATIONAL Linkage

›ACCESS table1 IN rel d/b LINK TO table2 IN rel d/b
- Linkage of views/tables
- More flexiblity in Linkages
- Multiple Views/Tables

COGNOS, Inc.



STARBASE

Linkage

■ Linkage Options

- OPTIONAL
  - Allows partial Record Complex

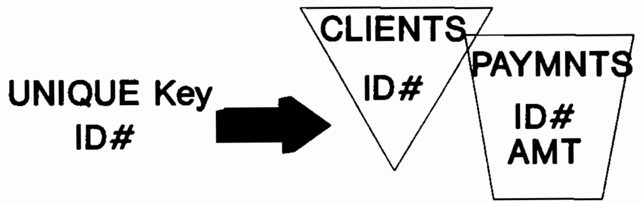
```
>ACCESS file1 LINK TO file2 OPTIONAL
```

- IF [NOT] RECORD file EXISTS
  - Test for partial Record Complex

```
>SELECT IF NOT RECORD file EXISTS
>DEFINE cnt=1 IF RECORD file EXISTS
```

Linkage

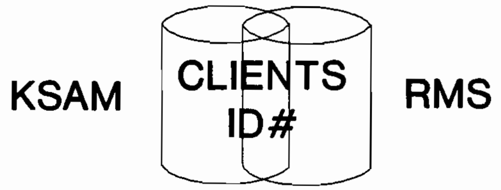
- Linkage Sequence May Eliminate Sort
  - Files with UNIQUE Keys
    - HP/IMAGE Masters



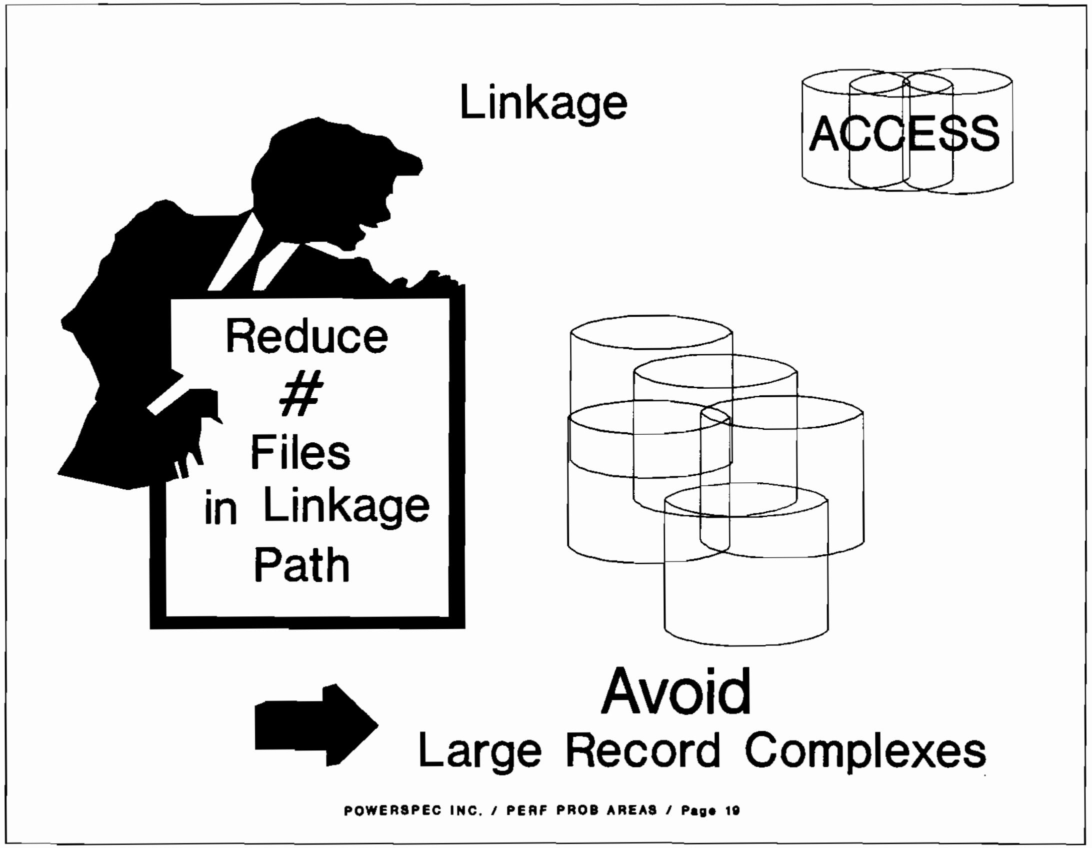
```
>ACCESS clients LINK ID# TO ID# OF paymnts
>SORTED ON ID# OF clients
>SET SUBFILE AT ID#
>REPORT SUMMARY ID# amt SUBTOTAL ...
```

Linkage

- Linkage Sequence May Eliminate Sort
  - Indexed Files
  - Key Ascending Sequence Retrieval



```
>ACCESS clients LINK TO ... LINK TO ...
>CHOOSE ID#                          ;No Value Specified
>SORTED ON ID#
>FOOTING AT ID# ...
```



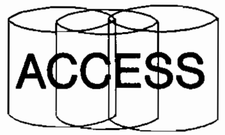
Linkage

ACCESS

Reduce
#
Files
in Linkage
Path

Avoid
Large Record Complexes

Linkage



Qualify Linkages

Speeds up Linkage Process
Avoid wrong QUIZ/QTP assumptions
Added Documentation
Better for Program Maintenance

PRIMARY File is a KEY to Performance !

Data Structures

Linkage

Record Selection

Expressions

Sorting

RECORD SELECTION

CHOOSE → KEY Access

CHOOSE
Primary File only

SELECT → Sequential Access
- All RECORDS Read

RECORD SELECTION

SELECT IF

SELECT

- AND conditions
  - Specify "least likely condition " FIRST
- OR conditions
  - Specify "most likely condition" FIRST
- INDEX function
  - More Efficient than multiple "ORs"

```
SELECT IF 0 NE INDEX("CA,NY,AZ,LA",state)

SELECT IF 0 NE INDEX("213,415",ASCII(phone,10)[1:3])
```

RECORD SELECTION

SELECT file IF

For Multiple File Linkages

Fewer I/Os

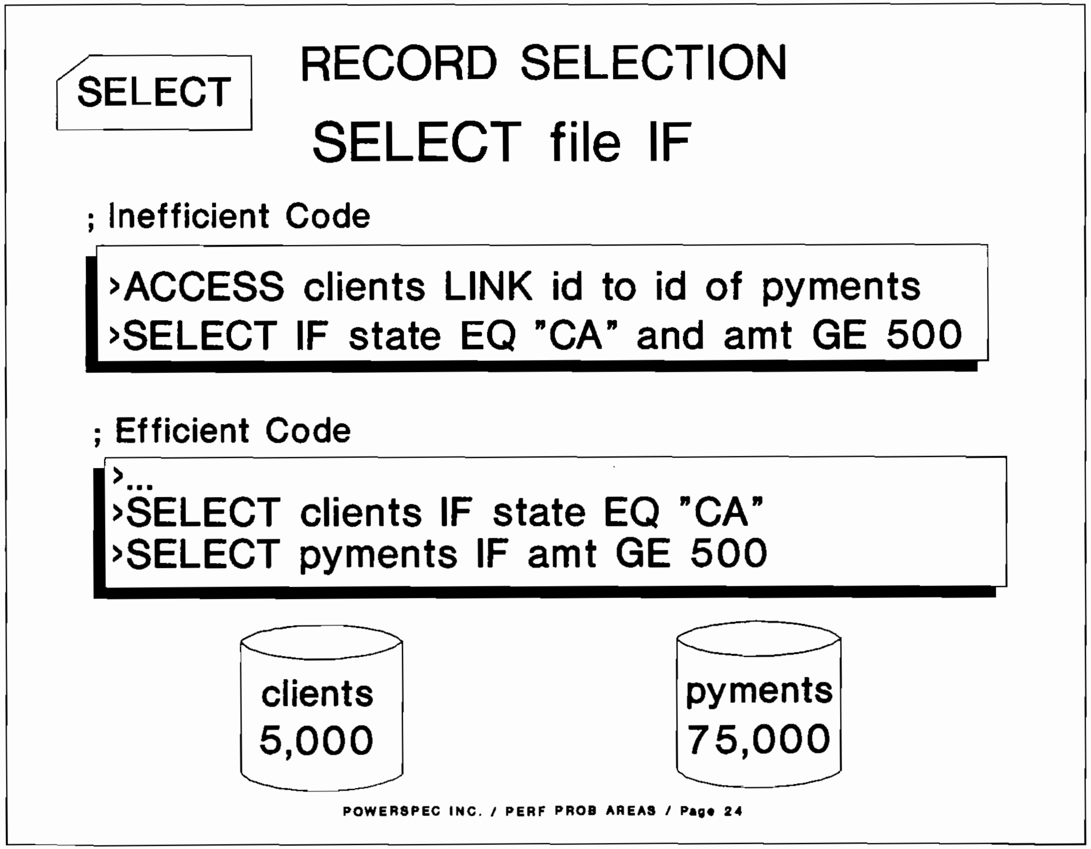
Less CPU Usage

More Efficient than SELECT IF

SELECT IF = Rejects Record Complex

SELECT file IF = Rejects Record

SELECT


SELECT

RECORD SELECTION

SELECT file IF

; Inefficient Code

```
>ACCESS clients LINK id to id of pyments
>SELECT IF state EQ "CA" and amt GE 500
```

; Efficient Code

```
>...
>SELECT clients IF state EQ "CA"
>SELECT pyments IF amt GE 500
```

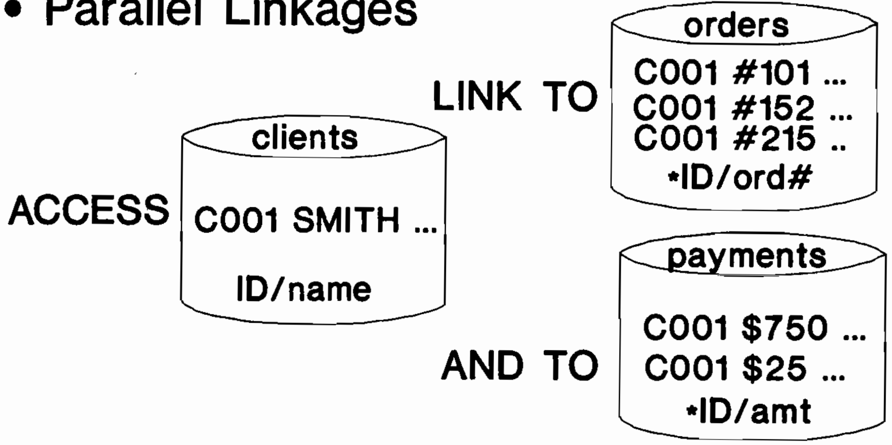
clients
5,000
pyments
75,000

SELECT

RECORD SELECTION

SELECT file IF

- Parallel Linkages



>SELECT payments IF amt GE 500

How many Record Complexes/Transactions ?

SELECT

RECORD SELECTION

Parallel Linkages

;SELECT file IF

```
>ACCESS clients LINK to orders AND to payments
>SELECT payments IF amt GE 500
```

```
          C001 SMITH ... C001 #101 ... C001 $750 ...
Results = C001 SMITH ... C001 #152 ...   no record
          C001 SMITH ... C001 #215 ...   no record
```

;SELECT IF

```
>ACCESS clients LINK to orders AND to payments
>SELECT IF amt GE 500
```

```
Results = C001 SMITH ... C001 #101 ... C001 $750
```

SELECT

RECORD SELECTION

- OPTIONAL Linkages

;Inefficient Code

```
>ACCESS clients LINK to payments OPTIONAL
>SELECT IF amt GE 500
```

- Test for Partial Record Complex

;Efficient Code

```
>ACCESS clients LINK to payments OPTIONAL
>SELECT IF RECORD payments EXISTS &
>        AND amt GE 500
```

Data Structures

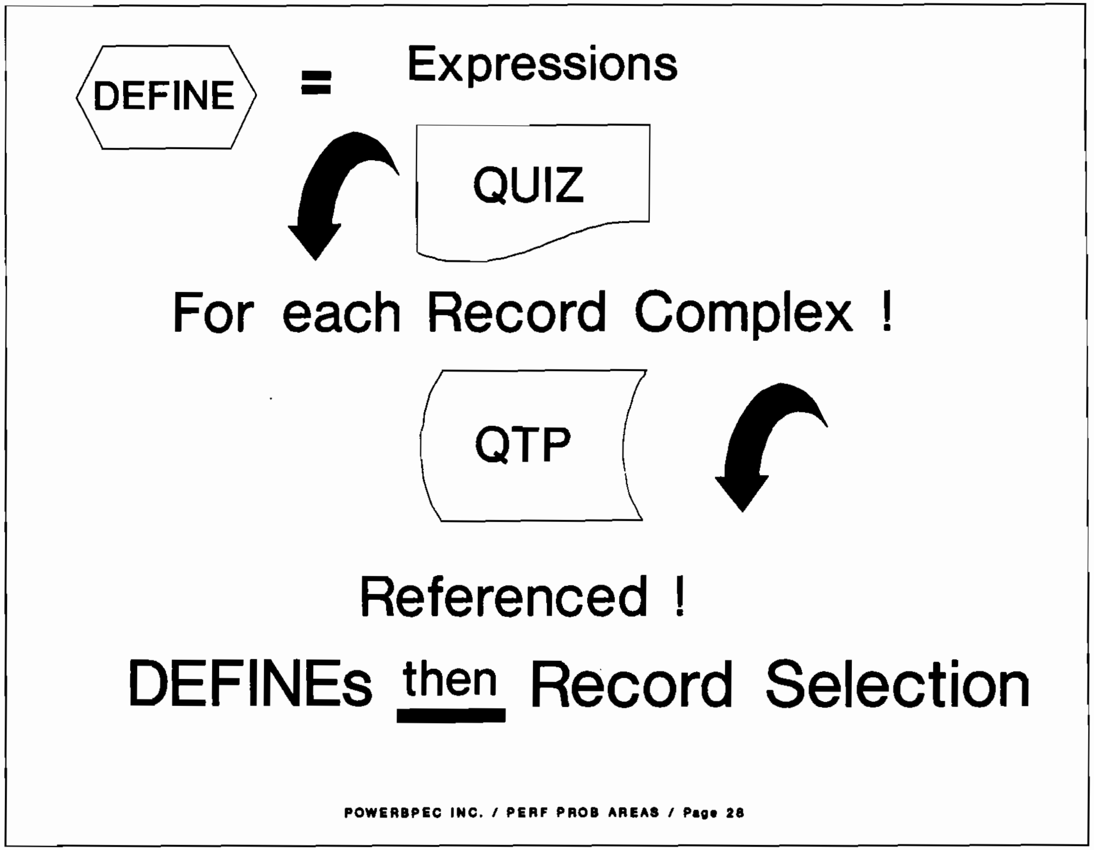
Linkage

Record Selection

Expressions

Sorting

3107-27A



DEFINE = Expressions

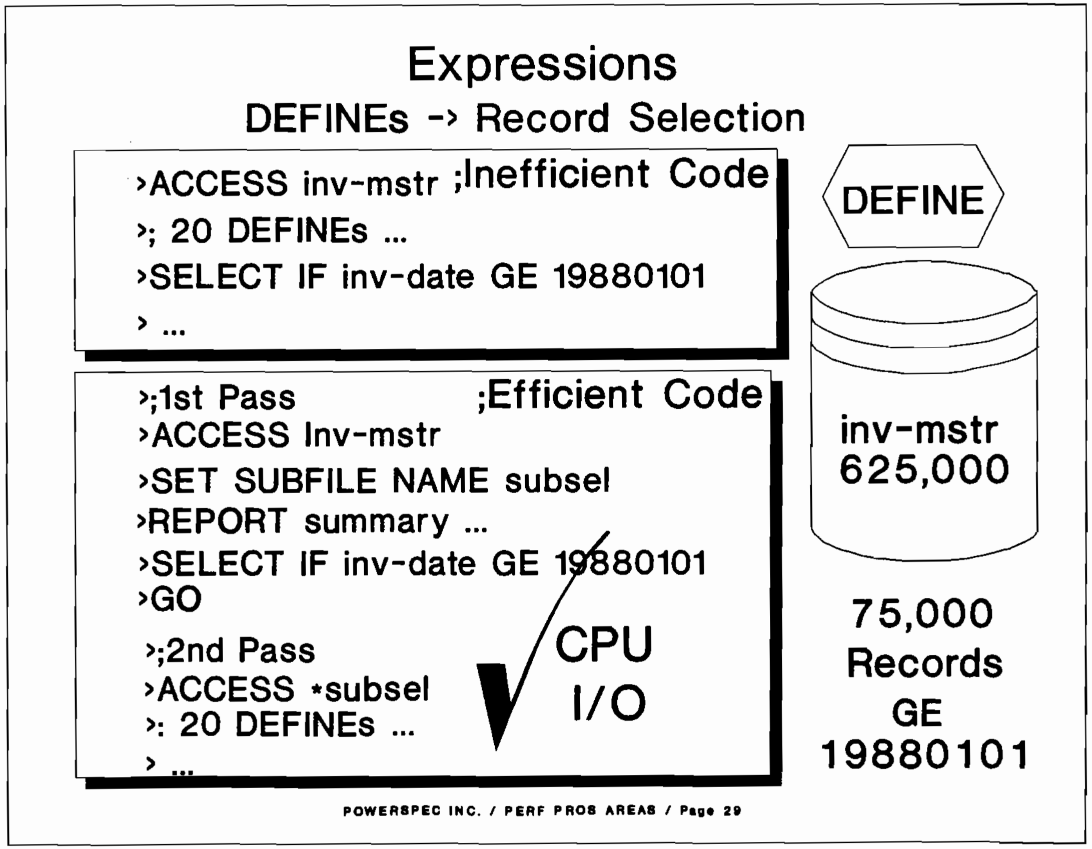
QUIZ

For each Record Complex !

QTP

Referenced !

DEFINEs <u>then</u> Record Selection



Expressions

DEFINEs -> Record Selection

```
>ACCESS inv-mstr ;Inefficient Code
>; 20 DEFINEs ...
>SELECT IF inv-date GE 19880101
> ...
```

```
>;1st Pass                ;Efficient Code
>ACCESS Inv-mstr
>SET SUBFILE NAME subsel
>REPORT summary ...
>SELECT IF inv-date GE 19880101
>GO
>;2nd Pass
>ACCESS *subsel
>: 20 DEFINEs ...
> ...
```

✓ CPU
I/O
DEFINE
inv-mstr
625,000
75,000
Records
GE
19880101

Expressions

DEFINE

IF ... ELSE vs CASE Processing

;Inefficient Code

```
>DEFINE region-code ZONED *2            = 51 &
>  IF city-code = 30                 ELSE 52 &
>  IF city-code = 31                 ELSE 53 &
>  IF city-code = 32 or city-code = 33 or city-code = 34
```

;Efficient Code

```
>DEFINE region-code ZONED *2 = CASE of city-code &
>   WHEN 30        : 51 &
>   WHEN 31        : 52 &
>   WHEN 32 to 34 : 53
```

Expressions

DEFINE

>;Inefficient Code

```
>DEFINE branch-nme CHAR *20 = &
>   CASE of branch-code
>WHEN "01"  : "MEMPHIS"        &
>WHEN "04"  : "TUCSON"         &
>WHEN "03"  : "ALBERQUERQUE" &
>.....
>SORT ON branch-nme
```

Add a DEFINE !

;Efficient Code

```
>DEFINE sort-nme CHAR *5 = branch-nme[1:5]
>SORT on sort-nme
```

DEFINE

Expressions

QTP

;Inefficient Code

```
>ACCESS ...                    ; Input Phase
>DEFINE x  CHAR *2 = acctno[8:2]
>DEFINE y  ZONE *2 = 01 IF x = "99" AND ...
>DEFINE z  ZONE *2 = 45 IF x NE "AA"  &
>    AND y NE "99"
```

BENCHMARK

13,353 Records / CPU SEC = 2050

QTP

X and Y are Re-Evaluated for Z ...

DEFINE

Expressions

QTP

;Efficient Code

```
>ACCESS ...                     ; Input Phase
>TEMP   x  CHAR *2              ; Output Phase
>  ITEM x = acctno[8:2]
>TEMP   z  ZONE *2
>  ITEM y = 01 IF x = "99" AND ...
>TEMP z  ZONE *2
>  ITEM z = 45 IF x NE "AA" AND y NE "99"
```

BENCHMARK

13,353 Records / CPU SEC = 250

QTP

TEMP/ITEM instead of DEFINE

DEFINE

Expressions

- Re-Define ITEMS in Dictionary

vs

- DEFINE items
    - SUBSTRING [n:m]
    - ASCII
    - NCONVERT

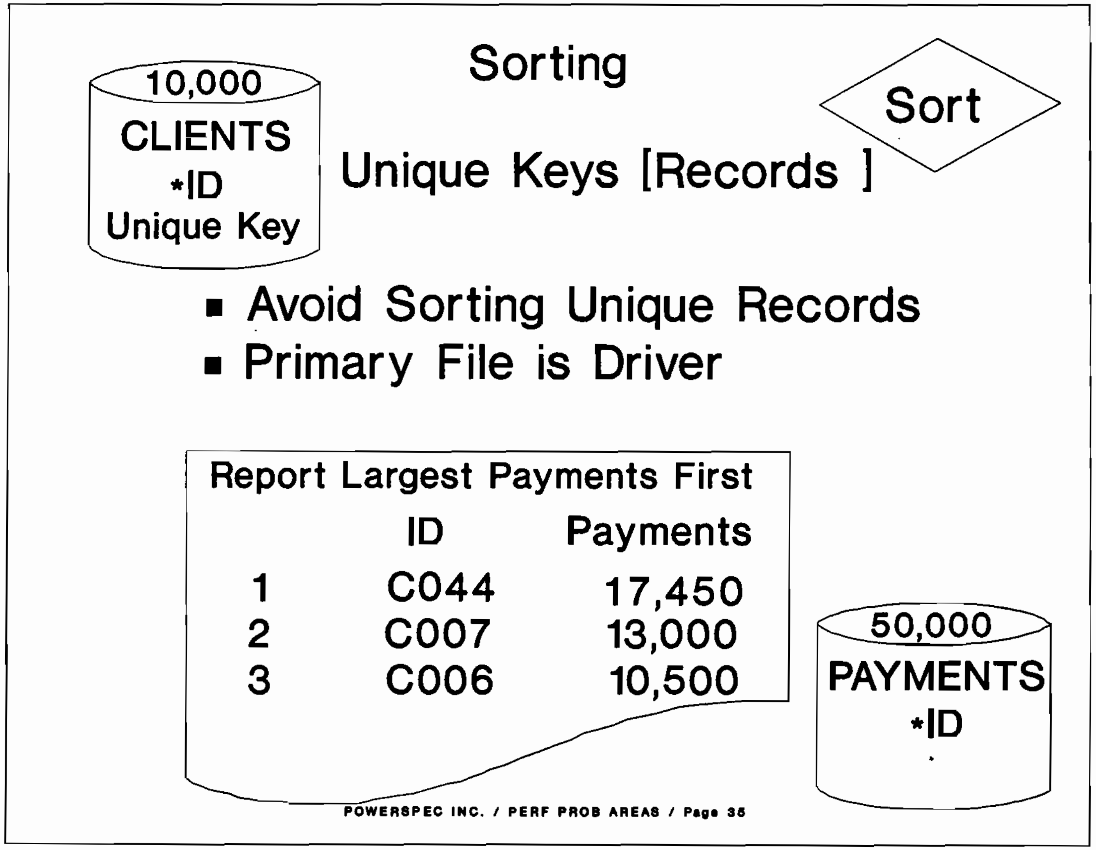
Data Structures

Linkage

Record Selection

Expressions

➡ Sorting



Sorting

10,000
CLIENTS
*ID
Unique Key

Sort

Unique Keys [Records]

- Avoid Sorting Unique Records
- Primary File is Driver

Report Largest Payments First

	ID	Payments
1	C044	17,450
2	C007	13,000
3	C006	10,500

50,000
PAYMENTS
*ID

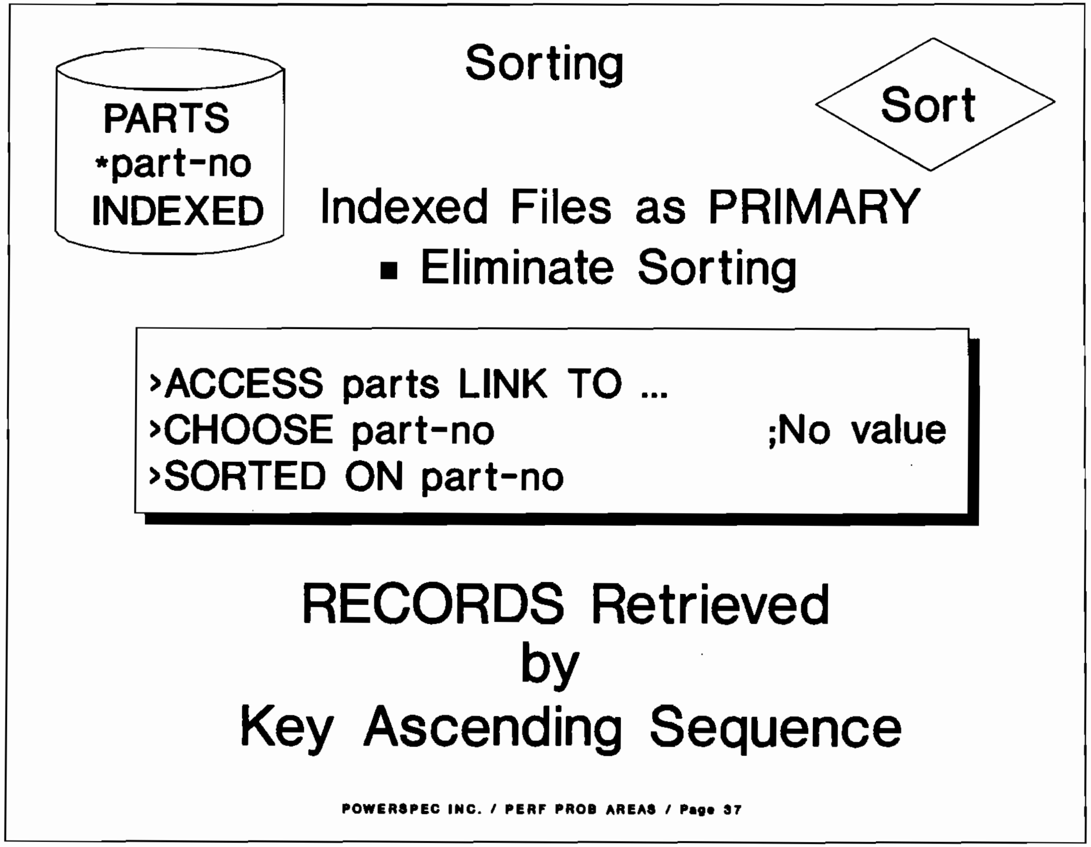
Sorting

SORT

SORT vs SORTED / Unique Records

```
>ACCESS clients LINK to payments          ;1st Pass
>DEFINE pyment-tot INT *9 = amt of payments
>SORTED ON ID of clients
>SET SUBFILE AT ID NAME subtot
>REPORT SUMMARY ID pyment-tot SUBTOTAL
>GO
```

```
>ACCESS *subtot                           ;2nd Pass
>SORT ON pyment-tot D
>REPORT COUNT NORESET ID pyment-tot
>GO
```



Sorting

PARTS
*part-no
INDEXED

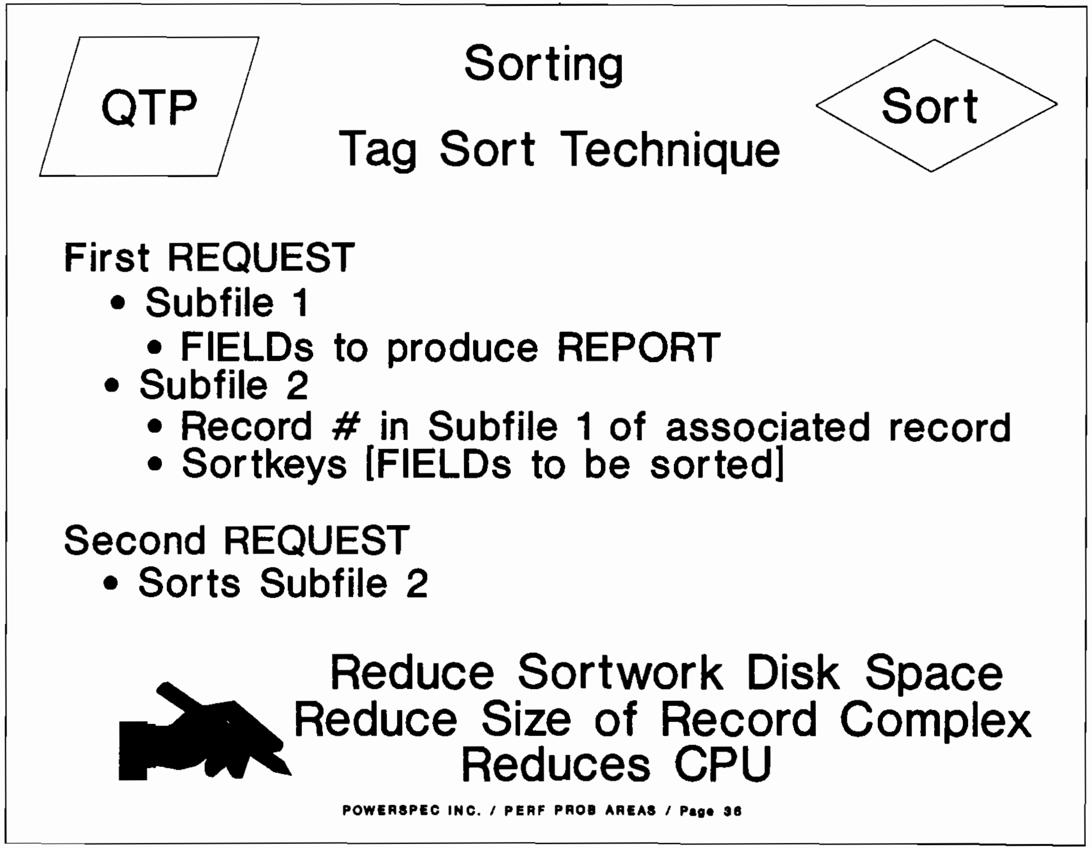
Sort

Indexed Files as PRIMARY

- Eliminate Sorting

```
>ACCESS parts LINK TO ...
>CHOOSE part-no                ;No value
>SORTED ON part-no
```

RECORDS Retrieved
by
Key Ascending Sequence



QTP

Sorting

Tag Sort Technique

Sort

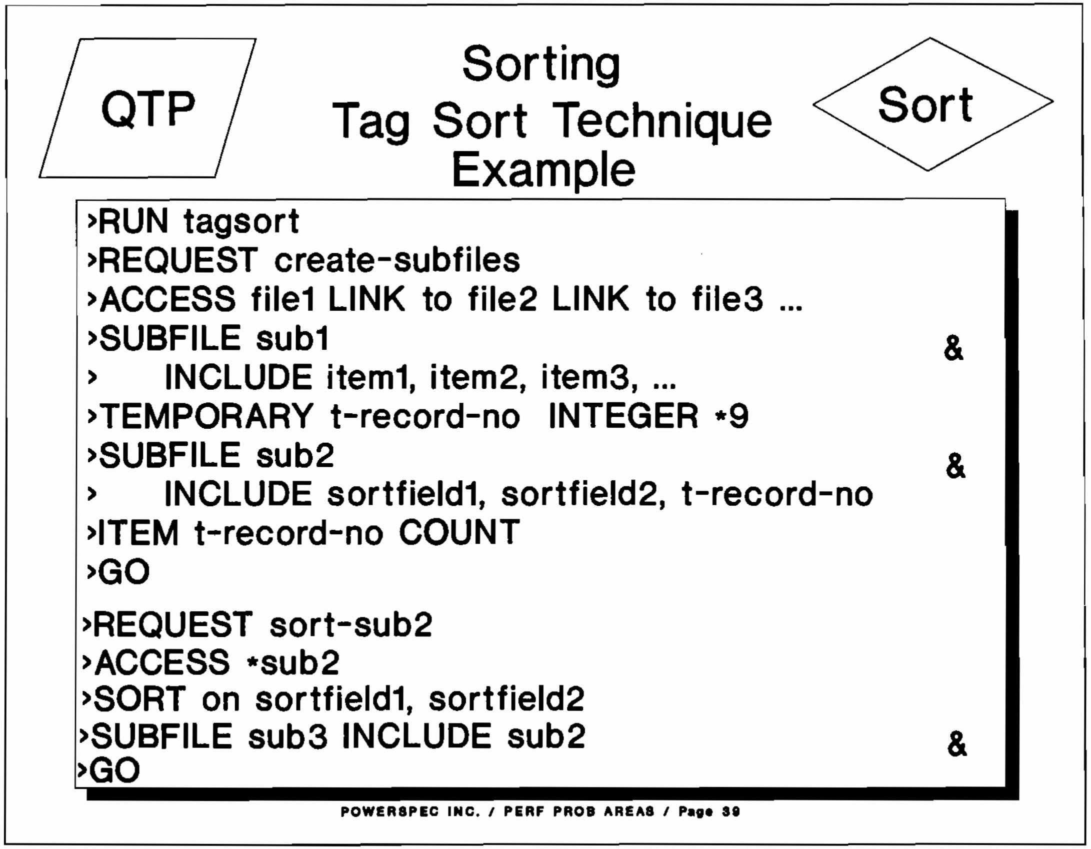
First REQUEST

- Subfile 1
  - FIELDs to produce REPORT
- Subfile 2
  - Record # in Subfile 1 of associated record
  - Sortkeys [FIELDs to be sorted]

Second REQUEST

- Sorts Subfile 2

Reduce Sortwork Disk Space
Reduce Size of Record Complex
Reduces CPU



QTP

Sorting
Tag Sort Technique
Example

Sort

```
>RUN tagsort
>REQUEST create-subfiles
>ACCESS file1 LINK to file2 LINK to file3 ...
>SUBFILE sub1                                          &
>    INCLUDE item1, item2, item3, ...
>TEMPORARY t-record-no  INTEGER *9
>SUBFILE sub2                                          &
>    INCLUDE sortfield1, sortfield2, t-record-no
>ITEM t-record-no COUNT
>GO

>REQUEST sort-sub2
>ACCESS *sub2
>SORT on sortfield1, sortfield2
>SUBFILE sub3 INCLUDE sub2                             &
>GO
```

QUIZ

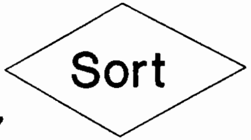
Sorting
Tag Sort Technique
Example

Sort

```
>ACCESS *sub3                                             &
>    LINK TO RECORD(t-record-no of sub3) of *sub1
>SORTED ON sortfield2, sortfield2
>REPORT ...
>PAGE HEADING ...
>...
>GO
```

- Sorted Subfile as Primary File
- Linkage by Record Expression
  - Record# Tagged at End of File
- SORTED instead of SORT

Sorting



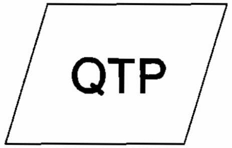
QTP: Front-End to QUIZ

QTP

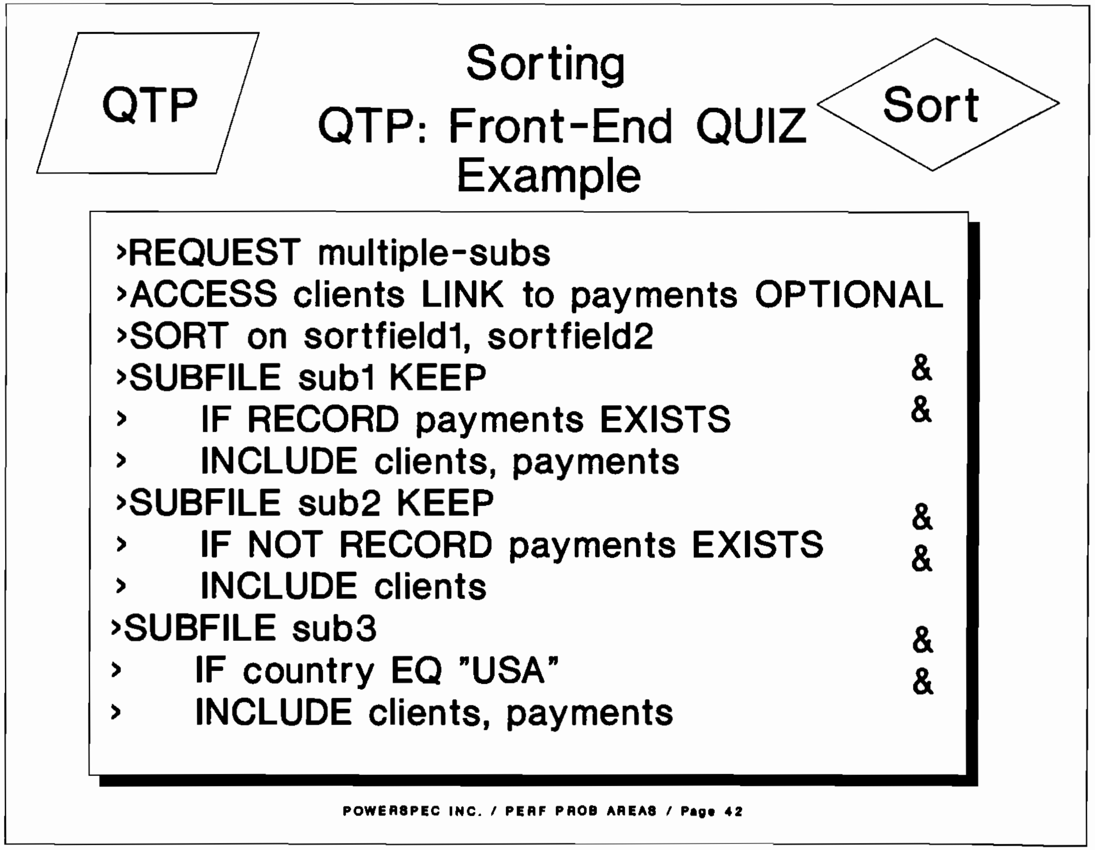
- Extract Multiple Subfiles in One Pass
  - Conditionally
  - Sort Multiple Subfiles

QUIZ

- Format Each Subfile into REPORT



- Reduces CPU
- Reduces I/O



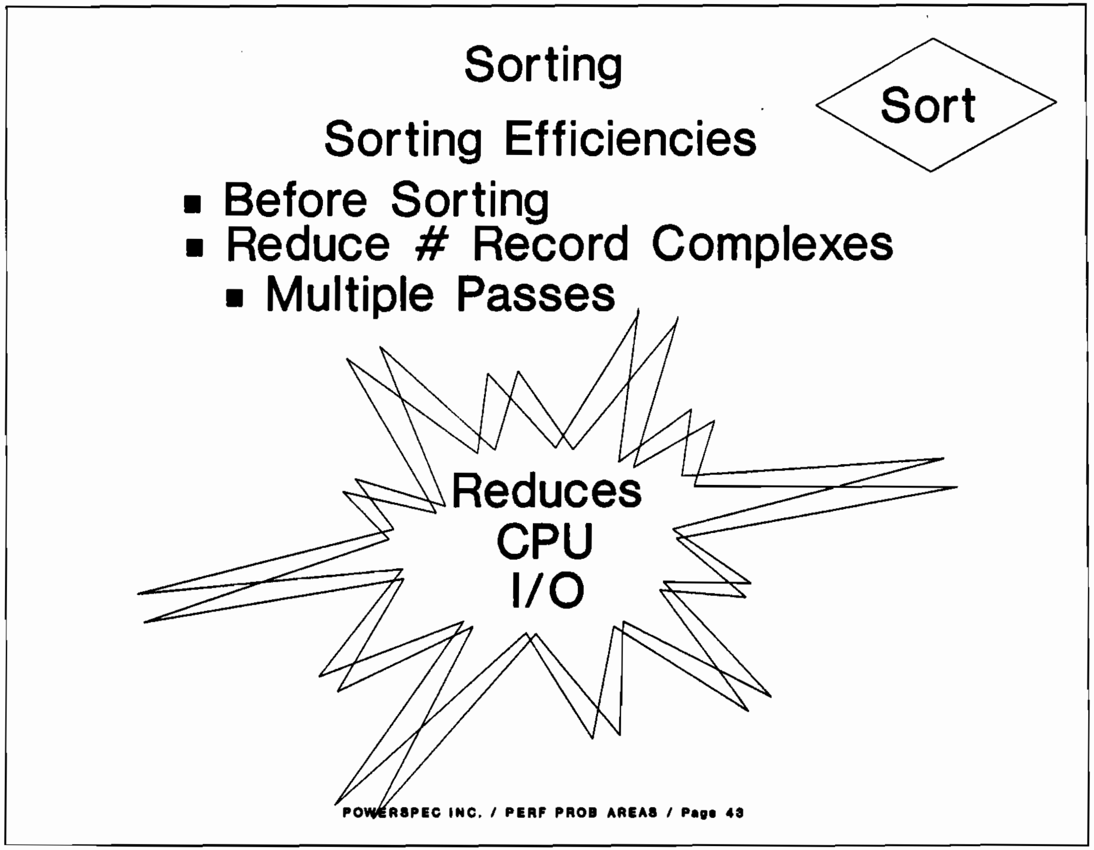
QTP

Sorting
QTP: Front-End QUIZ Example

Sort

```
>REQUEST multiple-subs
>ACCESS clients LINK to payments OPTIONAL
>SORT on sortfield1, sortfield2
>SUBFILE sub1 KEEP                                &
>     IF RECORD payments EXISTS                   &
>     INCLUDE clients, payments
>SUBFILE sub2 KEEP                                &
>     IF NOT RECORD payments EXISTS               &
>     INCLUDE clients
>SUBFILE sub3                                     &
>     IF country EQ "USA"                         &
>     INCLUDE clients, payments
```

POWERSPEC INC. / PERF PROB AREAS / Page 42



Sorting

Sort

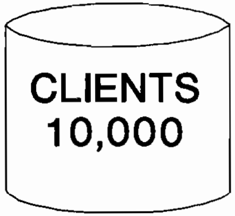
Sorting Efficiencies

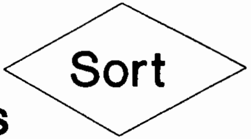
- Before Sorting
- Reduce # Record Complexes
  - Multiple Passes

Reduces
CPU
I/O

Sorting

Without Using Subfiles ;Inefficient Code

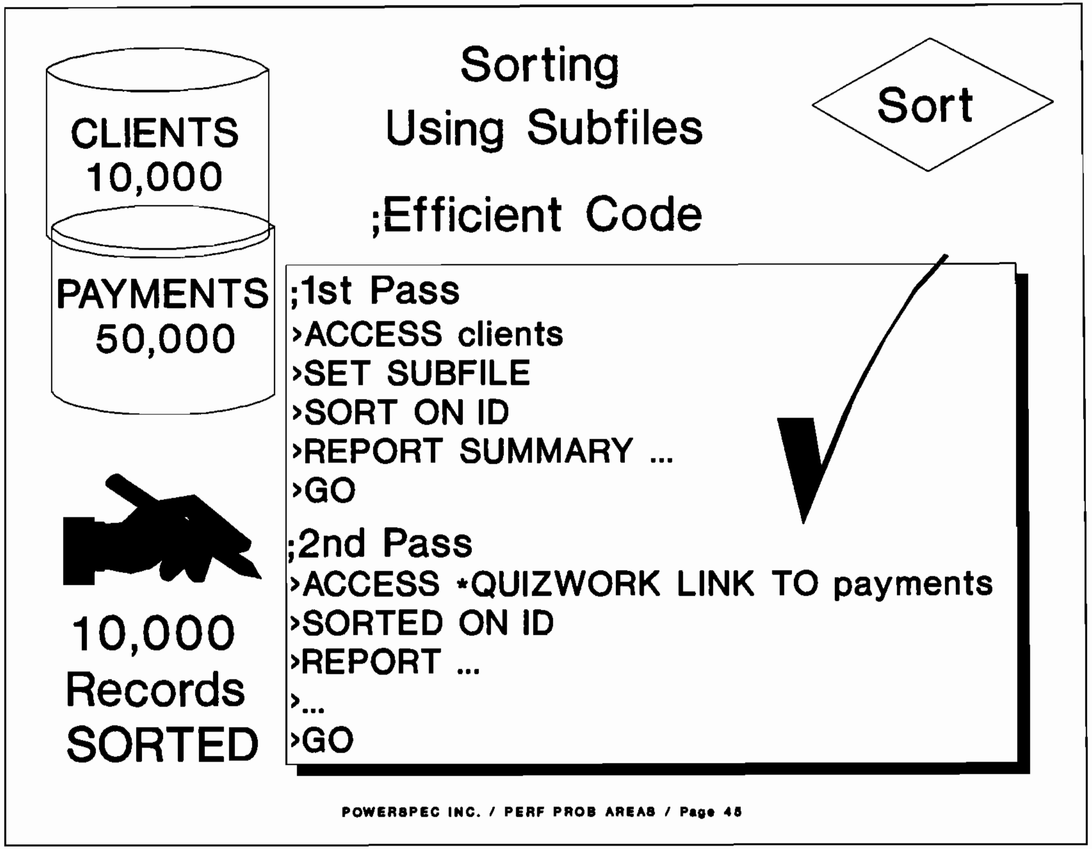




```
>ACCESS clients LINK to payments
>SORT ON ID of clients
>REPORT ...
>...
>GO
```

50,000 Record Complexes
SORTED

PAYMENTS
50,000



Sorting
Using Subfiles

Sort

CLIENTS
10,000

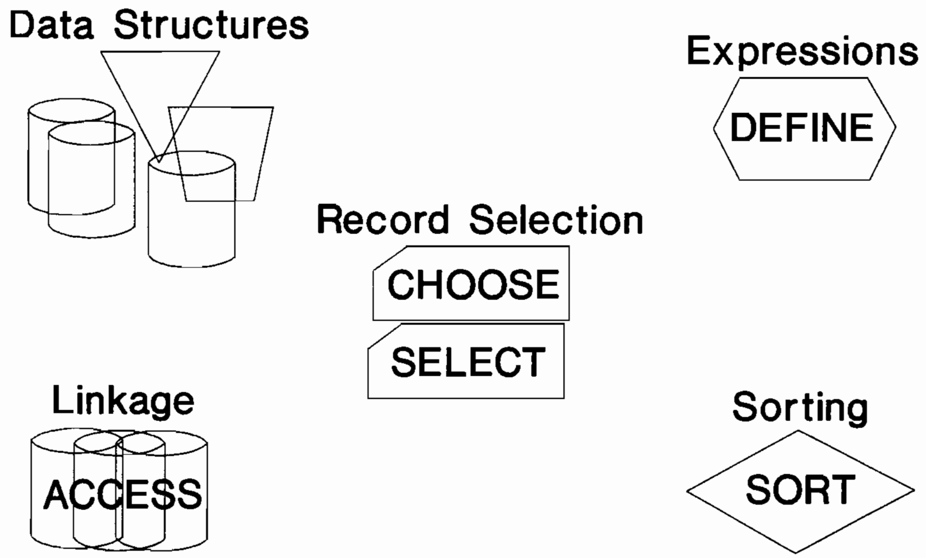
PAYMENTS
50,000

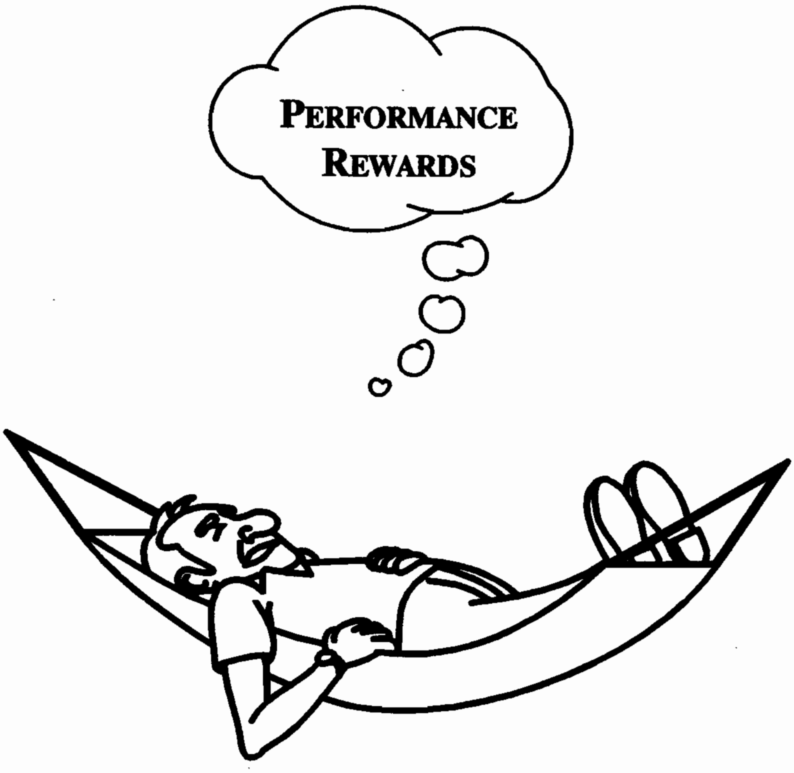
;Efficient Code

```
;1st Pass
>ACCESS clients
>SET SUBFILE
>SORT ON ID
>REPORT SUMMARY ...
>GO

;2nd Pass
>ACCESS *QUIZWORK LINK TO payments
>SORTED ON ID
>REPORT ...
>...
>GO
```

10,000
Records
SORTED

The Problem Areas





3107-47

TITLE:	Real World Performance Monitoring
AUTHOR:	Robert Lund
	Robert Lund & Associates
	34130 Parkwoods Dr. N.E.
	Albany, OR 97321

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 3108

TITLE: SQL and the Relational Data Model for the IMAGE Database

AUTHOR: Mark da Silva
University of Hawaii Cancer Center
1236 Lauhala Street #402
Honolulu, HI 96813
808-521-0054

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 3110

SECURING YOUR HP 3000

Dennis King
Mariette Araya
Hewlett-Packard
Corporate Offices
P. O. Box 10301
Palo Alto, CA 94303

If security is a state of mind, what state of mind are you in?

Alas, most security officers in charge of computer security probably have a false sense of security. For all the good intentions, long hours spent collecting gems of knowledge, attending seminars, and implementing security tools, most computers develop security weaknesses, or "holes", which put them at risk. Such risks can be unknown or overlooked by the unsuspecting security officer, who feels secure because he has "done all he can do." But has he? The answer can be found by careful examination of the one characteristic that distinguishes secure data centers from un-secure ones: an effective computer security plan.

Most MIS departments have a security plan in place, but very few of them have been effective in closing most of the known "holes" in system security. Why? Because most plans omit one or more of these four main tenets of implementing an effective computer security plan:

(1) Developing and continually updating a comprehensive written plan
(2) Obtaining full management support of your security plan
(3) Implementing cost-effective tools and job streams to automate the security process
(4) Ensuring user understanding, acceptance and support of the security measures in place

If any one of these four rules breaks down, you've lost the battle.

This paper focuses primarily on the first of these tenets: developing a comprehensive security plan. In this paper we will share an established and proven security plan written for the HP 3000. Specifically, this paper will concentrate on the following areas:

(A) System Management Security Procedures
(B) Access Management
(C) Access Monitoring
(D) Capability Controls
(E) Password Management
(F) Procedures for Compliance

Areas not addressed in this paper are: physical security, network security, PC security, disaster recovery, and system backups. Nevertheless, these topics must be addressed in your company's security plan. Appendix C contains a bibliography which should help your company establish standards for those topics that aren't covered in this paper, in addition to those that are.

It would be nice if all MIS departments aggressively improved security solely for the purpose of protecting data. The bottom line within many MIS departments, however, is to satisfy audit security requirements. This is understandable, since security breaches (many of which are never known) and security exposures rarely affect career paths like an audit "failure" would. Our paper is written from the viewpoint of satisfying both data security needs and satisfying audit requirements. Adopting a security plan like the one presented here reaps additional benefits for companies with distributed MIS departments. A standard plan can communicate 'best practices' and prevent duplication of efforts among distributed MIS personnel.

Clearly any long-term security plan will need to incorporate some HP or third-party tools to automate many of the security checks. Such automated tools are cost-effective and aren't prone to human error. These specific security tools will not be identified in this paper. However, because of the importance of showing customers how they can use security packages they may already have purchased, we have asked the suppliers of the major security packages to describe how their products will supplement the security plan presented in this paper. Their responses will be handed out during the presentation.

The following is our security plan, which contains the actual text of established standards. Our intent is for you, the reader, to be able to use this document as a model for your own HP 3000 security plan. You will need to change some of the wording to fit your company's vernacular, and you may need to add specific security items relating to your site. We should state that this plan does not make 'light' reading, but this is natural -- because the following is not an essay, but an established security plan. We've added a collection of 'best practices' (in italics) immediately after each standard as a guide to implementing them:

A. HP 3000 SYSTEM MANAGEMENT SECURITY PROCEDURES

1. Establishing A Local HP 3000 Security Plan

Each MIS department must develop a plan for securing its HP 3000 systems that address the following issues:

o Access Management - The MIS department must address how specific individuals will be granted access to sensitive information resources on its HP 3000 systems. In addition, it must determine what controls will be put in place to protect against unauthorized access to sensitive information resources.

o Access Monitoring - The MIS department must establish a procedure for regular reviews of the system log files in order to identify irregular activities.

o Capability Controls - The MIS department must define a policy on how special MPE capabilities will be managed.

o Password Management - The MIS department must create an effective password management policy.

o Procedures for Compliance - The MIS department must conduct regular self-audits and ensure that all users comply with the security plan.

Best Practices

o *All levels of management must be committed to the successful implementation of the security plan. Without this commitment, your security plan will fail because it will lack high priority or funding.*

o *The HP 3000 security plan should be reviewed and updated annually in order to adapt to changes in the operating system, new software, new hardware, etc. If your company does not have a plan, use the one presented here as your starting point.*

o *Stay informed of new technologies and security issues by reading related articles from HP's Communicator*, INTERACT, SuperGroup, and The HP Chronicle.

2. Establishing Local Self-Audit Procedures

HP 3000 self-audit procedures must be developed and approved by MIS management. These procedures must be included in the computer security plan and must be communicated to all appropriate system users.

Best Practices

- *Ensure that your user community understands and agrees with the security policy. Communicate only those portions of the security plan that pertain to the user. Don't send your users the entire plan, as they will probably not read it.*

- *Investigate available HP and third-party security packages or programs which can be used to automate your security process. Most of these packages can be justified on the basis of savings in resources and management time. In addition, ask your auditors to provide you with tools and procedures for securing your HP 3000.*

- *Keep an up-to-date security check-list available for your System Managers. This check-list should be a quick-reference summary of your established security practices.*

3. Establishing Local Responsibilities

Each MIS department must identify a person who will be responsible for ensuring that the individual MIS department complies with its own security plan.

Best Practices

- *Assign one person and a backup to the job of maintaining your security plan. This job can rarely be consistent and completely controlled among a group of individuals. This practice will also reduce the diffusion of responsibility and the finger-pointing so often encountered by auditors. At least the fingers will all point in the same direction. (No, not up.)*

- *In addition to being knowledgeable, the individual chosen must possess strong communication and negotiation skills. Your security plan will not succeed in an uncooperative environment.*

4. Establishing Self-Audit Frequency

Self-Audit procedures must be conducted at regular intervals. It will be the MIS department management's responsibility to determine this interval and to provide a written justification why the interval chosen provides adequate security for the site.

Best Practices

- *Conduct a comprehensive self-audit, as defined in the MIS security plan, on all HP 3000s at least every three months. In addition, a self-audit should be conducted after every operating system update or major application release. Machines requiring tighter security (e.g., the Payroll or Finance machines) should have self-audits performed every month.*

B. ACCESS MANAGEMENT

1. Granting Access to HP 3000 Information Resources

The MIS department must establish procedures to determine which individuals will be granted access to sensitive information resources residing on its HP 3000 computers. A sensitive information resource, usually a file, is one whose unauthorized access or modification can negatively impact company business.

Normally, an MIS department will assign Account Managers to manage specific applications or sensitive files, and the Account Managers will set up mechanisms for granting access to specific individuals or classes of individuals.

Best Practices

o *Perform a risk assessment on the costs of unauthorized disclosure or alteration of files and applications considered sensitive. Ideally, this is done during Disaster Recovery planning.*

o *Develop or purchase a port security tool to disallow special capability logons for DIAL-IN, PAD, X.25 and LAN accessors. Usually, this will involve a LOGON UDC.*

o *Users requesting new accounts must be required to fill out an Account Request Form. This form, shown in Figure 1., must be archived, and should be easily retrievable by account name, system name, and the Account Manager's name. An MPE flat file or database is well-suited for archival.*

o *Require positive authentication of any caller requesting access to your system. If it is an HP support engineer asking for this connection, ask the engineer to give you the Reference Call ID number. Allow access only if the ID given matches a valid open call ID.*

o *Each MIS department should require individuals to use unique MPE users IDs, whenever possible. Individuals who are given their own user IDs are much more likely to be protective of them.*

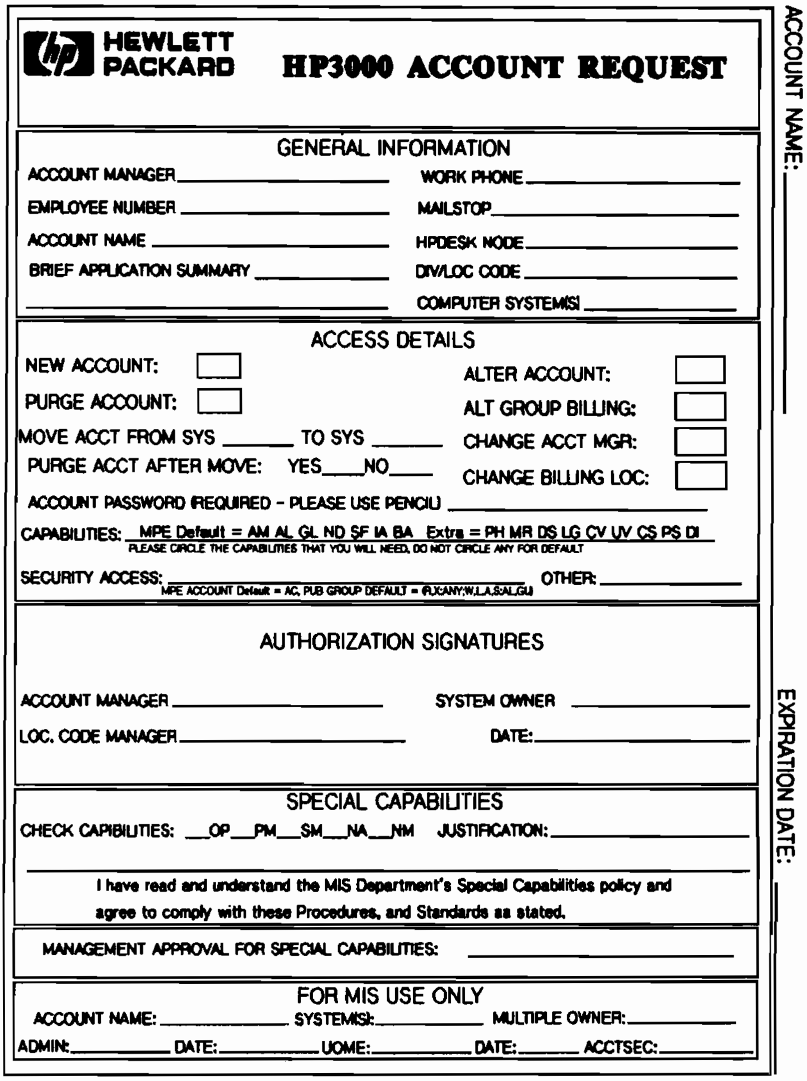
o *An unauthorized user who accesses your system (a most unfortunate event!) and doesn't log off immediately is trespassing. If the person knowingly accesses your system when instructed not to, it may be a crime. Add an appropriate welcome message. For example:*

"This is a private system operated for Company XYZ company business. Authorization from XYZ management is required to use this system. Use by unauthorized persons is prohibited."

o *If authentication of users is a very high priority at your site, read Miguel Cooper's October 1987 INTERACT article entitled "A Four-Phase Security System That Works."*

Figure 1.

SAMPLE ACCOUNT REQUEST FORM



HEWLETT PACKARD **HP3000 ACCOUNT REQUEST**

ACCOUNT NAME: ____________

GENERAL INFORMATION

ACCOUNT MANAGER ____________ WORK PHONE ____________

EMPLOYEE NUMBER ____________ MAILSTOP ____________

ACCOUNT NAME ____________ HPDESK NODE ____________

BRIEF APPLICATION SUMMARY ____________ DIV/LOC CODE ____________

____________ COMPUTER SYSTEM(S) ____________

ACCESS DETAILS

NEW ACCOUNT: ☐ ALTER ACCOUNT: ☐

PURGE ACCOUNT: ☐ ALT GROUP BILLING: ☐

MOVE ACCT FROM SYS _______ TO SYS _______ CHANGE ACCT MGR: ☐

PURGE ACCT AFTER MOVE: YES____NO____ CHANGE BILLING LOC: ☐

ACCOUNT PASSWORD (REQUIRED - PLEASE USE PENCIL) ____________

CAPABILITIES: MPE Default = AM AL GL ND SF IA BA Extra = PH MR DS LG CV UV CS PS DI
PLEASE CIRCLE THE CAPABILITIES THAT YOU WILL NEED, DO NOT CIRCLE ANY FOR DEFAULT

SECURITY ACCESS: ____________ OTHER: ____________
MPE ACCOUNT Default = AC, PUB GROUP DEFAULT = (R,X:ANY;W,L,A,S:AL,GU)

AUTHORIZATION SIGNATURES

ACCOUNT MANAGER ____________ SYSTEM OWNER ____________

LOC. CODE MANAGER ____________ DATE: ____________

EXPIRATION DATE: ____________

SPECIAL CAPABILITIES

CHECK CAPIBILITIES: __OP__PM__SM__NA__NM JUSTIFICATION: ____________

____

I have read and understand the MIS Department's Special Capabilities policy and agree to comply with these Procedures, and Standards as stated.

MANAGEMENT APPROVAL FOR SPECIAL CAPABILITIES: ____________

FOR MIS USE ONLY

ACCOUNT NAME: ____________ SYSTEM(S): ____________ MULTIPLE OWNER: ____________

ADMIN: ________ DATE: ________ UOME: ________ DATE: ______ ACCTSEC: ________

2. Controlling Access to HP 3000 Information Resources

The MIS department must establish procedures to control access to sensitive information resources so that only those individuals who are specifically authorized to access them may do so. In addition, each department will take appropriate measures to discourage unauthorized access to its HP 3000s.

This policy will normally involve having the Account Manager set up logons with passwords that are known only to specific individuals or classes of individuals, and establishing an account structure that limits access to these sensitive resources to the corresponding logons. This may need to be supplemented by additional controls, such as group passwords, file lockwords, database passwords, a front-end access control system, or additional access control logic within the application, in order to effectively deny access to unauthorized individuals.

When possible, access to sensitive resources should be controlled through identification and authentication of individuals, rather than having members of a class of users share a password.

The policy may optionally deny access to certain resources from particular sources (such as off-site X.25, off-site Internet, or dial-in).

__Best Practices__

o *Do not allow straight DIAL-IN access to your computer systems. Instead, use a DIAL BACK system. This will eliminate a majority of the threats posed by outside 'hackers'. In addition, using higher BAUD rate modems will reduce the threat posed by a large portion of hackers who can afford only 300 or 1200 BAUD modems.*

o *Change the MPE message catalog to give less aid to a hacker attempting to access the HP 3000. For example, when attempting to log on to a HP 3000 with improper syntax, the following message is displayed:*

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

  A much better (less informative) message would be:

  INVALID LOGON. CALL USER SUPPORT FOR ASSISTANCE

  Terry Simpkins' article entitled "HP 3000 Security - A Hacker's Perspective" covers this topic in great detail.

o *Configure all modem ports and data-switch ports to TYPE 16, , SUBTYPE 1 to ensure that sessions will terminate when the associated line is disconnected.*

o *:DOWN all modem ports until required. Most access attempts from the outside occur during off-hours.*

- *Investigate and implement access control features, such as call screening (offered by Public Data Network (PDN) vendors), if your systems are accessible from a PDN.*

- *The system itself is a sensitive resource. Ensure that only your operations staff and authorized support personnel can physically access your HP 3000s.*

- *Use a front-end port security package to verify that a given user is using an authorized port at the time of logon.*

3. MPE Access

The MIS department will determine whether access to the MPE Command Interpreter or to certain MPE commands should be restricted, and must establish and implement policies for granting or denying access to these facilities.

Best Practices

- *When appropriate, use LOGON, NOBREAK, NOHELP UDCs to prevent MPE access by those users who do not need it.*

  As stated by Michael Hornsby in his article "Is your HP 3000 data safe?", the most powerful thing you can do to tighten security on your system is to use menus to restrict access to specific application services. Oddly enough, the most common reason given for not using menus is that certain users need to be able to perform file transfers. But this is precisely the type of activity you want to maintain the tightest control over. In this case, programmatically perform the file transfer using PH capability and log the user and the file being accessed.

- *When disallowing MPE access, be aware that many subsystems and programs allow MPE access. Users who have access to them have access to MPE. Note: MPE XL currently allows an SM user to by-pass a logon UDC with the ;PARM=-1 option in the :HELLO command. We believe that this feature will become configurable in the future.*

4. Access to Sensitive System Information

Each MIS department will establish procedures for identifying and securing sensitive files residing in system accounts. A sensitive file is one whose function or content can be used to compromise system security or to access company confidential information.

The MIS department is the owner of a number of critical information resources on the HP 3000 system, including the operating system and subsystem software, the host configuration files, the backup media, local utility programs, and systems programming tools. The MIS department should identify the individuals authorized for various forms

of "System Management" access and should set up the security structure needed to protect these resources against unauthorized access.

Best Practices

- *Secure the NETCON database by adding a lockword to NETCONF. Use DBUTIL to disable subsystem access, and set a maintenance word on the NETCON database.*

- *Use the :ALTSEC command to set the access to COMMAND.PUB.SYS to (R,W,A:CR;L,X:ANY). This will prevent system users from finding out what UDC files are being used and any lockwords that might be associated with them.*

- *Disable the user OPERATOR.SYS and move the operator function to its own account. Any user with a log on into the SYS account has a variety of opportunities to procure PM or SM capability. Modify SYSSTART.PUB.SYS to initiate a programmatic session on the console to over-ride MPE's default OPERATOR.SYS logon. Use ALTSEC to make sure the file access to SYSSTART is (R,W,A,L,X:CR).*

- *After any operating system update or installation, remove sensitive files, groups, and accounts used during the update process. Create a check-list with names of the specific files, groups, and accounts concerned.*

  The sensitive files include OFICACCT.PUB.SYS, PL85ACCT.PUB.SYS, PL89ACCT.PUB.SYS, SUPACCT.PUB.SYS, and TELEACCT.PUB.SYS.

  Purge the group CONVALL.SYS on V/E systems as it contains an autologon program used in converting IMAGE databases (pre UB-MIT) to Turbo IMAGE databases (post UB-MIT). This autologon program can be used to circumvent any logon UDCs used to restrict access on your machines.

  Purge any unneeded accounts left behind after a system update. These may include SUPPORT, HPPL85, HPPL87, HPPL89, CONV, ITF3000, RJE, and HPWORD.

- *Remove sensitive files created by system maintenance processes, such as BULDACCT.PRV.TELESUP. After running BULDACCT, secure or remove the JOBACCT@, JOBCUDC, and BULDPASS files created during its execution. If you use the contributed program MOVEACCT, secure or remove any JOBLIST files it has created.*

- *Regularly purge all K####### files in all sensitive accounts. These files are created when a user is aborted out of Editor or TDP.*

- *In the likely event your MIS staff uses 'GOD' or autologon tools, ensure that these tools are placed in secure groups or are secured with lockwords or ACDs.*

o Do NOT place any files in PUB.SYS that are not part of MPE. The PUB.SYS group has far too many files for the average System Manager to audit. To help ensure that no unauthorized files like K files and jobstreams end up in PUB.SYS, HOME the user MANAGER.SYS to a secure group (R,W,A,L,X,S:GU).

5. Application Access Security

The MIS department must have a procedure in place to ensure that account and group access security, as defined by the application developer, adhere to internally developed Account Management Standards. It is the application developer's responsibility to document exceptions to these standards.

(Within HP, the HP 3000 Account Management Standards identify standard group naming conventions along with access and capability for each group. For example: recommended group names for the Financial Accounting System application (FAS) are FASJCL which contains job stream files, FASDTA which contains data/database files, FASPRG which contains program files etc. In addition, specific access and capabilities are recommended for each group.)

Best Practices

o MIS departments that develop and distribute software must also develop a standardized account access structure defining standard groups and access matrices. For example, the group PUB might set its access level at (R,L:AL;X:ANY;S,A,W:GL), whereas the FASJCL group might set a stricter access level with (R,L:AL,GU;X:AC;S,A,W:AL).

o Access to important source documents and blank input forms should be limited to authorized employees. A member from data processing and a member from the user department should jointly authorize, release, and transport the use of these security sensitive forms (e.g., checks) from the storage area.

o Transaction creation and authorization should not be performed by the same individual.

o Application programs should not allow input of transactions, dates, and control data from the central computer console.

o Application programs should log successful and unsuccessful access attempts to sensitive information. Reports should be generated for examination by authorized personnel to detect possible breaches or to identify a user in the case of a known security violation.

o Enable IMAGE logging on your sensitive databases to keep track of all changes made. The IMAGE log file provides an excellent audit trail, letting you know exactly who did what, and when.

- o *Program timed terminal reads into sensitive applications which will log users off after a long period of inactivity. A long period of inactivity, say 30 minutes, is an indication that the terminal is unattended.*

- o *Maintain two separate software libraries for applications: the production library, and a test library for all programs under development or modification.*

- o *Scrambling or encryption techniques should be used by MIS departments in storing sensitive information in order to prevent unauthorized persons from using any information they may obtain accidentally or intentionally. As an example, this practice could be used to prevent a user with access to DISKED5.PRV.TELESUP from obtaining information from a sensitive database (e.g., Payroll) directly from the disk. Make sure encryption keys are documented in a physically secure place so management can access the encrypted data when a critical employee leaves the organization.*

- o *Establish secured "boxes" for sensitive output. Instruct your dispatch personnel on how to identify and route sensitive output.*

- o *Implement procedures for disposal of confidential and private data from application systems. Paper and microfiche should be shredded, and magnetic media should be over-written. These procedures should include disposal of output from aborted computer runs.*

6. File Access Security

The MIS Department must work with Account Managers to ensure that files within accounts have been appropriately secured against unauthorized access.

Best Practices

- o *Secure all unauthorized released files. There are several tools available that will secure all files not authorized to be released. Set a system-wide UDC for the :RELEASE command that will inform the users of your policy regarding released files.*

- o *Investigate the use of ACDs (Access Control Definitions) to maintain a tight control on sensitive file access. Only use ACDs, however, when standard MPE file, group, and account security is not adequate. ACDs are particularly useful when the RELEASE command appears to be the only alternative. For example, the command :ALTSEC PROFITS.PUB.FINANCE;NEWACD=(R,W,A,L,X:MIKE.FAS, JOE.FAS) will allow only the users MIKE.FAS and JOE.FAS (and the System Manager) to access the file PROFITS.PUB.FINANCE, regardless of the MPE security previously set on the file (PROFITS), group (pub), and account (FAS). Not only did MGR.FINANCE get around having to RELEASE the file*

PROFITS, but was also able to specify exactly who could (and couldn't -- implicitly) access it.

o Add both read and write passwords to your databases. Change these passwords periodically (see section E. 4. on removing database passwords from source code), or when someone with knowledge of them leaves the organization.

o Disable QUERY access to your sensitive databases using DBUTIL's >SET SUBSYSTEMS=NONE command. This will help ensure that only authorized programs access your databases. Note: using the >SET SUBSYSTEMS=NONE command only works against subsystems which check the SUBSYSTEMS flag like QUERY, ASKPLUS, and SUPRTOOL.

o Only allow (R:ANY) access at the account level when it is truly needed.

C. ACCESS MONITORING

1. Review of System Logs

Each MIS department must enable System Logging as outlined below and conduct regular reviews of the system log files to identify irregular activities on the system.

Examples of irregular activities can include multiple invalid logon attempts, logons to special capability accounts, logons to sensitive accounts at unusual times of day, MPE V/E Security Configurator changes, etc.

Best Practices

o At a minimum, enable Logging itself, Job Initiation, Job Termination, and Console Logging.

o For HP 3000s requiring strict security, sites should consider enabling Process Termination, File Close, Line Disconnection, Line Close, and File Open. Keep in mind that enabling Process Termination, File Close, and File Open can add noticeable I/O overhead to MPE V/E systems.

o When reviewing system log files, pay close attention to:

- *Multiple invalid password attempts*
- *Logons to special capability accounts*
- *All logons from dial-in ports or public networks*
- *Logons at unusual times of day*

o Use LOGAUDIT.PUB.TELESUP to automate the detection of irregular activity. Michael Hornsby's August 1989 INTERACT article "Is your HP 3000 data safe?" contains a very detailed look at how LOGAUDIT can be *used to create reports on irregular system activity.*

D. CAPABILITY CONTROLS

1. Account/Group/User Access With Special Capabilities

Each MIS department must have a procedure for authorization of special capabilities. At a minimum, these capabilities include: SM, PM, OP, NM and NA. In addition, the MIS department must conduct regular reviews of all accounts, groups, and users with special capabilities.

This procedure must include a method for identifying accounts, groups, and users that have been granted special capabilities, who requested the special capabilities, a justification for having the capabilities, and management approval of the request.

Best Practices

o *Just as new accounts require proper authorization, so too, must the request for special capabilities. Typically, the request for a new account and for special capabilities will coincide and the Account Request Form should handle them both. When a special capability request occurs for an existing account, however, a second Account Request Form must be filled out to provide a proper audit trail.*

o *Review all accounts with special capabilities at least every three months. This can be done running LISTDIR5 (LISTF on MPE XL), redirecting $STDLIST to a disk file, using EDITOR to FIND all occurrences (use EDITOR's WHILE construct) of, say, "SM", and listing the relevant lines (LIST *,*-12) to an output file. Print the file for review.*

o *Although the following capabilities are not normally considered "special", any MIS department interested in tightening security should require proper authorization before granting them:*

DI -- DIagnostician; since the CEs and System Managers should be the only authorized users of device and CPU validation programs, this capability should be restricted to the SYS and TELESUP accounts.

PS -- Programmatic Sessions; since this capability will allow a session to be initiated on another device, users should be required to justify having this capability.

2. AM Capability

Each MIS department must have adequate control over the assignment of Account Manager (AM) capability.

Account Manager capability should be limited to those individuals who have a business need and who have adequate understanding of Account

Management responsibilities. Normally, one individual per account is sufficient to perform this function. Recognize that this AM user has access to all account resources, including capabilities.

Best Practices

o Only allow one Account Manager logon in each production account.

o Maintain a record of all Account Managers and keep a copy of the Account Request Forms they fill out.

o Develop an Account Management policy for your department and distribute that policy to the Account Manager whenever an Account Request Form is submitted.

3. PM Capability

Each MIS department must ensure that all program files and groups with PM capability have appropriate security to prevent unauthorized use.

Programs and groups with PM capability may provide unauthorized users the ability to compromise system security. In particular, programs which allow users to obtain special capabilities (e.g., "God" programs), must be secured from unauthorized users. Common methods used to provide additional security for PM programs are to password protect groups with PM capability or to lockword program files.

Best Practices

o Only grant PM capability to those individuals who can be trusted to maintain the integrity of the entire machine. Understand that a knowledgeable user with PM capability is, in one sense, more powerful than a user with SM capability. The PM user can actually by-pass the operating system controls, whereas the SM user is still bound by its laws. Of course, having either PM or SM capability allows a user to easily obtain the other.

o Understand that a user without PM capability can create a PM program. This can be done by writing a program that makes privileged calls, PREPing it without PM capability, and patching the resulting program file with a contributed utility to have PM capability. This all does our user little good unless he has write access to a PM group, since PM programs may only run out of groups with PM capability. So...

o NEVER, NEVER, NEVER allow released program files in groups with PM capability. Users can simply FCOPY their own programs over them. As a more general rule, do not allow Save and Write access to files in PM groups from outside of the groups.

o Check for programs with PM capability residing in groups without PM capability. These could be programs written with the intent of breaching system security once the user locates an existing PM *program to over-write. There are third-party tools which will report such programs.*

- *Check for 'GOD' and autologon programs, as they can be used to circumvent system security. Any files in the fileset @GOD@ or @LOGON@ in any group should be scrutinized.*

- *Password all groups with PM capability unless every user in the account is authorized to have PM capability.*

- *Readers who would like to find out more about the power of PM capability and how it can be obtained illegally are encouraged to read Eugene Volokh's book "Thoughts and Discourses on HP 3000 Software."*

4. OP Capability

Each MIS department must ensure that each user with OP (System Supervisor) capability is authorized to STORE and RESTORE any and all files on the system.

Best Practices

- *As a rule, only grant OP capability to your operations account (which should be separate from the SYS account), and your System Manager accounts.*

- *For those software packages that "require" OP capability:*

  - *Read the documentation to confirm OP capability is needed.*
  - *Determine which OP commands are actually needed and see if there is a work-around for each (e.g., ALLOWing the command).*
  - *Allow the capability, if no work-around is found, and establish authorization using the Account Request Form.*

- *If an account is assigned OP capability solely to STORE and RESTORE files (e.g., databases), investigate the possibility of placing customized STORE and RESTORE jobs in groups allowing only execute access in the SYS account. Place an ACD on these files with the ALTSEC command to specify who is able to stream these customized jobs. In this way, OP can be removed from the account's capability list.*

5. NM, NA Capabilities

Each MIS department must be aware that the Network Administrator(NA) and Node Manager (NM) capabilities allow users to control the LAN and Network services. In addition, these capabilities allow access to the peripherals on the Network (i.e. printers, shared disks, DTCs, etc.).

Best Practices

- *Without appropriate authorization, only grant NM and NA capabilities to SYS, TELESUP, and the operations account.*

- *If your system uses HP's Resource Sharing, be very cautious when allowing users NM capability. With it, users may run the Resource Sharing configurator utility RESMGR.PPC.SYS, and view the passwords to the Resource Sharing default logon user PPC.SYS. NM users can also make unauthorized global Resource Sharing configuration changes. The Resource Sharing logon does not need any capabilities beyond those given a default user, so consider moving the Resource Sharing logon into an account with default capabilities.*

- *Understand that as the computer world moves towards an environment of shared resources over LAN, these capabilities will have an even greater potential for allowing security breaches and disrupting users (:NETCONTROL STOP).*

6. SM Capability

Each MIS department must be aware that SM Capability allows users to access all data on the HP 3000. In addition, SM allows users to purge all files and accounts, with few exceptions.

Best Practices

- *Clearly, as with PM capability, only those individuals who are entrusted with the integrity of the entire machine should be granted SM capability.*

- *Restrict the SM logon to the MPE defaults: MANAGER.SYS and MGR.TELESUP. There is rarely a need to grant SM to any other logons. Remove SM capability from the SUPPORT account after all operating system updates, if you decide not to purge the account.*

- *Require a unique session identifier for all SM logons. Instruct your operators and support staff to abort any SM logon without a recognized session ID and to promptly report the incident.*

E. PASSWORD MANAGEMENT

1. Password Management

The MIS Department must establish local standards for the management of passwords. Local standards should specify how often passwords will be changed and, for passwords shared among a class of individuals, how the new passwords will be communicated.

Best Practices

- *Passwords should be changed, at a minimum, every six months, or when an employee with knowledge of them transfers, leaves the company, or takes an extended leave of absence.*

- *Have your personnel department create a policy to contact the appropriate MIS managers in the event of an employee transfer or termination, in order to ensure that access is disabled for these employees. As a follow-up, personnel should route a list of transferred and terminated employees to the MIS department monthly.*

- *Develop or purchase a tool to automate the password management processes of determining expired passwords, password length, missing passwords, etc. These steps will also aid the MIS department in spotting accounts that are no longer used. Rather than changing passwords on accounts no longer in use, most Account Managers will archive and purge the accounts.*

- *If there is a legitimate business reason for giving an employee access to computer systems after a termination or change of status, then a separate legal agreement must be signed requiring the employee to protect the company's intellectual property.*

- *Understand that NS/3000 allows a remote user to by-pass LOGON UDCs using the programmatic logon feature of DSCOPY. Thus, a remote user can DSCOPY files to or from a remote logon, even when the remote machine has a LOGON UDC intended to prevent MPE access. This implies that all remote users have read and write access to all files in groups whose logons are UNPROTECTED by MPE passwords. Because of this, all MPE users and accounts on machines connected to a LAN must have passwords. Such access can also be controlled by creating ACDs on the virtual terminal device class(es) via the ALTSEC command [e.g., :ALTSEC VTERM,DEVCLASS;NEWACD=(R,W,A,L,X:JOHN.PAYROLL)]. Other possibilities include disabling NFT or AUTOLOGON access with the NSCONTROL command, but be aware that these will disrupt other applications performing Net File Transfers between systems.*

- *Understand that IMAGE allows programmatic access of remote databases. The NS/3000 discussion from above applies here, too.*

2. Password Selection

The MIS department must establish local standards for the selection of effective passwords, including MPE passwords, databases, lockwords, and any passwords used by a front-end access control system. These standards should specify the minimum length of passwords, the minimum number of numerics required within the password, and any restrictions as to the semantic content of passwords.

The password is the most important symbol of user authentication. With it a user is allowed system access. In the wrong hands, passwords can be used by an unauthorized individual to compromise system security.

Poorly selected passwords (those that are easy to guess) can be easily found and matched by a password generation program, giving an intruder unauthorized access to company assets, records and information.

Best Practices

o When choosing a password, use the following guidelines:

- Avoid passwords using simple keyboard sequences, such as "QWERTY".

- Avoid passwords associated with a logon. For example, if the user logon is MGR and the user password is "MGR", or "RGM".

- Avoid passwords associated with the user, such as a family or pet name, or a hobby.

- Select passwords that are difficult to guess, but easy to remember. One good method of doing this is combining common words and numerics, as in "CARMA90".

o Passwords should be a minimum of five characters in length to prevent a password generation program from systematically obtaining a logon.

3. Password Requirement

The MIS department must establish standards to ensure that every logon is password protected. The levels of passwords required per logon should provide additional security for systems with X.25 or LAN access and accounts/groups/users where special capabilities or sensitive data reside.

A UDC-based, front-end access control system or a security screening process provided by an application are examples of user authentication utilities which can be used.

Best Practices

o Unless appropriate written approval is received, require passwords on all MPE users and accounts. At the very least, password all special capability and sensitive users and accounts.

o Add a password to all PM groups in accounts where non-PM users may log on. If possible, don't allow non-PM users to log on to an account with PM capability.

o Add group passwords to all groups in accounts shared among two or more users, whose data is not intended for all individuals accessing the account. Don't let your Account Managers ignore this issue.

o Investigate HP and third-party software packages which can enforce your MIS department's password requirements. In addition, investigate packages that allow password requirements on terminal *ports, modems, and virtual ports.*

4. Password Protection

Procedures must be established for the identification and elimination of MPE passwords and lockwords from the following:

- **Job Stream files**
- **Schema files**
- **UDC files**
- **Source files**

In situations where removing the passwords is not feasible, the MIS department must secure the file such that read access is restricted to the file creator.

A streamer tool will eliminate the need for passwords when streaming jobs. Passwords within jobstreams and source files not only pose a security risk (especially when combined with the :RELEASE command), but also make it very difficult to change passwords.

Best Practices

- *Procure a streamer package that will allow your users to stream jobs without embedded passwords in them. Ensure the streamer tool chosen can handle:*
  - *Programmatic streaming*
  - *Password insertion for outgoing remote sessions*
  - *Parameter substitution for items such as database passwords, dates, selection criteria, etc.*
- *Do not allow your programming staff to embed database or MPE passwords in source files and schema files. Use DBUTIL to add/change database passwords after the root file is created. If the need exists, purchase a third-party tool to eliminate the need for embedded database passwords in source code or jobstreams. This will allow database administrators to change database passwords and eliminate (ex)programmers as threats to the security of your databases.*
- *Use lockwords on those UDCs where sensitive information cannot be removed. Again, make sure access to COMMAND.PUB.SYS is set to (R,W,A:CR;L,X:ANY).*

5. Default Passwords

All accounts that are created via software installation or maintenance processes must be protected by site-specific passwords, as soon as the accounts are created. This is especially important with third-party software with default passwords, well-known throughout the industry.

The presence of default passwords on accounts is one of the biggest threats to system security. Accounts with default passwords offer an easy logon to anyone seeking access to an HP 3000. Particular care should be taken to make sure accounts with special capabilities do not have default passwords.

Best Practices

o Appendix B contains a detailed list of HP and third-party accounts that are shipped either with no passwords or with default passwords.

F. PROCEDURES FOR COMPLIANCE

1. Documentation of Local Procedures

Each MIS department must be able to show evidence of proper procedures in place for ensuring compliance with the local HP 3000 Self-Audit Standards. This is done by documenting the procedures and the results of tests for compliance.

This documentation should be made available to appropriate persons, e.g., Internal Audit, functional managers, department contacts, etc.

Reference: Appendix A contains sample guidelines for documentation.

2. Evidence of Compliance

100% Compliance with the HP 3000 Self-Audit Standards is the goal of a self-audit.

Best Practices

o Each MIS department must maintain records of self-audit for at least six months. These records will provide a history of problem identification and corrective actions. Without this information, MIS management and/or auditors may not be able to rate the effectiveness of the security plan.

3. Documentation of Communication

The MIS department is responsible for communicating its own HP 3000 self-audit procedures to its user community. Evidence of this communication should be documented.

Best Practices

o As stated earlier, your users will not be able to comply with established policies unless they are informed. Limit the communication to only the information the user needs to know.

(This concludes the HP 3000 security plan and best practices.)

CONCLUSION

Both the media and some vendors of security software packages have focused a lot of attention on the threat of outside 'hackers' gaining access to your computers. It certainly captures our imaginations when we hear about junior high school students breaking into government computers. But by most accounts, this attention is highly over publicized. Most studies show that as much as 85% of all security breaches occur within an organization. For this reason, your internal security controls require at least as much attention as your external ones.

Often, internal security controls make normal system access more difficult for the users. So try to keep a balance between system security and productivity. Don't, for example, require users to enter three MPE passwords, a port security password, and a lockword to access an application. Not only will your users despise you, they will search for methods to circumvent the excessive security measures. In this case, it might be a PC script file which supplies the five passwords requested above. The result could be less security rather than more. So, take some responsibility in developing systems that are easy to use, but provide adequate authentication of your users.

Understand that your security plan must be flexible and kept up-to-date with security measures to compensate for new technologies. Perhaps it is appropriate to allow more than one AM logon on an account. It may also be cost-effective and an acceptable security risk to allow your software developers OP capability so your developers don't need to wait for STOREs and RESTORE. And such exceptions can fit nicely into your plan, as long as someone with the proper authorization ANALYZES THE RISKS and DOCUMENTS THE EXCEPTIONS in the MIS security plan. (Exceptions which grant development users greater freedom than production users is a common practice -- but only if they use separate machines and different passwords.) Proper documentation of exceptions will allow both the MIS staff and the auditors to fully comprehend and to assess the effectiveness of your security plan.

We've spent a great deal of time discussing the development of a comprehensive security plan and the need to communicate this plan to the user community. Clearly, your involvement doesn't stop here. You will need to communicate policy violations to the appropriate System Managers, Account Managers and users in a timely manner. Your ability to effectively deliver such notification will depend on how well you can automate your security procedures.

As you develop new security procedures, remember that your computer's security is only as strong as its weakest link; a link that both internal and external hackers pride themselves on finding. Keep in mind a security weakness does not have to exist on your machine to pose a threat to its security -- system security also greatly depends on the security of all machines in the network. So protect your company's assets by developing a comprehensive security plan and applying it to all machines in your company.

APPENDIX A

PROCEDURES FOR VERIFICATION AND COMPLIANCE

The following steps are an example of a proper MIS department procedure for establishing and verifying good business practices.

1. The MIS manager designates an individual to:

   - direct the development of a site security plan
   - direct the development of local HP 3000 self-audit procedures
   - direct implementation of the local procedures
   - establish ownership of the different sections of the local procedures

2. The MIS department establishes a process to verify that these guidelines are being followed. One possible process:

   a. The MIS Department:

      - communicates guidelines to functional management
      - ensures management endorsement of guidelines
      - verifies that the guidelines are followed

   b. The MIS Department identifies and documents the use of software tools used in performing self-audits and provides a check-list of points for compliance.

   c. Self-audits should be conducted at set time intervals, but at least every three months. Evidence of checking for compliance should include:

      - dates of audits
      - total number of HP 3000s on site
      - number of passes and fails
      - corrective action taken for problems

3. The MIS department documents results from departments and makes them available for review.

HP and Third-Party Default Passwords

Vendor	Package	Account/User	Password	Capabilities
Adager	Adager	Rego	antigua	PM
Bradmark	DBGENEAL	Bradmark	(none)	PM,OP,DI
Cognos	Powerhouse	Cognos	cognos	PM
Tymlabs	BackPack	Tym	(none)	PM
Vesoft	MPEX	Vesoft	Eugene	PM,OP
Robelle	Suprtool(others)	Robelle	XXXX	PM,OP
Infocentre	Speedware	Infosys	(none)	
UNISON	RADAR(others)		(none)	PM,OP
DISC	OMNIDEX	Disc	(none)	PM
OPT	QSORT(others)	Opt	(none)	
ALDON	SCOMPARE(others)	Aldon	(none)	
PROTOS	PROTOS	Protos	(none)	
HP	MPE	Telesup/Mgr	HPONLY / MGR	SM,PM,OP
		field.support	HPONLY	

*NOTE: There are additional accounts not identified here.

APPENDIX C - BIBLIOGRAPHY

ARTICLES: Title	Date/Source	Author
HP3000 Network and Security	8/89 INTERACT	R. Costandi
Conversations at The Water Cooler	8/89 INTERACT	A. Furnivall
Focus on Employees, Not Hackers	8/89 INTERACT	M.E. Kabay
Keeping up With Security Issues An Annotated Bibliography	8/89 INTERACT	M.E. Kabay
EDP and The Big Bad Auditor	8/89 INTERACT	M. Marcotte
Is Your HP3000 Data Safe?	8/89 INTERACT	M. Hornsby
EDP Security Checklist ($60.00) Associates Limited, Chesterville, Ontario, Canada	1989	M.B. Foster
What To Do If You Have a Security Breach	6/89 SuperGroup	B. Leight
The Two-Legged Virus	6/89 SuperGroup	M. Riemer
Computer Crimes: Data Diddling	6/89 INTERACT	M.E. Kabay
Computer Crimes: Sabotage	5/89 INTERACT	M.E. Kabay
New Tricks in V-Delta-4	4/89 INTERACT	M.E. Kabay
Physical Threats	3/89 INTERACT	M.E. Kabay
System Security - As Soon as I Can Find The Time	9/88 INTERACT	S.G. Bloom
Security Myths	7/88 SuperGroup	E. Volokh
Foundation for HP Data Security	1988 INTEREX	K. Spencer
A Four-Phase Security System That Works	10/87 INTERACT	M. Cooper
The Security Issue: An Overview	10/87 INTERACT	I. Blake
HP3000 Security - A Hacker's Perspective	10/87 INTERACT	T.W.Simpkins
A Guide to Breaching HP3000 Security	10/87 INTERACT	Phil Curry

Books:

Thoughts and Discourses on HP3000 Software	1989 VESOFT	E. Volokh
Computer Security Handbook 2nd Edition	1988 MacMillan	A.E.Hutt, et.al.

Eight Unknown Ways to Obtain SM Capability in Under Sixty Seconds

by

Carl E. Melchior
Independent Contractor's Group
1008 Thrush Road
Dallas, TX 75028
214-690-7847

So, you think your computer is secure? Well it's not. I know, you are using standard MPE security, installed a third party vendor security package and even met all of your company's audit criteria. The problem is none of these methods will insure computer security. I am going to tell you right now that there is no HP computer facility operating under normal conditions that I can not break into if I am given access to their site.

The old classic fight rages on. How much security is too much? The problem is not how much security your site has, but in how that security is managed. The perfect example is the one that is present in every HP site. How many people keep their passwords secret? Anybody who has been around computers knows that eventually you wind up with the passwords of 50% of your co-workers. It doesn't matter what kind of security you have if it is not managed in a secure way.

In particular I am going to address how easy it is to break a computer's security. No, you don't have to be an SPL programmer or system's guru in order to obtain SM. You just have to use a little common sense and you will see how easy it really is. Now I don't advocate by any means that you try any of these methods as unauthorized computer access may constitute criminal action. The reason for this paper is to alert system managers as well as DP managers about the ways that your site can be violated.

What if I were to tell you that there is a way to obtain the SM password without ever logging on to the HP? Impossible you say? Wrong!!! There are actually two ways you can obtain the SM password without ever having to logon to the HP and I've tested both of them. Later on I'll show you how but for now remember one thing and it is this. All of the ways you are about to be shown can be accomplished with less than sixty seconds of actual work. There is no magic to it, there are no tricks. These are just ingenious ways that have evolved to violate security systems. Some of the ways are straight forward and some involve a few spy-like techniques. Once you think you've seen it all is just the time when a new way pops up that nobody is even aware of.

Read on now and see how many of these techniques can be applied to your site. Remember, sixty seconds.

In general there are three quick methods for breaking into computers. The first is by accessing communication's lines, the second is by accessing terminals or PCs and the third is by gaining access legally and then retaining it illegally. Let's look at what the standard HP site has to offer. At most sites I've visited the management is very proud of the fact that they are a state of the art DP facility. This usually means that instead of terminals they have PCs linked over phone lines. Herein lies the root of most security problems.

1) The first and probably easiest way to get an SM password is to "bug" the SM's PC. This can be done in a variety of ways. It would be similar to hooking up a datascope on your terminal and viewing all of the terminal output that comes across the data line. One way that works is to take a standard one line in two lines out y-converter and plug it into the SM's modular data jack next to his desk. Our culprit would then run the SM's computer line into one side of the jack and his own terminal line into the other side of the jack. In effect what will happen is that he will monitor everything that SM does. One thing that would have to be done is to cut the transmit wire on the culprit's side of the jack so that he would receive only and not be transmitting any signals. There are a few other ways of accomplishing the same thing but you get the general idea. Since there aren't many SM's who kneel down every morning to check there jack configuration this will usually go undetected. The only other problem is the unauthorized person may have to hook up to a closer terminal if his office is located on the other side of the complex from the SM.

2) The next three ways address the fallacy that terminals locked by user passwords are secured. Nothing is further from the truth. Matter of fact when someone locks their terminal by requiring a password to unlock it, they have virtually left their system wide open. Here is how it works.

2) Let's say your the system manager and you get a call to come to someone's office. You get up from your desk but before you leave you password lock your terminal meaning that before anyone can get back to the colon prompt he has to know your password. After our culprit sees you leave he then enters your office. Since SM is using a PC he is probably using one of the standard third party vendor HP terminal emulators. Almost all of them have a very unique option. That option is to log what ever you do to disk. If our unauthorized person drops into PC mode which does not effect

our locked session then he/she can turn disk logging on and return the PC to session mode with the session still asking for the password to unlock the screen. However, everything the SM types in will now be logged to disk without his knowledge. At the end of the day when the SM leaves the culprit then returns to read everything that SM has done today. Data that is entered with echo off will not show up on the log file but any job streams with passwords or any database critical data will be shown. It doesn't take long to identify which parts of the data are passwords.

3) If someone were really interested in carrying the above example a little bit further all they have to do is to write a program that does one thing. That program would emulate the password program.. Imagine this. Once again our SM leaves his terminal locked by password in his office. Our would be thief walks into the SM's office and enters three bad password attempts. This now logs SM off of the system. Our thief now logs on as himself and runs his own password emulation program. The first thing he does is enters two bad password attempts and leaves the third one sitting on the screen waiting for the SM to return. When SM returns he enters his password and the emulation program fakes a bad password attempt for the third time and the SM is logged off from the culprits session, not SM's session. Normally not to much attention is paid to password attempts on locked terminals since any accidental entering of characters can cause the terminal to issue a couple of bad messages. I have found that the normal sequence of events is for the SM to log back on as SM and maybe try the real password program again and this time it will work right and he will think he just typed in something wrong. One thing to remember is that since the culprit was running his own program he had the ability to trap the SM's password when he entered it on the screen, even though echo was off. Our culprit is now SM.

4) Probably the easiest way to obtain SM is to swap port plugs or modular jacks. I have seen offices where access was available at three different junction switches where the swap could be made. If you wait until the system manager is on the phone and not using his terminal all you have to do is swap his port plug to your terminal and you are SM until you switch it back. You can list every password and user and he will never see it on his screen. The only requirement for this to work is that he does not have his screen password locked.

5) One of the best ways I have seen to get the SM password involved a company that kept there SM password sealed in a tamper proof confidential envelope." It was specially made so that it

couldn't be read if held up to the light. In emergency cases an analyst would come in from home open the envelope and log on as system manager to fix the emergency. The next day the SM would change the password and put the new password in another sealed envelope. However, there exists in every site a common material that will foil this security. If you walk into your computer room and look around you will probably find a can of Freon solvent that is used to clean tape drives. If you pour about one ounce of this liquid onto the envelope it will turn the envelope into "glass" for about 30 seconds and then dry without any trace of tampering.
You can read anything and everything inside the envelope. This is what the U.S. Postal Service uses to detect mail bombs. Be careful though, this method works on anything in sealed envelopes especially paychecks or confidential information.

6) If your not the type of person who would go to these lengths then all you really have to do is create your own emergency as noted above and legally obtain the SM password. Once you are into the account there are a number of things you can do to retain SM capability illegally. You could alter SM's catalog to execute your own little program to always write SM's password periodically to a PUB file. This eliminates suspicion on your part. Once you have SM it is virtually impossible to track your ability to recall it from your own personal files at any time. This is a whole subject onto itself and one that provides an abundance of illegal ways to retain SM capability and all undetectable by standard means.

7) Now I know you have been waiting to find out how you can obtain SM password without ever logging on to the system. Well I already told you one way in #5 above. You don't need to logon to the HP to read the SM password if you have your trusty can of Freon. But here is the way I like best of all. There is a third party vendor that has come out with a product that didn't have security in mind when they made it although it is a very useful product. This software gets loaded on to your PC and runs in the background recording your keystrokes so that you can recall them at a later time and use them in macros. So this is how it works. Once again our culprit waits for SM to leave at the end of the day. He/She walks into SM's office and loads this software to the PC disk. The next morning our SM logs on and everything he does remains in the keystrokes log file. This program even catches echo off passwords in most situations. When SM goes to lunch our culprit reads the PC disk file and has SM's password. He/She then purges the software program off of the PC.

8) The last and most abused way is that if all else fails you can

always walk into the system manager's office every morning and look over his/her shoulder until you can memorize enough keystrokes to figure out SM's password. Sometimes simple is best.

Now that you have seen some of the ways your site can be broken into, let me make a few suggestions on what you need to do if you want to insure site security.

1) For all people that use PC's, they should always physically lock their terminals and keyboards. This will prevent somebody from having unauthorized access.

2) Never have your lockword for password lock programs the same as your MPE-id password. This means that if somebody finds out your lock password that they don't have your MPE-id password and that to get it they would have to wait until you are logged on and at a lock prompt. They would then have to enter your lockword and run Listuser to obtain your MPE-id password. This is not totally safe but at least they have to risk detection by passers-by while using your terminal.

3) If you abort for unknown reason (i.e. bad password attempts) check the computer log and make sure it was you that logged off. In any case the computer log files should be checked daily and SM logons verified. No sites do it that I know of but it is one of the few ways to help insure security.

4) And lastly, the only way to really insure your system is safe once it has been violated is to do a reload from tape. I know it is a time consuming process but it is the ONLY way to insure computer security from that point on. Remember, 60 seconds is the time it takes for the SM to walk down the hall to get that cup of coffee and come back. How many cups of coffee does your SM drink in a day?

SOFTWARE CREATION WITH A 4GL LANGUAGE: LESSONS LEARNED

by
Gene Harmon
AH Computer Services, Inc.
8210 Terrace Drive
El Cerrito, CA 94530
415 525-5070

I. Introduction

The usual use of a 4GL language, such as the one I use: the Cognos Powerhouse product, is to generate business application solutions for users. The 4GL product is commonly referred to as application software and performs the job of database maintenance, inquiry and reporting.

MIS installations also have other types of software, which typically fall into the categories of system, development, and utility software . . . these software packages may come in bundled or unbundled form with the hardware purchased, or may be purchased from third party suppliers.

However, when special needs arise, a 4GL software package can be adapted to serve the purposes of one of the other categories of software. The techniques used are somewhat similar to those of the development of application software, but must also incorporate the parsing and file handling techniques required by utility packages.

The reasons which prompted me to write the software described in this paper are two: 1. I wished to save time in the generation of template report programs (and there were none available for purchase) and 2. I wanted to have a documentation and cross-reference package which I could take from installation to installation without having to buy one from either the 4GL software supplier or from some other vendor.

II. Schema Conversion

For the purposes of creation of my first piece of software: the Quiz program generator, I needed to have available computer files which described the files that the 4GL knew about in its schema. The easiest way for me to capture this information was to direct the output of the dictionary display program: Qshow to an MPE file, and then use that MPE file as a driver for creation of a database consisting of files and elements. All of this can be done with procedure code in the Cognos Screen Handler: QUICK so that all the user has to do pick a menu choice which will generate the Files/Elements database.

To aid in the handling of the Qshow process by the Quick program, it was necessary to set up to MPE direct files in the Cognos dictionary: LINEIMG2 and LINEIMG4.

The mechanics of obtaining the Qshow output are as follows:

STEP 1.

```
DEFINE D-BUILD-1 CHAR*60 = PACK("BUILD " + "RCSHOUSE" + &
                          ";DISC=100;REC=-80,16,F,ASCII;TEMP")

DEFINE D-BUILD3 CHAR*60 = PACK("BUILD " + "LINEIMG2" + &
                          ";DISC=10;REC=-80,16,F,ASCII;TEMP")

DEFINE D-FILE-QSCHOUSE CHAR*50 = "FILE LINEIMG2 = RSCHOUSE" + &
                                 ",OLDTEMP"

RUN COMMAND D-BUILD-1
RUN COMMAND D-BUILD3
RUN COMMAND D-FILE-QSCHOUSE

LET LINE-IMAGEA OF LINEIMG2 = "SET REPORT DEVICE PRINTER"
PUT LINEIMG2
LET LINE-IMAGEA OF LINEIMG2 = "SET COMPRESS"
PUT LINEIMG2
LET LINE-IMAGEA OF LINEIMG2 = "SET NOPRINT"
PUT LINEIMG2
LET LINE-IMAGEA OF LINEIMG2 = "SHOW FILES DETAIL"
PUT LINEIMG2
LET LINE-IMAGEA OF LINEIMG2 = "EXIT"
PUT LINEIMG2
CLOSE LINEIMG2
RUN COMMAND "RESET LINEIMG2"
```

STEP FUNCTIONS:

A. Create a physical file for putting Qshow commands.

B. Execute a File Equate of that physical file to the logical file LINEIMG2 so that quick may put the Qshow commands to that file.

C. Set up the Qshow command file with the above "LET's" and "PUT's"

D. Close and reset the logical file LINEIMG2.

STEP 2.

```
DEFINE D-BUILD CHAR*60 = PACK("BUILD " + "QLST" + &
                          ";DISC=10000;REC=-80,16,F,ASCII;TEMP")

DEFINE D-FILE-QSCHLST CHAR*50 = "FILE QSHOWLIST = " + "QLST" + &
                                "',OLDTEMP;DEV=DISC"

DEFINE D-FILE-EQ1 CHAR*50 = "FILE LINEIMG4.GENE" + " = QLST" &
                             + ",OLDTEMP"

DEFINE D-PURGE3 = "PURGE " + "LINEIMG2"

RUN COMMAND D-BUILD
RUN COMMAND "FILE STDLIST = $NULL"
RUN COMMAND "FILE STDPRINT = $NULL"
RUN COMMAND "FILE QSHOUSE = LINEIMG2"
RUN COMMAND D-FILE-QSCHLST
RUN COMMAND "RUN QSHOW.CURRENT.COGNOS"
RUN COMMAND "RESET QSHOUSE"
RUN COMMAND "RESET QSHOLIST"
RUN COMMAND "RESET STDLIST"
RUN COMMAND "RESET STDPRINT"
RUN COMMAND D-PURGE3
RUN COMMAND D-FILE-EQ1
```

STEP FUNCTIONS:

A. Set up a physical file to hold the output of Qshow (D-BUILD).

B. Issue file equates to suppress the print devices.

C. Issue a file equate to equate the Qshow use file to LINEIMG2 (which now the Qshow commands in it)

D. Issue a file equate to equate the Qshow dictionary output file to the MPE file QLST built by D-BUILD.

E. Run Qshow and reset all the files you made reference to.

F. Purge LINEIMG2

G. Issue a file equate to equate LINEIMG4 to the MPE file QLST which now holds all of the Qshow output.

The final output of these two steps can be seen on the following page. Since this output is now in an MPE file equated to LINEIMG4, it is now possible to parse each of the records and build a database consisting of sets containing the information needed about files and elements. As can be seen from the example, there is sufficient information contained in the file to determine all files, miscellaneous information about files, all elements, miscellaneous information about elements, all keys, and record and element size. If more information was wanted about elements, then you could have used the Qshow statement "SHOW ELEMENTS" in combination with "SHOW FILES" instead of "SHOW FILES DETAIL".

PARSING CONSIDERATIONS:

Once the dictionary output is sitting in an MPE file equated to LINEIMG4, we can do something like this in the QUICK program:

```
  WHILE RETRIEVING LINEIMG4 SEQUENTIAL
   BEGIN
     IF LINE-IMAGE[1:50] <> " "
       THEN BEGIN
         LET LINE-IMAGE OF LINEIMG4 = PACK(TRUN(LINE-IMAGE OF
                                     LINEIMG4))
     IF LINE-IMAGE OF LINEIMG4 [1:5] = "File:"
           THEN DO INTERNAL WRITE-MASTER

  . . . . .
```

The above code will not process blank records, which will save on processing time, and will additionally keep extra records out of the processing cycle. I have found that eliminating extra spaces between literal information on the records can be a help when you wish to position yourself on the first position of the very next literal of the record as you are parsing your way literal-by-literal across the record

```
   LET PTRX = INDEX(LINE-IMAGE, "INTEGER")
```

The above code (once you have determined that you are dealing with a record containing an element) will tell you that the element is of the type "Integer" if the value in PTRX is greater than zero; since PTRX will contain the starting position in the record of the literal "INTEGER" (if found), then you could say

```
  LET PTRY = INDEX(LINE-IMAGE [PTRX: (81- PTRX)], " ")
```

```
>SHOW FILE QFILDET1
-----------------------------------------------------------------


          File:                QFILDET1
          Organization:        DETAIL
          Type:                TURBOIMAGE
          Open:                QFILDET1 of AHIQ
          Capacity:            5001 Records

--Record Contents--

Key  Item                      Type       Offset  Size  Occurs
 R   QFILE-NAME                CHARACTER     0     16
 R   QKEY                      CHARACTER    16      6
     QELEM                     CHARACTER    22     16
     QPIC                      CHARACTER    38      6
     QLEN                      CHARACTER    44      4
     Q-TYPE                    CHARACTER    48     16
     Q-FLAG-PICK               CHARACTER    64      2
     Q-FLAG-SORT               INTEGER      66      2
     Q-FLAG-TOT                CHARACTER    68      2
     Q-FLAG-SEL                CHARACTER    70      2
     Q-FLAG-DEF                CHARACTER    72      2
     Q-FLAG-OTH1               CHARACTER    74      2
     Q-FLAG-OTH2               CHARACTER    76      2
     Q-FLAG-OTH3               CHARACTER    78      2

Record Size:    80 Bytes
       Keys:    QFILE-NAME          Repeating of QAUTO
                QKEY                Repeating of QAUTO1
```

Then you would know that the next literal on the record starts at (PTRY + 1)...always remembering that if your record length is 80, then you aren't interested in any values greater than 80.

Sometimes it is necessary to search backwards through the record (perhaps looking for the first " " or some other significant character. This can be accomplished by doing an internal recursive routine. For example:

```
PROCEDURE INTERNAL FIND-LAST-POS
BEGIN
  LET T-POS = T-POS -1
  IF LINE-IMAGE OF LINEIMG4 [T-POS:1] = " "
    THEN DO INTERNAL FIND-LAST-POS
END
```

In the above code, the value of T-POS would be set at the value of the first position in LINE-IMAGE not containing a blank when the procedure was exited.

III. QUIZ STATEMENT GENERATOR

Although the previous section was required to set up the files for the Quiz statement generator to work on, the exercise was a good start on the techniques needed to approach this next task of generating a Quiz source statement program based on a selection of file and element names picked by the user.

In addition to a beginning to the understanding of a standard parsing technique, this exercise required the generous use of Image workfiles to accomplish its goals. Rather than attempt to include all of the keys and indicators in a main series of File and Element and Report datasets, I found it was easiest to just create a new dataset as a repository for whatever information needed to be passed on the next required screen or to the final step of Quiz statement generation. This technique requires that these datasets be functionally erased upon first entry into the screen, or upon first entry into the process. In practice, I have found that it's easiest to erase all of the internal workfiles upon entry into the first functional screen of the process, and to let files which are subject to screen update be changed by the user.

DELETE EXAMPLE:

```
  WHILE RETRIEVING QWORKD VIA QKEY USING (T-KEY + "AA")
    BEGIN
      DELETE QWORKD
      PUT QWORKD RESET
      END
```

It is also helpful also allow the user to assign a 4-position user key which will carry through the entire process. This will allow the user to sustain several work sets at the same time, and will facilitate multi-user access to your work files. If you do this, then you should also, when in the menu which assigns the user-id, allow for deletion or continuation of the work sets associated with a particular user-id.

One of the advantages of the multiple image work file approach is that you can always add another function quite easily. For instance, the last step of the Quiz statement generator uses the following image work files:

```
  HOLD-SET-STATEMENTS
  HOLD-FILE-EQUATES
  HOLD-ACCESS-STMTS
  HOLD-SELECT-STMTS
  HOLD-SORT-STMTS
  HOLD-REPORT-STMTS
  HOLD-FOOTINGS
```

However, half of these files were added as enhancements. Another advantage of the Image files is their ability to add items in sorted sequence; this feature allows the designer more flexibility in screen design (can allow the user to arbitrarily number his selections) and file manipulation.

One problem which can occur is that of exceeding the default execution time parameters controlling internal buffers, tables, expression sizes, code pages, etc. Some of them may have to be changed, and it may not always be obvious which are the right ones to change. My approach was to put various parameters to the maximum while putting others to the minimum until my programs finally worked. Two of the parameters which seem most critical for this type of work in the Cognos product are expression size and Procedure code pages. While some of these problems may not exist on the Spectrum hardware, some will remain --the one which I have run into is the number of file buffers which Cognos can maintain.

III. DOCUMENTATION PACKAGE

Writing the documentation package proved to be a combination of parsing learned in the dictionary set-up and the Image work file technique learned in the Quiz statement generator. There were additional problems to be resolved involving the access of all files in a group and more severe parsing challenges.

The goals of the documentation package were to examine all of the application source code written using the Cognos product, and generate reports showing items of interest found in a source module, such as file names, element names, output and update actions, etc. Also, these items identified could be used in a cross-reference screen system.

A. Access all files in a group

The method used here was to direct an MPE LISTF of all files in the group of interest to an MPE file, and then use that file as a driver to access each of the files in turn and put those files, record by record, through the documentation parsing process. When used at various installations, it turns out that you cannot always be sure that the files have the same record-length, so an additional step is used to FCOPY each file into a file which has a record length of 80.

```
DEFINE D-LISTF CHAR*50 = &
       "LISTF @." + TRUN(T-GROUP) + ";*LINEIMG3"

DEFINE D-LISTF-EQ CHAR*50 = &
  "FILE LINEIMG3 =" + "LINEIMGX" + ",OLDTEMP"

DEFINE D-LISTF-BLD CHAR*60 = PACK("BUILD " + "LINEIMGX" + &
       ";TEMP;DISC=300;REC=-80,1,F,ASCII;NOCCTL")

DEFINE D-JOBNAME CHAR*17 = "LINEIMGX"

DEFINE D-BUILD CHAR*60 = PACK("BUILD " + "AHIDCOPY" + &
       ";DISC=30000;REC=-80,16,F,ASCII;TEMP")

DEFINE T-FCOPY CHAR*79 = "RUN FCOPY.PUB.SYS;INFO= " + &
      '"FROM=' + TRUN(T-FROM) + ";TO=AHIDCOPY;IGNERR" + '"'

DEFINE D-SAVE CHAR*40 = "SAVE " + D-JOBNAME
```

```
PROCEDURE INTERNAL BUILDFILE-GRP
BEGIN
  LET T-FROM = TRUN(LINE-IMAGE OF LINEIMG4) + "." &
               + TRUN (T-GROUPNM)
  RUN COMMAND T-FCOPY
  RUN COMMAND "FILE LINEIMG2.GENE = AHIDCOPY"
END

PROCEDURE INTERNAL BUILDFILE
BEGIN
  RUN COMMAND D-BUILD
  RUN COMMAND D-LISTF-BLD
  RUN COMMAND D-LISTF-EQ
  RUN COMMAND D-LISTF
  CLOSE LINEIMG3
  RUN COMMAND D-SAVE
  RUN COMMAND "RESET LINEIMG3"
  RUN COMMAND "FILE LINEIMG4.GENE = LINEIMGX"
END

PROCEDURE INTERNAL SET-FILE-GROUP
BEGIN
  DO BUILDFILE-GRP
  WHILE RETRIEVING LINEIMG2 SEQUENTIAL
   BEGIN
     LET T-IMAGE = LJ(LINE-IMAGEA OF LINEIMG2)
     LET PTRX = INDEX(T-IMAGE, ";")
     IF  PTRX > 1
       THEN
         LET T-IMAGE = T-IMAGE[1:(PTRX - 1)]
     IF T-IMAGE [1:80] <> " " AND
        T-IMAGE [1:1]  <> ";"
       THEN BEGIN
         LET T-IMAGE = PACK(UPSHIFT(TRUN(T-IMAGE)))
         DO INTERNAL FIND-SCREEN-FUNCT
         DO INTERNAL PROCESS-SCREEN-FUNCT
         . . . .
         END
     END
   CLOSE LINEIMG2
   RUN COMMAND "RESET LINEIMG2.GENE"
END
```

```
PROCEDURE INTERNAL SET-FILE
BEGIN
  DO BUILDFILE
  WHILE RETRIEVING LINEIMG4 SEQUENTIAL
    BEGIN
      IF LINE-IMAGE OF LINEIMG4 [1:50] <> " " AND &
         LINE-IMAGE OF LINEIMG4 [1:8]  <> "FILENAME"
      THEN BEGIN
        DO INTERNAL SET-FILE-GROUP
        . . .
        END
     END
   CLOSE LINEIMG4
   RUN COMMAND "RESET LINEIMG4.GENE"
END

PROCEDURE ENTRY
BEGIN
  DO INTERNAL SET-FILE

. . .
END
```

FUNCTIONAL EXPLANATION OF ABOVE CODE:

Since this is a silent (or ghost) screen (activities ENTRY) which is called from a menu screen (while passing filename.groupname), the whole thing starts off at the PROCEDURE ENTRY. This procedure calls SET-FILE which in turn calls BUILDFILE. The procedure BUILDFILE creates a file which will hold all the output from the LISTF command. The procedure SET-FILE looks at all of the filenames delivered by the LISTF command and calls the procedure SET-FILE-GROUP which copies each file into the physical file AHIDCOPY (by calling BUILDFILE-GRP) which is then equated to LINEIMG2 which finally contains all of the records for a particular source file in a group. This source file is then parsed record-by-record by the remainder of the program via calls in the procedure SET-FILE-GROUP.

Looking at the procedure SET-FILE-GROUP, notice that the comment lines, which are flagged by a ";" in Cognos source code are being screened out. Also notice the all of the source code is being upshifted, packed (only 1 space between literals), and truncated.

B. THE STANDARD MODEL OF ANALYSIS

Once confronted with the task of analyzing the set of records of the source file of interest, I found that a standard model of code structure could pretty well satisfy the analysis needs of any set of literals. In other words, for each pass of one record through all of the code, I found that if I wanted to find either a file name or a field name or a particular procedure code instruction, the overall structure of the code for each type of search turned out to be just about the same.

```
TEMP T-IMAGE    CHAR*400 INITIAL ""
TEMP T-IMAGED   CHAR*400 INITIAL ""
TEMP T-FUNCTA   CHAR*1   INITIAL ""
TEMP T-POS      NUM*4    INITIAL 0
TEMP PTRA       NUM*4    INITIAL 0

PROCEDURE INTERNAL FIND-AMPER2
BEGIN
  LET T-FUNCTA = "N"
  LET T-POS    = T-POS - 1      ;WORK BACKWARDS THRU THE RECORD
  IF T-IMAGE[T-POS:1] <> "&" AND ;HAVEN'T FOUND THE CONTIN. CH.
   T-POS <> 1                    ;HAVEN'T HIT POS 1 OF THE REC.
      THEN DO INTERNAL FIND-AMPER2 ;LOOP BACK THRU
  IF T-IMAGE[T-POS:1] = "&"       ;THIS IS A CONTIN. CHARACTER
     THEN LET T-FUNCTA = "Y"    ;GO GET ANOTHER REC. FOR THIS SERIES
END



PROCEDURE INTERNAL FIND-FIELD-FUNCT
BEGIN
  LET PTRA = 0
  LET PTRB = 0
  LET PTRC = 0
  LET PTRA = INDEX(T-IMAGE, '"') ; COULD THIS BE PART OF A LITERAL?
  LET PTRB = INDEX(T-IMAGE, "FIELD ") ;LOOKING FOR A 'FIELD' STMT
  IF (PTRA <> 0 AND PTRA > PTRB AND PTRB = 1) OR &
     (PTRB = 1  AND PTRA = 0)
     THEN BEGIN
       LET T-FUNCTA = "Y"  ;GOT A HIT ON 'FIELD'- START THE PROCESS
       LET T-IMAGED = " "  ;RESET THE WORK AREA FOR 'FIELD'
       END
END
```

```
PROCEDURE INTERNAL PROCESS-FIELD-FUNCT
BEGIN
  LET PTRX = 0
  LET PTRY = 0
  LET PTRZ = 0
  IF T-FUNCTA = "Y" ;OK TO START THE PROCESS FOR A 'FIELD' SEARCH
    THEN BEGIN
      LET T-POS = 81
      DO INTERNAL FIND-AMPER2
      LET T-IMAGED = TRUN(T-IMAGED) + T-IMAGE  ;FOLD IN THE RECORDS
      IF T-FUNCTA = "N" ; THERE WERE NO MORE CONTIN. RECS.
        THEN BEGIN
          LET T-FUNCTA                  = " " ;RESET FOR NEXT PASS
          LET PTRB                      = INDEX(T-IMAGED,"FIELD ")
          LET PTRC                       = PTRB + 6 ;BUMP UP TO NEXT
          LET D-SCREEN OF DOCFLD       = TRUN(LINE-IMAGE OF LINEIMG4)
          LET D-SCREEN OF XREFKSAM     = TRUN(LINE-IMAGE OF LINEIMG4)
          LET T-IMAGED                  = T-IMAGED[PTRC:(400 - PTRC)]
          LET PTRD                      = INDEX(T-IMAGED, " ")
          LET D-FIELD  OF DOCFLD        = T-IMAGED[1:PTRD]
          LET D-FIELD-VALUE OF XREFKSAM = T-IMAGED[1:PTRD]
```

FUNCTIONAL EXPLANATION OF PRECEDING CODE:

As each record is passed through the code, the procedure FIND-FIELD-FUNCT is looking for one which has the literal "FIELD " in it (but not part of some other literal bounded by quotes).
Once found, this and each succeeding record is examined in the procedure FIND-AMPER2 to see if there is to be another record which will contain part of the field statement (there is an ampersand at the end of the record being processed). When the last record of the FIELD statement is processed (no ampersand was found as the last character of the record), then the procedure PROCESS-FIELD-FUNCT is fully processed.

As can be seen, there is a work area called T-IMAGED which is used to assemble together in one long string all the records which comprise the FIELD statement. T-IMAGED is 400 characters long, and, since all the records have been stripped of all their extraneous spaces and truncated, there is enough room to hold the complete statement. The fact that the complete text of the statement is in just one long string helps considerably in the analysis of the statement for all the contained variables. Also, as you can see with the code:

```
  LET T-IMAGED = T-IMAGED[PTRC:(400 - PTRC)]
```

it allows you to dispose of already analyzed code as you go along, and just concentrate on what is left.

C. OTHER TECHNIQUES

When this documentation effort was extended to Quiz and Qtp source code, it was found that the analysis of the ACCESS statement was best done by loading the entire statement one record per literal into an Image work file, and then using a

```
 WHILE RETRIEVING filename VIA key USING data-item-name BACKWARDS
```

to analyze the statement. Because Cognos allows so many linkage variations from fileA to fileB, it is very difficult to know what to do with the literals until you finally get to fileB; this method does away with the need to go back along the string and identify what you have already processed.

Sometimes, you may need to create workfiles to use later on in the process. For example, when analyzing FILE statements, in Quick source code, they may have alias names which are often referred to in the Procedure section of Quick. When encountering these alias names in the procedure code, it is necessary to do a lookup on the Alias-Work-File to see what the dictionary file name is so that it can be reported properly.

FINAL THOUGHTS

When using a 4GL to create utility software, it is possible to duplicate many of the features found in third-party packages. Since you do not have direct access to the intrinsics, there is a certain amount of awkwardness and lack of speed in some of the operations you need to perform; in fact, some of the things you want to do may be impossible. However, I hope I've shown you some techniques which may allow you to access changing source code in a dynamic manner for whatever purposes you require.

Stop Babysitting Your HP3000

Craig Conrad
McDonnell Douglas Space Systems Co.
5301 Bolsa Ave. MS 12-1
Huntington Beach, Ca. 92647
(714)896-3311, ext. 0028

Introduction

Any computer system with the right hardware, software and operations techniques can run with little human intervention. The HP3000 is no exception. A variety of tools have been developed to manage our jobs, keep an eye on our spoolfiles, shrink our backups and even alert us when the environment in the computer room goes awry. Combine these products with some creative operations techniques and more of your time can be spent pursuing new avenues rather than stagnating in an operations routine.

This paper will discuss the tools and techniques applied to automate a twenty-four hour shop. It will explore some of the options available with job management systems, spool managers, down detectors and backup tools currently on the market. It will also look at methods of accomplishing many of the same results if your D.P. budget is prohibitive of purchasing these tools, including some innovative ways of dealing with those multiple reel backups that necessitate second shift personnel.

Since this paper discusses Third-party solutions to many of our operations problems, it should also be mentioned that a great many tools have also been written by your fellow HP'ers and you might want to make the Swap Tapes your first stop when setting out to automate. I will be happy to provide information to anyone interested, to help in obtaining copies of contributed programs or jobs and systems that I have developed that are discussed below.

What Will Investing In Automation Buy Me?

The investment in automation can be an extremely advantageous pursuit. It need not be expensive and many times will require only moments of your time to whip up a small jobstream, freeing someone of a routine task. For example, it may be necessary to start a particular process at 8:00pm each evening. However, depending on the day of the week, it may be a different job. You could build a menu system for your operator(s) to use or give them a list of commands to key in. But what if your operator calls in sick, or is working on another problem and forgets to start the process? Instead, build a job that will launch at 8:00pm and determine the day of week, day of month or time of day and then launch the correct job for the particular conditions. Also include a section in this job so that when it's finished, it will restream itself for the next run. You can carry this one step further by including a STREAM statement in your "SYSSTART" file so the job is set up whenever the system is rebooted. It's quite possible to build a jobstream, and never have to stream it for it to run. Your process now runs automatically, exactly when is supposed to, and never runs the wrong job on the wrong day.

Improve Productivity and Moral

The decision to start an automating project should not be with the intent of cutting staff. If you cannot afford to hire staff or the final outcome is a staff reduction, fine. However, during the automation process, it will be quite important to have the support of your staff and if they feel their jobs are being threatened, they won't be much help. Make the automation project something that everyone can participate in and they will see it as a challenge. A good approach to take might be to present the idea that automation will free them to work on other projects. They will have an opportunity to expand their knowledge and further their careers. Each individual will begin to look for little things that they don't want to do manually and many of these can be incorporated as well.

The result is that your staff feels good because they have helped create a better environment to work in and can now improve their skills in other areas or handle things that have been waiting for their attention. Your staff now has a stake in the operation and loyalty and responsibility can be improved.

If you're automating because you have limited staff and can't afford any more, then you'll benefit greatly by creating more time for yourself and maybe you can eliminate some (or all) of those greater than eight hour days.

Reduction of Errors and Inconsistencies

By building smarter jobstreams, the chance of error can almost be completely eliminated. Almost, because there will still be occasions when a job won't run correctly because of such things as 'exclusive file access' errors, etc. However, a correctly written jobstream can check for these errors and not continue with further steps until corrective action has been taken. Most job schedulers will also allow you to set dependencies on jobs and not permit dependant jobs to run if a preceding job of the process has failed. A job that stops when it encounters problems will save you a lot of time when you don't have to back out changes that should have never been made in the first place.

Many shops have their operators do routine processing; the same function each day or each week. An example would be an update to a database. Data in this database is extracted from an IBM system and you receive an extract tape once a week. Your operator mounts the tape and then uses a system of screens to process it. Suppose that the operator inadvertently runs the process in the wrong order or is distracted by someone with a terminal problem? What's going to happen to your update? It will either be corrupt and have to be corrected or it will not finish in a timely manner. Is there really anything in that system of screens that a batch job can't do? Sometimes, yes. Most times, NO. If you rewrite the process into a jobstream then your operator's only responsibility will be to mount the tape and intervene if problems are encountered. They suddenly have the time required to deal with that terminal problem and your update gets done as well.

Scheduling and Performance Improvements

Your system never loses track of the time. It can therefore never forget to run something at a given time. By using the build-in scheduling features or having a job scheduler run your jobs, processing schedules can be strictly adhered to.

One last advantage to automating can be an increase in the performance in your system. By examining the load of your system ("HPTREND" can help here), you can schedule a lot of processing during the slower periods of use. Take advantage of lunch hours, morning and afternoon breaks, the time between five o'clock and when your backup starts and any other time that shows a low CPU utilization. By balancing your load, you'll get the most processing power for your dollar.

Automating Your Backups

Probably the biggest problem facing most shops is getting those backups done. After second shifts, third shifts, split shifts and batch processing, when is there time left to spin tapes? We've all resigned to the fact that some of your processor's time is going to have to be dedicated to getting backups done. The real issue is when to do it and how long does it take?

First we need to determine what strategies we can apply. Some of the options are, the standard SYSDUMP and STORE/RESTORE, DBSTORE/DBRESTORE, third party utilities, and a method I've used that I call "Perpetual Backup", which is a method where you are always backing up something at any given time.

SYSDUMP, STORE/RESTORE, DBSTORE,DBRESTORE

The most common backup methods are the ones that H.P. gave you when you bought your system. These, from my experiences, also seem to be the safest.

Let's look at how far we can take automating the full and partial backup with the options available. A jobstream, "BKUPJOB.OPERATOR.SYS", which has appeared in the "Communicator" is included at the end of this paper. This job runs everyday at a specified time. It determines what time it is and what day it is and then based on that information, performs a specific backup. Using IF/ELSE/ENDIF condition checking, it is capable of making some simple decisions. Simple decisions, however, are all that's required to perform a backup. Also note that there are some programs which are run to logoff forgetful users. These programs are contributed and have been on the Swap Tape at one time or another. If you can't find them somewhere, I'll be happy to assist you.

"BKUPJOB" first determines the day of the week. If it's the day that a full backup is to take place, it continues to perform that function, otherwise it transfers control down to the partial backup routine. After each backup routine, the job checks JCW's to insure that no errors have occurred. If errors have occurred, the proper warning is sent to the console to alert the operator. The last function of the job is to reschedule itself. The routine at the end of the jobstream again checks to see what day it is and then issues a stream command appropriate for the next run. The only thing that's left for your operator to do is hang tapes. This eliminates any mistakes that your operator might make while entering SYSDUMP dialog or even doing the wrong backup on the wrong day.

The partial backup routine should exclude database files. You won't want a partial database store if you need to recover and including these files will almost always take your partial backup into multiple reels. We'll talk about storing the databases in a moment. Also, keeping an eye on your LOG files will help keep your partial down to size. LOG files should be LOGSNAPed to tape on a regular basis.

So, now you have an unattended partial backup and a full backup that only requires changing tapes. It runs by itself and never needs to be restreamed.

Handling your databases will take some thought as to what you can afford to do. If you have some disk space you can get copies of your databases during off hours and do the backup at your leisure. This is the method I prefer as I can mount one tape before I go home and my partial gets done and copies of my databases are made that I can backup when I come in the next morning. If you don't have the space to make copies of your databases, you can still automate the backup procedure in the same manner that the full backup was done above. I would recommend using DBSTORE if you use this procedure as DBSTORE gains semi-exclusive access to the database before doing the store. This helps to insure that none of the database files gets left out of your store. You'll still have to mount tapes but you won't have to worry about anything else.

Copying databases can be done with "ADAGER", "MPEX" or other products. The job DBCOPY.JOB.SYS demonstrates how this would be done with "MPEX". The "BACKUP" account can then be stored to tape at a more convenient time using the STORE facility.

Another option is to use "IMAGE" logging. This way you would be able to get copies of your databases on your full backup and only need to backup the logfiles which you could let your partial do. Recovery becomes more of an issue because you'll have to restore a copy of the database and then update it from the logfile. Depending on your environment, this could cost a lot of time after you've got your system back up and running as you may still be functionally down if you can't access your databases. However, you will probably not lose any data. So weighing the cost of time to recover versus man-hours to re-enter data from a day-old database copy will determine the best method for you. One other point worth mentioning is where to place the logfiles. The best place would be on a private volume as this will not be affected if you should lose your system disc. If you lose the system disc, the directories stored there are also lost and you'll be doing a reload and you probably won't have an up-to-date logfile on tape.

The "Perpetual Backup" can be useful if you can split up your data fairly easily and you have a little extra disk space which you can afford to use temporarily. The way this works is by setting a schedule where you will make copies of different accounts (or groups if you only need a particular group such as a data group) on different days. You will then copy that backup account to tape when you arrive each morning. It is necessary to keep backup accounts that have the same basic structures as those that are being backed up. These accounts don't need all the capabilities that your live accounts do, but they do need all the same groups. Here is what a simple "Perpetual Backup" schedule might look like:

```
Monday Night

        Copy @.@.ABC to account ABCBKUP
        Copy @.@.DEV to account DEVBKUP

Tuesday Morning

        STORE @.@.ABCBKUP,@.@.DEVBKUP

Tuesday Night

        Copy @.@.PROD to account PRODBKUP

Wednesday Morning

        STORE @.@.PRODBKUP
```

A batch job can do these copies for you but you will need some type of utility to do @.@.account type copies. FCOPY can't use wildcards so something like "MPEX" is necessary. Some other points to be aware of with this type of backup scheme are: 1) Notice that the "SYS", "TELESUP" and accounts where third party software resides are not included in the backup schedule. These accounts do not change enough to make backing them up all the time useful. It would be best just to make sure to keep a current copy of them on hand and only back them up when they change, such as an update, etc. 2) No matter what type of backup scheme you choose, you should always have a current "COLDLOAD" tape, but it's even more important here because you never do a full or partial backup (with SYSDUMP) that includes all of "PUB.SYS". Make sure you include "PUB.SYS" on your "COLDLOAD"s. 3) You will still want to do a normal partial backup, however it will be modified to do different accounts depending on the day of the week so that files that have changed in one particular account or another get backed up in between full account backups. 4) Most importantly, if you choose this type of scheme, you have probably already figured out that your management of tapes is now extremely critical. You need to be very exacting in choosing the right tapes in the event of a system recovery. It becomes more of a matter than simply hanging the last partial and full backup tape sets, so a good Tape Management Library system is a must!

Backup Products

Another alternative is the use of a data compression backup tool. There are several on the market and they offer features such as reduced tape sets, faster backups, unattended backups to disc, backups over networks and some throw in some nice utilities.

Reduced tape sets does not always mean faster backups. Some of the performance gains by these products are achieved by balancing the processor to the particular tape drive. This is done by keeping the tape drive streaming so that it is not paused waiting for the processor to catch up. The decrease (or increase) in time to do a backup will actually depend on the cpu/tape drive combination for your shop. If your cpu already keeps up with your tape drive, you probably won't see any time savings even though it may cut your tape set in half. I have actually had my backup take longer on one system when using one of these products.

Unattended backups to disc usually are performed by compressing the data and writing it to a disc file which you back up when you come in in the morning. The first tape of the set may actually be written and the rest will go to disc.

Network backups will allow you to backup one system to another. This would be useful as long as your network was reliable and preferably the systems would not be in the same room. Mirroring databases is another useful aspect of network backups that is becoming very highly used to insure database integrity. Duplicate databases reside on two different systems and all transactions posted to the production database are also posted to the mirror copy at the same time.

Try to demo at least three different packages before you decide to purchase any system management software. There are many companies with products that have the same basic qualities but you'll want to examine your specific requirements, as well.

Job Scheduling

The most important piece to the automation puzzle is the ability to schedule jobs. Whether you use the AT,DAY parameters of the STREAM command or a scheduler, this will be your control center. It will command your system to perform all those tedious functions precisely when they're supposed to happen.

"IF/ELSE/ENDIF" AND "AT,DAY" PARAMETERS

This parameter of the STREAM command is the most inexpensive job scheduler around. When used, it places a job in the "SCHEDULED" queue. When the assigned day and/or time rolls around on the system clock, the job takes off. However, STREAM-scheduling all your jobs daily can be a tedious job in itself. There are a couple of ways to automate this. One might be a job that issues all the necessary STREAMs' to schedule your jobs. Another would be jobs that restream themselves. You'll have to examine both methods to determine which will require the least or easiest maintenance. Maintaining all your schedules in one master job keeps everything together, but if you have a lot of scheduling to do, the size of the file could get out of hand. Keeping each job's schedule within it's own jobfile will mean that you will need to keep track of the current version of the file, if there's more than one.

Whichever method you choose, you will use the IF/ELSE/ENDIF feature to determine when the next run should be. This use of IF/ELSE/ENDIF tests for conditions of day and/or time and when a "true" condition is found, the associated commands are executed.

Following is an example of IF/ELSE/ENDIF:

```
!SETJCW MONDAY = 2
!SETJCW TUESDAY = 3
!IF HPDAY = MONDAY
!  STREAM JOB1;AT=10:00;DAY=TUESDAY
!ELSE
!IF HPDAY = TUESDAY
!STREAM JOB1;AT=14:00;DAY=FRIDAY
!ENDIF
```

I was able to develop many of my ideas by reading through the JCW (Job Control Word) section of the System Management manual. There are several JCW's that can be very helpful when you need to check for various conditions or system status'. The system reserved JCW's such HPDAY, which was used above, are explained in the manual and these are the ones you'll use for status checking. If you're running an XL svstem you may also want to look at the documentation on System Variables. There are many of these which can be very useful as well as fun to play with.

What can I get from Job Schedulers?

Job Schedulers offer a convenient interface to the automation picture. Obviously, the basic component of a scheduler is the ability to launch a job at a given time. Schedulers range in price and sophistication. If all you want to do is launch jobs, it may be more economical for you to use the method just discussed above and have each job keep it's own schedule. However, if you want features such as job dependencies, job logging, job notification, schedule reporting, the ability to suspend jobs on holidays, or other interfaces to system operations tools, one of the schedulers on the market might be for you.

Assigning Job Dependencies

The ability to assign job dependencies is probably one of the most important features that a scheduler can offer. This is the ability to assign dependencies to jobs so that a series of job for a particular function will stop executing if a critical job in the series fails. Obviously, you wouldn't want a job to update a database if the job that creates the load file bombs. This may stop your automated process but it's much easier to fix a bug and continue where you left off, than to recover your data after a corrupt load.

Job Logging

Job logging can be extremely handy to help you keep an eye on job completions, complete or incomplete, successful or unsuccessful, and can provide a nice history and utilization resource.

Job Notification and Down Detectors

Job notification is a feature that can ease a great deal of phone calls from users who want to know if their jobs are finished yet. There are two companies that I have run across who market job notification and down detection systems that actually have the ability to phone users and deliver a voice message. These systems interface with the job scheduler and also are able to monitor other conditions such as down systems, computer room environment, network difficulties, etc. They have a phone directory which stores numbers for specific jobs or groups of jobs enabling them to report job completions and job aborts or simply when a job has completed, or failed to complete, certain steps. They also keep a phone list of support personnel for use in reporting down conditions. When a job notification or down condition occurs, the system dials the appropriate number from it's directory and delivers a pre-recorded message for the particular condition.

One of the major differences in these two systems is the hardware required. One vendor markets hardware and software which utilizes a P.C. as its control unit. While this may offer greater flexibility, it also means that a P.C. must be dedicated to the process. I don't know of many shops that have so much surplus equipment that they can afford to lose one of it's P.C.'s.

The other vendor, which is the one I used, markets it's own small box that contains all the necessary hardware. It uses a standard terminal port on the HP3000 and the 3000 acts as the processor. Phone lists are downloaded to the box so that it has the numbers to dial in the event that the 3000 goes down. It also contains a temperature monitor and, if plugged into the same power source as the 3000, will detect power failures. The software can be purchased in a networking version and non-networking version. The networking version, allows one unit to monitor several CPU's with one CPU acting as the host. The same vendor also markets a very nice job scheduler at a reasonable price that is also available in a networking or non-networking version. The two products interact with each other or can be used separately.

Managing Spoolfiles

Another tedious function, is that of dealing with spoolfiles. This is another area that can be automated. However, unless you have the time and inclination to write your own software, I would suggest that this would be an area where you would want to concentrate your efforts on looking for a spoolfile manager package. Many spoolfile managers are included in other system operations tool packages and that would be a plus when evaluating those packages.

Spoolfile Managers will do such things as scanning $STDLISTs for errors and printing the ones that do contain errors along with a document to aid in the correction process and purging $STDLISTs that don't have errors, adding header pages with user information like name, mail station, phone extensions, etc., and the ability for users to access their reports on-line and then print all or part of the report or not print it at all.

Many spoolfile managers are available in networking versions and are quite helpful for routing print to remote systems and printers. They offer a great deal more flexibility than "RSPOOL", but again, if all you want to do is move print, why spend money on elaborate tools.

The down detector that I've purchased and discussed above, has a spoolfile scan option available which scans $STDLISTs and reports errors. It also watches for old spoolfiles and can be configured to purge files that are so many days old.

In the course of writing this paper, I saw a demo of a spoolfile manager that intercepts the spoolfiles and sets them aside. This vendor's software then allows users to access their own reports online and look them over. Users can do searches through the report and find a particular part which interests them and also have the option to print only those parts. If your users are like most, they normally do not use all the data that we give them on paper, so this could save paper as well as the time it takes for your operations staff to tear down reports.

Automation and Multiple Systems

Automation of multiple systems shouldn't need to be anymore work than automating one system. If your systems are networked together, it's a fairly simple matter to choose one system to act as the host and set up procedures so that it can control most of the tasks for the other systems. Most operations management software is available in networking versions expressly for this purpose, thus eliminating the need to purchase multiple software licences.

A good example of how a host machine would function would be in the job scheduling area. You would use a networking version of a job scheduler on a single system. All jobs for the remote systems would be set up in the scheduler on the host and the host would then take care of launching the jobs on the remote system.

You could accomplish the same results by writing scheduling jobs on the host system using IF/ELSE/ENDIF and AT/DAY parameters of the STREAM command as discussed earlier, except this time have the job log on to the remote system to stream the job which is to be executed. By using a series of these scheduler jobs, you could build your own job scheduling system.

Other Little Things To Free up Your Time

Operator Menu Screens

Even though we've seen that we can trust our systems to run themselves and check for unusual conditions, there may be functions that you won't want to leave unattended. You can still make things easier and smoother by use of Operator Screens. These are menu screens that contain all the functions that you want performed by your operator. You can write you own with whatever means you use to develop software or you can write a system of screens that simply use the function keys. Function keys are loaded with escape sequences and the screen area can be used for information or help regarding the use of each key. There are usually a couple of programs on the swap tapes which deal with function keys, but you can also build a fairly nice system by using imbedded UDC's to perform the escape sequences and load the keys. This method of using imbedded UDC's was presented at a "Hewlett Packard AMS Seminar" which I attended, and can be made available to you if you desire.

Notifying Your Users of Special Events

One task that I always found quite bothersome is that of editing files to be displayed by means of the WELCOME function. The biggest issue here was to remember to change or create the file and to start it's display to the users and then stop it when it was no longer needed.

I solved this problem by writing a simple system to automatically keep track of my messages and begin and end their display at the appropriate times. I wrote my system in Powerhouse from Cognos, but it could be done in View or just about any other language or 4th G.L.. The principle of the system is to have a screen that gives me a window to enter the message I want to display and fields for start and stop dates. This is then stored in a database and each night at 12:05am a job runs and searches the database comparing the system date to the ranges of start and stop dates. When a record is found it is written to a file which is then displayed at logon time. A default record is also stored in the database and is used if there are no other records selected. The nightly job can then perform a "WELCOME' command using the file it's created. I discovered a program called "NEWS" on one of the Swap Tapes. I have this program run in the system logon UDC and it reads the file created by the nightly job.

This way I can leave my WELCOME message alone and use it to identify the system.

In summary, no one loses when you automate your data center. You have more time to spend on more progressive tasks. Your operations staff can assist you with more of system management chores thereby learning and building experience and furthering their careers. Your management will be impressed when they see how much smoother your operation runs and if you can save them money in the process by cutting out needless overtime and costly downtime due to errors, so much the better. So, if you haven't already automated your shop, give it a try. It's challenging, rewarding, educational and fun.

BKUPJOB.OPERATOR Job Stream

```
!JOB BACKUP,OPERATOR.SYS;HIPRI;PRI=CS
!COMMENT
****************************************************************
!COMMENT *  THIS JOB WILL AUTOMATICALLY DO A PARTIAL BACKUP ON
MON,
!COMMENT *  WED, THUR, AND FRIDAY.  IT WILL DO A FULL BACKUP ON
TUE.
!COMMENT *  BACKUP WILL OCCUR AT 1:00 AM IN THE MORNING.  USER
NEEDS
!COMMENT *  TO MAKE SURE TAPES ARE ONLINE BEFORE GOING HOME AT
NIGHT.
!COMMENT
****************************************************************
!SETJCW MONDAY=2
!SETJCW TUESDAY=3
!SETJCW WEDNESDAY=4
!SETJCW THURSDAY=5
!SETJCW FRIDAY=6
!IF HPDAY >= MONDAY  AND HPDAY <= FRIDAY THEN
!   IF HPDAY = TUESDAY THEN
!      CONTINUE
!      RUN ALLOWME.STREAMS
!      CONTINUE
!      LIMIT 5,1
!      CONTINUE
!      PURGE SJFILE
!      BUILD SJFILE;REC=-80,1,F,ASCII
!      CONTINUE
!      RUN FCOPY.PUB.SYS;STDLIST=SJFILE;INFO=":SHOWJOB"
!      CONTINUE
!      RUN ABORTJX.STREAMS
!      MAILOFF
!
TELLOP;*****************************************************
!      TELLOP; FULL BACKUP IS STARTING NOW
!       TELLOP; PLEASE INSURE THAT NO OTHER JOBS/SESSIONS ARE
ACTIVE
!       TELLOP; AND THAT LIMITS ARE SET TO: LIMIT 1,1
!       TELLOP;                  ***
!       TELLOP; CONSOLE I/O REQUEST FOR FULLDUMP PENDING
!
TELLOP;*************************************************************
!       FILE FULLDUMP;DEV=7
!       FILE SYSDLIST;DEV=LP,13,1
!       CONTINUE
!       SYSDUMP *FULLDUMP
        NO                       << NO CHANGES >>
        1/1/1                    << DATE FOR FULL SYSDUMP >>
        @.PUB.SYS,@.@.SYS-@.PUB.SYS,@.@.@-@.@.SYS;PROGRESS=2
        YES                      << LIST FILES DUMPED >>
!       IF JCW >= FATAL THEN
!
TELLOP;**********************************************************
!          TELLOP;FULL BACKUP FAILED
!          TELLOP;PLEASE CHECK JOB STREAMS TO DETERMINE ERROR
!
TELLOP;**********************************************************
!          CONTINUE
!          LIMIT 6,24
!       ELSE
!
TELLOP;**********************************************************
!          TELLOP;FULL BACKUP  &dBSUCCESSFUL
!
TELLOP;**********************************************************
!          CONTINUE
!          RUN DDATE
!          COMMENT - STREAM JOB FOR WEDNESDAY DAILY
!          IF DATESET = 0 THEN
!             STREAM BKUPJOB;DAY=WEDNESDAY;AT=01:00
!          ELSE
!
TELLOP;+++++++++++++++++++++++++++++++++++++++++++++++
!             TELLOP;BACKUP JOB NOT AUTOMATICALLY STREAMED
!
TELLOP;+++++++++++++++++++++++++++++++++++++++++++++++
!          ENDIF
!
TELLOP;**********************************************************
!          TELLOP;RESET LIMITS FOR NORMAL OPERATIONS, I.E. 6,24
!
TELLOP;**********************************************************
!          CONTINUE
!          LIMIT 6,24
!          SET STDLIST=DELETE
!       ENDIF
!    ELSE
!
TELLOP;*************************************************************
!       TELLOP; PARTIAL BACKUP IS STARTING NOW
!
TELLOP;*************************************************************
```

```
!       CONTINUE
!       RUN ALLOWME.STREAMS
!       CONTINUE
!       PURGE SJFILE
!       CONTINUE
!       BUILD SJFILE;REC=-80,1,F,ASCII
!       CONTINUE
!       RUN FCOPY.PUB.SYS;STDLIST=SJFILE;INFO=":SHOWJOB"
!       CONTINUE
!       RUN ABORTJX.STREAMS
!       FILE PARTDUMP;DEV=7
!       FILE SYSDLIST;DEV=LP,13,1
!       CONTINUE
!       SYSDUMP *PARTDUMP
        NO                  << NO CHANGES >>
        04/08/86      << **DCARD** DATE OF LAST FULL SYSDUMP >>
        @.@.@                       <<DUMP SUBSETS >>
        YES                 << LIST FILES DUMPED >>
!       IF JCW >= FATAL THEN
!
TELLOP;*****************************************************
!          TELLOP;PARTIAL BACKUP FAILED
!          TELLOP;PLEASE CHECK JOB STREAMS TO DETERMINE ERROR
!
TELLOP;*****************************************************
!       ELSE
!
TELLOP;*****************************************************
!          TELLOP;PARTIAL BACKUP  &dBSUCCESSFUL &d@
!
TELLOP;*****************************************************
!          COMMENT - STREAM BACKUP JOB FOR OTHER DAILY'S
!          IF HPDAY=MONDAY THEN
!             STREAM BKUPJOB;DAY=TUESDAY;AT=01:00
!          ENDIF
!          IF HPDAY=WEDNESDAY THEN
!             STREAM BKUPJOB;DAY=THURSDAY;AT=01:00
!          ENDIF
!          IF HPDAY=THURSDAY THEN
!             STREAM BKUPJOB;DAY=FRIDAY;AT=01:00
!          ENDIF
!          IF HPDAY=FRIDAY THEN
!             STREAM BKUPJOB;DAY=MONDAY;AT=01:00
!          ENDIF
!          SET STDLIST=DELETE
!       ENDIF
!    ENDIF
!ELSE
!    COMMENT - TRIED TO START ON WEEKEND, RESTART JOB
!    TELLOP;*****************************************************
!    TELLOP; TRIED TO START BACKUP JOB ON WEEKEND, RESTARTING JOB
!    TELLOP;*****************************************************
!    STREAM BKUPJOB;DAY=MONDAY;AT=01:00
!ENDIF
!EOJ
```

Stop Babysitting Your HP3000

Automating the Shop

- ❑ Job Management
- ❑ Spoolfile Management
- ❑ Automating Backups
- ❑ Miscellaneous Time Savers

Stop Babysitting Your HP3000

Why Should I Automate?

- ❑ Improve Productivity and Moral
- ❑ Reduce Errors and Inconsistencies
- ❑ Scheduling and Performance Improvements

Stop Babysitting Your HP3000

Automating Backups

- ❑ Backup Methods
- ❑ BKUPJOB.OPERATOR.SYS
- ❑ Backup Only What's Necessary
- ❑ Database Backup
- ❑ "Perpetual Backup"

Stop Babysitting Your HP3000

Job Scheduling

- ☐ HPDAY and AT/DAY Parameters
- ☐ Job Schedulers
- ☐ Job Notification and Down Detectors
- ☐ Managing Spoolfiles

Stop Babysitting Your HP3000

HPDAY and AT/DAY Parameters

```
!  SETJCW MONDAY = 2
!  SETJCW TUESDAY = 3
!  IF HPDAY = MONDAY
!      STREAM JOB1;AT=10:00;DAY=TUESDAY
!  ELSE
!    IF HPDAY = TUESDAY
!        STREAM JOB1;AT=14:00;DAY=FRIDAY
!    ENDIF
!  ENDIF
```

Stop Babysitting Your HP3000

Job Schedulers

- ❑ ***Ability To Schedule Jobs***
- ❑ ***Assign Job Dependencies***
- ❑ ***Job Logging***

Craig Conrad 4B

Stop Babysitting Your HP3000

Job Notification and Down Detectors

- ❑ *What do they do?*
- ❑ *Hardware Requirements*
- ❑ *Interface To Other Management Tools*

Stop Babysitting Your HP3000

Managing Spoolfiles

- ❑ *Scan $STDLISTs For Errors*
- ❑ *Purge Old Spoolfiles*
- ❑ *Dress-up Report Output*
- ❑ *Allow Users Access They Can Understand*

Stop Babysitting Your HP3000

Automation of Multiple Systems

- ❑ *Shouldn't Be More Work Than a Single System*
- ❑ *One System Acts As Host*
- ❑ *Build Your Own Scheduler with Scheduler Jobs*

Stop Babysitting Your HP3000

Other Helpful Ideas

- ❑ ***Operator Menu Screens***
- ❑ ***Notifying Users of Special Events***

Stop! Don't pick up the phone! Order Electronically!

by

Kathy Spanel

GBS Consultants, Inc.
6087 S. Quebec St., Suite 101
Englewood, CO. 80111
(303) 721-0770

Imagine, if you will, that you are in the business of receiving and processing orders. Not a difficult concept, really. Your customer picks up the phone, dials your number, speaks with an order processing clerk, and places the order.

The process is really quite smooth...except, its the busy season. And your customer often hears a busy signal or processing a three or four-hundred line order can take the better part of the day. And then it happens, your customer asks the inevitable question..."You know, we have a pc...couldn't we just dial up your system and transfer the order electronically?"...followed by a slight pause and then the final comment..."You know, your competitor can accept an order electronically...".

You cringe. You think..."the only real difficulty is transmitting the order" and realize that, in fact, several file transfer products exist for the HP3000. Piece of cake!

Or so you thought. A quick survey of your customers indicate they each have one of a number of Inventory Control systems each with their own flavor of communications software. It becomes obvious to you that this is a big hurdle. And once over that hurdle, there are many more...such as, what do we do with the order once its on the HP3000? How do we get the confirmation back to the customer? How do we inform the customer of specials that we normally communicate by phone?

With this paper, the many issues that arise in attempting to develop such a system will be addressed...issues such as:

- o Communications Software
- o Handling different formats
- o Processing the order thru your existing Order Processing software
- o Preparing a confirmation
- o Developing a Console to manage the Communications lines

In addressing these issues, the implementation of an existing Electronic Ordering System will be reviewed.

Perhaps before we get too far, lets take a look at how not to approach this task!

We began with a single customer who wished to place an order electronically. Just one... so why bother with an elaborate package when we might be able to put something together rather quickly and inexpensively though it may not be too sophisticated. This was our first mistake. Nothing can ever really be put together quickly and satisfy the needs of anybody... which is exactly what we found out!

Speaking with our customer's software vendor, we found that they communicated and transmitted orders via an XModem protocol. We had heard of XModem and had even seemed to remember reading about just such a utility as part of the Interex contributed library. After having located XModem, we felt we could design something suitable.

It was quite simple. When the user completes their order, they request to transfer the order. Their Inventory Control System dials and connects. The user then issues a standard Hewlett-Packard logon. An option logon udc handles the rest.

Well, not quite. Now that we have the order, what next? We certainly could not print the received order and have a clerk enter it as though it had been received as a standard phone order! We were in luck. Our order processing application has both a character mode and a block mode means of entering orders. We could conceivably use the character mode version. Our only problem was the need to convert the order to a format suitable for use with the character mode application. Through a series of report-writer routines, we were able to transform the transmitted order into one which could be processed with our character mode process. We breathed a sigh of relief. But, again, too soon.

What next? Our customer wanted us to send back a confirmation of the order, electronically of course! And, a special format was required. We again, put our report-writer to use and merely extracted the processed order and created a confirmation as required.

Our logon udc became a little more complex requiring the transmission of an order to us and then a transmission of the confirmation back to them.

We were holding our own. But problems soon arose.

o The character mode order processing application often returned errors if any unexpected conditions were encountered. Because it ran in the background, the error messages could not be responded to often causing the process to loop endlessly or abort.

- o A need arose to have the ability to request confirmation of an order only. To further complicate matters, the request for confirmation could be for a specific order or the last order transmitted.

- o We found that our customer had very little exposure to computers. Issuing the logon sequence was very foreign and difficult for them. Additionally, some of the software suppliers systems could not give the customer access to issue the logon.

- o Depending on the length of the order, the elapsed time between the receipt of the order and the transmission of the confirmation could be in excess of thirty minutes. Not good.

- o And, finally, the inevitable happened. Another customer wanted to transmit orders electronically. Then another. And another. The number was growing exponentially. Our little system could handle a few more customers with some modifications, but not the volume we were experiencing.

And, so we stepped back, re-examined our requirements, and began afresh. All was not lost. We did, in fact, learn much about our users, their software suppliers, communication software, order processing software, and the list continues.

How then should it be done? Lets now look at the key issues in implementing such a system.

Communication Software

The biggest challenge we faced was in the area of data communications. None of us being data communications gurus left us feeling a bit unsettled. Terms such as Enq and Nak were a bit foreign to us. But, as uncomfortable as we were, we were able to successfully understand the data communications requirements.

We owe our success in large part to the use of a data scope. If you are planning to embark on such a venture as ours, access to a data scope is imperative. Many hours of troubleshooting can be eliminated through its use. We, unfortunately, did not gain access to a data scope until well into the project. The data scope has now become a permanent fixture connected between one of our electronic ordering modems and an HP3000 port.

You will find that by the end of the project, you will be able to pass yourself off as someone who is comfortable with data communications. Throwing terms like Enq and Nak around will be second nature. Use of the data scope will be commonplace. You'll be a wizard! I'm getting carried away. You might not be a wizard, but you'll be close!

A quick survey of our customer's software suppliers found that they all consistently utilized an XModem protocol as the means of communications. Excuse me, did I say consistently? We found that, yes, XModem is the defacto standard in our industry for electronically transferring files. But, upon viewing the actual communications with a data scope, we found some slight inconsistencies.

A number of articles have been written regarding XModem communications. It is suggested that a review of these documents be completed in an effort to gain as much of an understanding of XModem prior to tackling any modifications.

For the most part, the existing version of XModem as it exists on the Contributed Library is adequate. It is able to transmit files, both downloading and uploading. Yet, we encountered a few hurdles.

First hurdle. As mentioned previously, our customers found it quite difficult to issue a logon sequence. More importantly, some of the software packages did not permit the customer access to the colon prompt. They merely press transmit and no further access is allowed. You may be wondering why we did not request that the software supplier modify their software to accommodate our needs. Primarily, the software suppliers did not feel it necessary to modify their software. Our competitors were able to accept electronic orders with no changes.

So, we designed a program, Courier, whose responsibility it was to direct the traffic on the electronic ordering modems. Our Courier initiates the XModem process. XModem then opens the port and begins posting Naks to the port. Unfortunately, if XModem does not receive data after a given number of attempts, it times out and terminates.

Our next hurdle. In our environment, we needed XModem to Nak the port continuously. When a customer dials in, their communications software would recognize the Nak and begin transmission. Some minor modifications to XModem permitted this type of operation.

Great. We had the order. Next hurdle. We needed a way of letting the customer know that we were processing the order and thereby keep the telephone connection. Here is where some inconsistencies surfaced. Some software suppliers required an Enq, some required an XModem packet with a message indicating the order was in process. The other complication was in determining what was required to keep the line up. We did not know which software supplier was transmitting the order until after the order was received. Another function for the Courier.

Upon receipt of the order, the Courier opens the order file, makes a determination as to the technique required to hold the line up, and initiates XModem passing a parameter specifying the technique. Another slight modification to XModem. Additionally, the Courier initiates another process to manage the processing of the order.

XModem then executes posting Enqs or packets to keep the line up. At the same time, XModem has a read posted on the confirmation file, a message file. When the order has completed processing and the confirmation is generated, the read on the message file completes and the confirmation is transmitted to the customer.

Upon completion of the transmission, the customer's process hangs up and XModem terminates. The Courier then turns the line around by initiating XModem to receive an order.

The cycle is complete.

This implementation of the Courier allows for the greatest throughput on the electronic ordering modems. Because the processing of the order is managed by another process, it remains independent of the XModem process. In the event, our customer disconnects prior to receiving confirmation, the XModem process terminates and the line is turned around in preparation to receive the next order. The processing of the prior order received will complete independent of the receipt of the next order.

Different Formats...Preparing the Order

By now you may be asking yourself "Why did they not just specify to their clients the required means of transmission, the required format, etc.?" Ideally, yes, specifying our requirements to our clients would be far less complicated. If you have this option, by all means take it. In our environment, we were at the mercy of our clients. Remember, virtually all of our clients have no programming staff and are generally not well versed in formats, specifications, etc. And, so, a means had to be developed by which we could accept virtually any order format.

Here we have to make a choice. Shall we require that the process actually placing the order be intelligent enough to handle multiple formats or shall we utilize a pre-process to convert from a clients format to yet another standard format (but, this time, one we have defined!). We chose the later.

Converting to a standard format does have some drawbacks such as the overhead incurred by the conversion itself. But, the benefits of later processing the order from a standard format, particularly simplifying the process, far outweighed the drawbacks. In addition, some pre-processing could be completed such as verification of customer number, ship-to information, etc.

How does one go about specifying a standard format? Remembering that we are attempting to simplify the actual order processing, perhaps selecting a format that supplies the answers to the order processing prompts is conceivable.

Initially, you may consider that the actual conversion of the order to the standard should be parameter driven to enable new formats to be implemented quickly and easily. However, we found that the formats varied so radically that the feasibility of defining conversion parameters was not likely. Therefore, we coded a series of routines to convert from a given format to the standard format. Should a new client present a new format, we merely code a new routine.

The outcome, then, is all orders formatted identically.

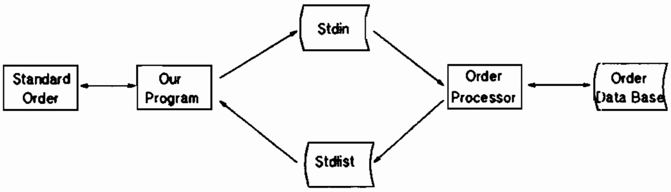
Processing the Order

We now have an order to be processed. We also have an existing order processor. Actually, we have two... one which processes orders via VPlus/3000, the other via character mode. Because we felt the order would be processed in the background, the VPlus/3000 version was out. The character mode version had potential.

The biggest challenge in using a character mode, interactive process is to programmatically respond to the questions, error messages, warnings, and informative messages much as one would were they actually processing the order interactively.

Now, I may have gotten ahead of myself. First things first. It occurred to us that we needed a means of sending information to the order processor and receiving the results. The concept of redirecting StdIn and StdList suggested itself.

Consider the following. We write a program which reads our standard formatted order. This same program initiates the order processor as a son process directing its StdIn and StdList to two message files. We then read the questions from the order processor's StdList file and post our answers to its Stdin file and process the order. At the same time, our process can update the order to reflect if the item was ordered, backordered, out of stock, etc..



Perhaps an even bigger challenge than that mentioned before (we're full of big challenges in this project!), is knowing all the possible questions the Order Processor may ask. Here, documentation and/or source code is your only resource.

Generating the Confirmation

Yes, we are on the home stretch. We now need to merely generate a confirmation file to be transmitted back to the user. After all we have accomplished thus far, this seems relatively insignificant.

We chose to think of this process as merely the reverse of our Order Preparation. You can be assured that your clients will expect the confirmation in a certain format. We merely convert our standard formatted order which was updated while processing the order to the desired format. It is important that all the information required to generate the confirmation be captured at the time of Order Preparation and/or Order Processing.

At this time, it may also be desired to capture such information as the customer number, sales order number, and confirmation file name for statistical purposes as well as should the client later request re-confirmation.

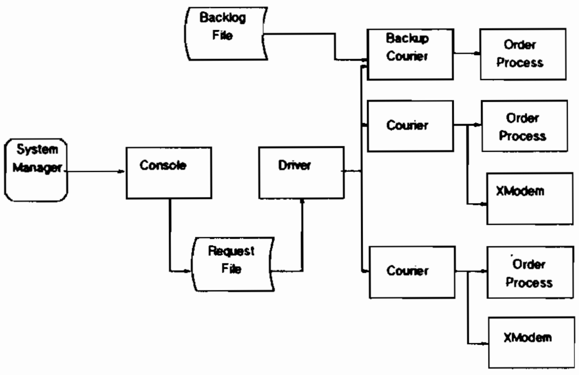
A Console... Tying it Altogether

The one thing that seemed to be missing is a means by which we could easily identify the activity of electronic ordering. Also, most data communication products, which this could most certainly be considered, have some means by which to start, stop, review status, and review errors encountered.

And, so, a "console" was developed. Through the console, electronic ordering can be initiated and terminated. Also, all the various processes post a status message to the console's status file so that via a console command it can be displayed that an order is being prepared, another processed, and perhaps another is being transmitted. Additionally, all the various processes post any errors encountered to the console's error file so that another console's command can display any errors encountered such as Invalid Customer Number. Finally, the confirmation preparer posts a message to the console's order file identifying the customers and sales order numbers processed so that at quick glance it can easily be identified the volume of orders being processed.

The Console communicates these requests from the electronic ordering System Manager to the Driver, another process that merely "drives" the entire process. Requests to Start and Stop Couriers are posted to the Driver request file and executed by the Driver. The Driver executes as a batch job.

The following is a diagram depicting the major components of Electronic Ordering.



Other Considerations

One of our initial concerns, among many, was how do we alert our electronic ordering customers to specials, new products, and general tidbits. Our phone ordering clients were easy. We verbally pass along the information. But how do we do this electronically?

We found that most of the confirmation formats specified by our clients allow for a text message. This appeared to be our opening. We could format a message and include it as part of the confirmation. How do we allow for the message to be easily changed? Via the console. We devised a new command allowing for the editing of the message. The message can then be included as part of the confirmation.

Another issue was availability, Ideally, we wanted our clients to be able to transmit orders 24 hours a day. Realistically, they could with the exception of system backups. Also , ideally, we wanted our clients orders to be processed 24 hours a day. Realistically, they could with the exception of again system backups and any batch processing requiring exclusive access. But, once again, how do we alert our clients that their order will be processed at a later time and to check back for confirmation? Once, again, via the console. We devised two new commands, OFF and ON, which do just that... turn processing of orders off and on. Prior to exclusive processing, access is turned OFF, and then turned ON later. Prior to processing the orders, then, we are able to check if access is OFF or ON. If off, a message (again customizable via yet another console command) is returned to the user as the confirmation.

Finally, we wanted to provide other means by which orders could be processed. For example, in the evening we receive via RJE an order from a customer. Unlike our XModem ordering customers, this customer transmits the order and disconnects. Approximately an hour later, we initiate the transfer of the confirmation. This process is managed by a "backup" Courier, a Courier that merely initiates the processing of the order and does not manage the electronic ordering modems. This same "backup" Courier can also manage orders that are transmitted when access to the order process is turned off. Orders are captured in a backlog file and then later processed by one or more "backup" Couriers.

The End Result

How did we make out?

Looking back, many questions come to mind. Perhaps the system or at least a portion should have been designed for a personal computer. Perhaps all the formatting and reformatting of orders and confirmations is excessive and results in wasted disc space and i/o overhead.

We wished to achieve a goal of one-hundred customers ordering electronically. We have far surpassed that goal with the number of clients increasing exponentially. As well, electronic ordering now accounts for more than one-third of the sales volume.

You be the judge.

Structured Online Program

by Terry Warns
WL SOFTWARE
1895 Crooks Rd
Suite 120
Troy, MI 48084
(313)-641-0550

Introduction:

This paper shows how we can structure an online program using COBOL and VPLUS in order to reduce development time and the maintenance effort.

Problem:

Most computer systems today are developed with the end user using a terminal and entering, updating or inquiring upon data. Although we still need to produce reports and have batch activities, the focus of new development is online transactions.

I wrote my first online program in 1978 on an IBM mainframe using CICS. People told me it was impossible to write a structured program and indeed I found it difficult. In the early 80's, I was told the same about COBOL/VPLUS programs. It seemed many people felt online and structured programming was like military intelligence; the terms just didn't go together.

Because online programs are not structured and tend to be spaghetti code we find that online programs are difficult to write and maintain. In fact some people use 4GL's to avoid this problem.

At the same time, we find that managers require two years of experience before hiring anyone. They do not have the time or resources to train new programmers. Colleges that are training programmers rarely show them how to write online programs. They are difficult to write and more difficult to test or grade. Yet most new systems require a great deal of online programming expertise.

At WL SOFTWARE we write COBOL/VPLUS programs for a fix price. We needed a way to reduce our development time and our maintenance effort in order to be profitable.

This paper presents WL SOFTWARE's solution to create structured online programs. This technique is so successful that we feel a new programmer without experience can create and maintain online program after one month of experience. There are three areas to consider. First is the user presentation. Second is the programs interface to the screen. Third is the program design and structure. We will be assuming VPLUS but many of the ideas also apply to character mode screens.

User Interface:

The first consideration is to determine what are the features that the average user will require on every screen. The list WL SOFTWARE developed is the following:

1. REFRESH the screen
2. Print the screen image immediately
3. Exit from the screen
4. Provide a multi page help facility that is maintainable by the users themselves
5. A calculator
6. access to MPE (ie SHOWJOB)
7. access to E-MAIL
8. Hot switch to other programs
9. A notepad, scheduler, and meeting reminder

All of the above features should be provided to every screen without the application program knowledge or concern. Also other features could be added without changing the application program.

The above contains nine functions but on the HP there are only 8 function keys. We needed a way to allow these functions on every screen but also allow for application specific function keys. We decided to toggle our function keys so at any time the user can switch between the application and non application function keys. Now I have 16 function keys for every program; 10 for common functions and 6 for applications. This could easily be expanded to 24 or 32 function keys if your applications require it.

We made some assumptions in order to simplify WLSCREEN. First we assumed all forms for a given program are in one forms file. The name of the forms file is FORMFILE. Second we assumed the HELP text would be in an editor file with the formname as the prefix and HELP as the group name. We wanted to encourage users to write help screens in their own terminology. Third the print key would print the screen immediately rather than hold all print requests until VPLUS was exited.

E-MAIL, the HOTKEY, and the reminder functions are not yet implemented. We feel that WLSCREEN will check your mail box every time WLSCREEN is called. If mail is waiting the mail will be presented in the particular E-MAIL's normal presentation prior to your screen being presented to you. Within any application you can send a message to another user. WLSCREEN would need customizations for your particular EMAIL being used. One problem is to relate the logon user to the E-MAIL user.

The reminder function has similar problems about identification. Also we need to establish a database to store the reminder information. Both the EMAIL and the reminder would need to be checked on a regular basis.

The HOT KEY function will work with the users MENU function. Any program on any menu should be able to be instantly brought up without losing the current application information or screens. We need an interface to the organizations menu system.

Many of these functions are available on PC's today. In order for minicomputer software to survive in the 1990's, we must provide the same functionality that is common among users. At the same time we must reduce the development effort and maintenance costs in order to compete with PC software.

PROGRAM INTERFACE

With the common features defined we set about a program design to separate the application from these features. The features will be in every program and we wanted these features to be transparent from the application.

At the same time we saw our people struggle with VPLUS and found that we could not predict either the effort to develop a screen program, debug a screen program, or maintain a screen program. We needed a way to simplify the VPLUS interface and provide the user with the maximum functionality possible.

We began the development of a subroutine called WLSCREEN. The goal of WLSCREEN is to provide the 9 common functions for every screen no matter what screen being used and provide a simpler programmer interface to VPLUS. We did not want any calls to VPLUS except through WLSCREEN. When we called WLSCREEN we provided the screen buffer and WLSCREEN returned a fully edited screen buffer. All the VPLUS routines and the implementation of the above 9 features are taken care of by WLSCREEN.

The following is the call to WLSCREEN in a COBOL program:

```
      CALL "WLSCREEN" using VPLUS-WORK-AREA
                            SCREEN-BUFFER
                            WLSCREEN-WORK-AREA.
```

The VPLUS-WORK-AREA is defined as the work area that VPLUS requires. The calling program initialize this area to low-values at the beginning of the program and never references the area again. Of course this area would be passed to subroutines that interface to the screen. The screen buffer is the buffer area that VPLUS uses to pass data to and from the screen.

The basic assumption here is that WLSCREEN will return only the function keys and fully edited screen buffer as defined in VPLUS field edits. The application program works only with the WLSCREEN-WORK-AREA.

The following is the COBOL layout of WLSCREEN-WORK-AREA.

```
01  WLSCREEN-WORK-AREA.
    05  WLSCREEN-FORM-NAME          PIC X(16).
    05  WLSCREEN-MESSAGE            PIC X(80).
    05  WLSCREEN-CURSOR             PIC S9(4) COMP.
    05  WLSCREEN-FKEY               PIC S9(4) COMP.
    05  WLSCREEN-ERROR-TABLE.
        10  WLSCREEN-ERROR-OCCURS OCCURS 20 TIMES
                                  INDEXED BY WL-EX.
            15  WLSCREEN-ERROR-FLD PIC S9(4) COMP.
    05  WLSCREEN-LABEL-TABLE.
        10  WLSCREEN-LABEL-OCCURS  OCCURS 6 TIMES
                                   INDEXED BY WL-LX.
            15  WLSCREEN-LABEL     PIC X(16).
    05  WLSCREEN-CHANGE-TABLE.
        10  WLSCREEN-CHANGE-OCCURS  OCCURS 10 TIMES
                                    INDEXED BY WL-CX.
            15  WLSCREEN-CHANGE-FLD  PIC S9(4) COMP.
    05  WLSCREEN-CALCULATOR-AREA    PIC X(40).
    05  WLSCREEN-USER-NAME          PIC X(8).
    05  WLSCREEN-GROUP-NAME         PIC X(8).
    05  WLSCREEN-ACCT-NAME          PIC X(8).
    05  WLSCREEN-TERMINAL           PIC S9(4) COMP.
    05  WLSCREEN-FKEY-TABLE.
        10  WLSCREEN-FKEY-OCCURS    OCCURS 7 TIMES.
            15  WLSCREEN-FKEY-IND   PIC X.
```

We will describe the function of each field.

Field: WLSCREEN-FORM-NAME:

This field denotes the form that is used by WLSCREEN on this call. WLSCREEN compares this form to the current form and switches forms if necessary. This field has a tendency to flatten the screen hierarchy. It is now much easier to switch between screens.

Field: WLSCREEN-MESSAGE:

This field contains the message for the message line whether the message is an error message or not. There is only one message line on the screen so we have only one way to put the message onto the screen.

Field: WLSCREEN-CURSOR:

This field contains the field number where we wish to place the cursor. If there is an error the cursor will be placed in the first field that is in error.

Field: WLSCREEN-FKEY:

This field contains the function key (0 is for ENTER) that is being return to the application. The current version supports 6 function keys and the enter key but can easily be expanded to support more function keys without changing existing applications. We assume only a valid function key will be returned by WLSCREEN.

Field: WLSCREEN-ERROR-TABLE:

This table allows up to 20 fields to be denoted as an error. The application program denotes only the fields in error. Twenty was somewhat arbitrary but we felt if you had more than twenty errors on the screen that nothing was really correct anyway. The error message will be put in the WLSCREEN-MESSAGE field. A zero field number indicates that the table occurrence does not have an error. Most times we move low-values to the table before we begin editing in the application. We also assume that a VPLUS defined errors and movements have already occurred. These errors are application specific.

Field: WLSCREEN-LABEL-TABLE:

This table allows you to override the screen function labels. WLSCREEN currently reserves F7 and F8. If the label is space we use the screen defaults. This is useful when you want to change function keys.

Field: WLSCREEN-CALCULATOR-AREA:

We store the calculator results in a work area so you do not lose the values you are working with between screens. The application can do nothing with this field.

Field: WLSCREEN-USER-NAME, GROUP-NAME, ACCT-NAME, TERMINAL:

These fields are needed many times in application programs for audit purposes. Instead of programming each application to retrieve this information we allowed WLSCREEN to do it. Now this information is available for every screen.

Field: WLSCREEN-FKEY-TABLE:

This field indicates to WLSCREEN which function keys are valid to return in WLSCREEN-FKEY. The values are E,I or V. E is for edit the data prior to returning which implies update the screen buffer. V for a valid function key but do not update the buffer. I is for an invalid function key. If this function is used by the user WLSCREEN will provide the user with an INVALID FUNCTION KEY message. WLSCREEN will NOT return to the application program. This is an easy way to turn on and off valid function keys.

If you are familiar with VPLUS many of these fields relates closely to VPLUS intrinsics, but it is much easier for a programmer to provide a formname, a message, and the cursor position than it is to call up to 22 subroutines in the correct sequence. Consequently the programmer can concentrate on the application and not even concern himself with the VPLUS.

STRUCTURE ONLINE PROGRAMS:

Now we can look at the structure of the online program given this subroutine. First we start at the WLSCREEN call.

```
      CALL "WLSCREEN" using VPLUS-WORK-AREA
                            SCREEN-BUFFER
                            WLSCREEN-WORK-AREA.
```

Since the program will continue to display the screen until the user provides an indication he wishes to leave, WLSCREEN is in the middle of a loop. We assume that prior to WLSCREEN you have set up all the parameters and wish to display a screen to the user. The first thing we do is put WLSCREEN in a loop such as the following:

```
      PERFORM UNTIL WLSCREEN-FKEY = 8
         CALL WLSCREEN
      END-PERFORM.
```

Our assumption is that WLSCREEN will return to a screen only valid data for our application and specificly ENTER or F1 thru F6. Also we structure the user interface so that each function key is a unique action requested by the user. Therefore you may expand the program as follows:

```
     PERFORM UNTIL WLSCREEN-FKEY = 8
       CALL WLSCREEN

       EVALUATE WLSCREEN-FKEY
        WHEN 0  PERFORM 1000-EDIT-UPDATE
        WHEN 1  PERFORM 1100-F1
        WHEN 2  PERFORM 1200-F2
        WHEN 3  PERFORM 1300-F3
        WHEN 4  PERFORM 1400-F4
        WHEN 5  PERFORM 1500-F5
        WHEN 6  PERFORM 1600-F6
      END-EVALUATE
    END-PERFORM.
```

The effect of this structure is that now the programmer has broken down his big online program into 7 smaller programs and each can be written and tested separately. The programmer can concentrate on what he is to do on F1 rather than any VPLUS calls. Instant productivity.

We expand the above structure to allow for moving easily between multiple screens by evaluating the last form used.

```
     PERFORM UNTIL WLSCREEN-FKEY = 8
       CALL WLSCREEN

     EVALUATE WLSCREEN-FORMNAME
       WHEN "GL"
         EVALUATE WLSCREEN-FKEY
          WHEN 0   PERFORM 1000-GL-EDIT-UPDATE
          WHEN 1   PERFORM 1100-GL-F1
          WHEN 2   PERFORM 1200-GL-F2
          WHEN 3   PERFORM 1300-GL-F3
          WHEN 4   PERFORM 1400-GL-F4
          WHEN 5   PERFORM 1500-GL-F5
          WHEN 6   PERFORM 1600-GL-F6
         END-EVALUATE
      WHEN "AP"
         EVALUATE WLSCREEN-FKEY
          WHEN 0   PERFORM 2000-AP-EDIT-UPDATE
          WHEN 1   PERFORM 2100-AP-F1
          WHEN 2   PERFORM 2200-AP-F2
          WHEN 3   PERFORM 2300-AP-F3
          WHEN 4   PERFORM 2400-AP-F4
          WHEN 5   PERFORM 2500-AP-F5
          WHEN 6   PERFORM 2600-AP-F6
         END-EVALUATE
   END-EVALUATE.
END-PERFORM.
```

In order to move from the GL form to the AP form all the programmer has to do is move "AP" to WLSCREEN-FORMNAME. We found this feature alone had a tendency to eliminate screen hierarchies and provide us a easy mechanism to switch between screens, another feature that users are very interested in. We found that adding a another form into a program was trivial and adding a another function key was painless also. With many forms in a program the control loop could be large but our structure allows it to be manageable.

The performed paragraphs sets up all the WLSCREEN work area prior to its completion. For example, you are displaying a series of detail records for a given master record. You define F6 as NEXT PAGE and F5 as FIRST PAGE. When you reach the end of the detail chain you make F6 invalid by moving "I" to WLSCREEN-FKEY-IND(6). With this function key marked I for invalid, there will be no way you can perform the next page paragraph. WLSCREEN will display to the user INVALID KEY. Your application program effectively eliminated reading past the end of chain.

This permits every performed paragraph to be truly structured and the controlling paragraph meets structured programming rules. We have a completely structured online program.

We found the following using WLSCREEN:

- New programmers can easily write online programs in short time
- Add new forms and functions quickly
- Maintenance costs are reduced
- Users happy with consistent screen interface
- We can add new global features to all screens without reprogramming every application
- We feel we can port WLSCREEN to other operating systems

I contributed WLSCREEN to the swap tape along with WLFORM, WLTABLE, and WLTBLSC. WLFORM is a forms file that provides forms for WLTABLE and the calculator. WLTABLE is a program that uses WLSCREEN so you can see the structure. WLTABLE maintains a series of tables in a table database. WLTBLSC is the database schema for WLTABLE. We have WLSCREEN working on both Classic and XL machines. WLSCREEN is written in COBOL and uses only documented intrinsic.

THE SYSTEM MANAGER'S TOOLBOX

ISAAC BLAKE
City of Tempe, Arizona
Information Systems
120 East Fifth Street
Tempe, AZ 85281 3797
(602)350-8218

INTRODUCTION

Being a system manager is one of the most challenging jobs in data processing. The typical system manager is a person who has to handle a varity of problems and be a jack of all trades. A system manager has to address areas such as performance, data communications, security, operations, telecommunications, technical support, resource planning, training, programming, and the list goes on and on! The SM is often placed in a situation where he has to be a doctor, mechanic, firefighter, and even a magician. This paper will attempt to cover ways for a SM to be able to resolve problems quickly when they come up and hopefully before they happen!

WHY A TOOLBOX?

The concept of a toolbox will be a place where we will put items that we need for our job and site. Looking at an actual doctor and mechanic, they have their own "toolboxes", the doctor has his medical bag, and the mechanic has his shop tools. Granted your toolbox probably will be too large for you to carry, but the main point is to have it, and, in a place where you can get to it quickly.

As I cover items that you should consider placing in your toolbox, remember one thing, this is YOUR toolbox for YOUR site. I can't possibly cover every possible item that you'll need. So make sure that you add items that you specifically need, and items that you have discovered from other users, publications, or from experience.

ORGANIZING YOUR TOOLBOX

The first thing we need to do is to break our toolbox down into seperate areas. I'm going to suggest we break it down in the following manner:

Immediate Use

These will be items that you will need instantly to resolve a problem. This should be considered your hot list. Frequently this is where a lot of time is spent and lost trying to find something. Some good examples are your dialup modem numbers, passwords, configuration, and hardware serial numbers. Many times when I have been on the phone with a customer trying to resolve a problem with them, it has taken them five to ten minutes to get this information. I point this out because that is time you could use doing something else.

Frequent Use

Items in this area will be ones that you will need on a day by day basis. Reference manuals, hardware tools (like screwdrivers, wire cutters, or maybe a hammer for really bad problems!), $STDLIST's of jobs, console logs, etc...

Seldom Use

Just about everything else that you need. These items are the ones that you will only need every so often. Your DUS (Diagnostic Utility System) tape is a good example. Hopefully you don't need to run something like SADUTIL on a frequent basis, but when you have a problem and need it, you shouldn't have to spend 15 minutes to an hour finding it!

BASIC ITEMS FOR YOUR TOOLBOX

So now that we have defined and layed out our toolbox, let's look at what might be a basic list of items to put in it. Remember as a System Manager you are called upon for a wide range of things, so you will need items covering areas such as hardware, software, administrative, and misc. Most of these items will be included in your "Immediate" area of your toolbox.

* Current System Configuration: Have a listing from SYSDUMP, or, SYSINFO in the TELESUP account, showing your I/O configuration, table sizes, etc. Also include your own list of what devices are attached to particular devices. You can be creative, setting up device classes such as HP7933, HP7978, LASERJET, etc... which can help with this. Make sure to include what type of terminal you have hooked up to a port. Know your system so that you can say it's a 3000 series 42XP with 4MB of memory on T-delta-4. I realize that it can be confusing, but there is a lot of difference

between a series 44, 48, and 58, or, series 64, 68, and 70. Also, MPE V is just an internal layout, so know what V.UU.FF you're on, or the name like Q-delta-1, V/P, V/E, T-MIT, etc...

* Network layout: Even for the simplest system include what type of ports you have (ADCC vs ATP, 25 pin vs 3 pin), wiring layout, how the cables are labeled (they should be), Ldev's and phone numbers for modem(s), and type of equipment. For other networks, include the type of network (DS, DS/X.25, MRJE, MTS, LAN), how it's connected (direct connect, dial, leased line, satellite), the type of equipment in use (Stat Mux, port selector) and the vendor. If you have external lines, make sure you have the circuit numbers, locations, and type for each line.

* Latest TELESUP version: This will help the HP people resolve your problem and give you some nice utilities to use. Alot of valuable time can be lost because the version of TELESUP you have is outdated. For example, it makes it difficult to troubleshoot a problem with the new EAGLE drives if the version of CS80UTIL you have on your system doesn't support it, or, trying to look at a U-MIT virtual memory dump of a system failure with an old IDAT. As of this writing the latest version of TELESUP is 2620 (May 1986), you can get this version of TELESUP from your account CE.

* Reference materials: Nothing is more frustrating than having a problem you know that you can resolve, if only you could find that cursed manual! Put your basic HP manual set, publications, device reference manuals (terminals, printers, modems), crib notes, Communicators, SSB's, RC Application Notes, and so forth in one place. Have a checkout for these materials so that you may track them down if they are not there. Remember, alot of time can be saved, if for example, you have a problem with a ThinkJet printer and you have the manual handy to verify the switch settings.

* Administrative items: A good example of this type of item would be the serial number(s), device type(s), contract number(s), and coverage hours of your hardware. Other items would be your accounting structure, PICS handle, vendor service telephone numbers, etc...

* Hardware tools: The basic items like screwdrivers, extension cord, loopback connectors, breakout box, gender changers, and any other site specific components belong there.

* Software Utilities. Gather and learn the numerous utilities, programs, and features that HP, the user groups, and vendors put out. There are a lot of these excellent

utilities out there to assist you on a daily basis to make your job easier.

Other items worth having are: ColdLoad tape, DUS tape, Formatted floppies, Dump tapes, and maybe even prayer beads along with a few magical chants!

USING YOUR TOOLBOX

It's important that when you or someone else uses items from the toolbox that they are returned. When I first tried this approach it worked great, but as time went by, a lot of the items were no longer in the toolbox. So make sure to periodically inspect and account for the items. The items that come up missing most of the time are the tools that people borrow from you, and the reference material.

As you start and continue using your toolbox take the time to re-evaluate it and the items in it. Every time you get into a situation where your toolbox comes up short, look at what you need to add. Basically maintaining the toolbox will be an ongoing process.

PREVENTIVE TOOLS

If you can reduce the chance of a problem occurring before it happens, you won't be in the hot seat as much. Often our jobs are so busy that it seems like we can't take the time to do a little preventive work. However, the time spent in doing this will usually result in fewer and less complicated problems. I'm sure you can see that working in a proactive mode is better than a reactive mode.

A good example of some Preventive Tools is HP's Predictive Support and HPTREND. Predictive Support (P.S) is a set of utilities that when you run them will go out and check your hardware. It utilizes the system I/O error logging, CS80 drive logging, and memory logging. If P.S detects a problem or potential problem with your hardware it will automatically call the Response Center to log the problem. HPTREND on the other hand, runs in the background and collects global performance information. Every quarter this information is transmitted to the Response Center, then a report is produced and mailed to you. Both of these systems will oftern allow a problem to be detected before it goes critical and fails.

Other Preventive Tools are ones like TUNER, FREE, and VINIT. TUNER will allow you to check your table sizes on a running system. By checking this on a frequent basis (weekly), you can monitor how full your table sizes are getting, and schedule time to change them if they are becoming full. This will allow you, for example, to increase the SwapTable at your convenience rather than when you get a System Failure 602. FREE and VINIT are good tools for checking and managing your disc freespace. By using the condense feature of VINIT you will reduce the fragmentation of your disc and increase your largest free areas. Then by using FREE you can monitor your discspace and reduce the change of running out of disc and/or spool space. Also make sure to check the freespace of Ldev#1 frequently, especially BEFORE you do a COLDSTART or UPDATE. You must have at least 17,000 contiguous sectors on Ldev#1 for these operations to complete. If you don't, you can run the risk of running out of disc space, and a possible RELOAD!

There are many additional tools to help you with performance, datacomm, security, etc... Look at your site and what problems are facing it, then go out and get these tools and add them to your toolbox.

I've found that one of the best tools you can use is information. Almost always, the quickest way to resolve a problem is a problem that you can fix yourself. Take the time to document potential and/or possible problems your users might experience, then train them on how to take care of them. It takes very little time to sit down and write a document to help a user to check out a terminal that's not working, and it will save you the time it would take to find out that the terminal isn't in REMOTE MODE or unplugged. Here at the Response Center we have been sending out the Application Notes and RC Q&A's for awhile, the program has met with great success and positive feedback. Try the same in your shop.

OTHER TOOLS

As you can tell by now, your toolbox is getting probably getting quite full! Before we run out of room, let's take a look at a few more tools worth mentioning.

* Resources: Identify who in your shop can be used as a resource and what for. There may be a person who just loves to do something like cabling (well maybe just likes to...), by keeping track of this you can easily find these people when you have a need. Also by cross-training people, you make it so that you are not the only one to resolve a problem.

* Publications: Interact, HP Chronicle, SuperGroup, Conference Proceedings, and so forth are a great wealth of knowledge. Look especially at the columns where users write

in talking about problems and war stories, this is where you can find out about potential new and/or old problems.

* What if?: Spend time developing plans for unusual situations. For example, what are you going to do when you come in Monday morning and find the computer room temperature at 100+ degrees, or, where is the fire extinguisher if you have a fire. It's always easier to think and plan ahead, rather than trying to figure out what you are going to as it's happening.

* SMBWA: Spelled out is: System Management By Wandering Around. I have found that this is a great way to find problems before they blow up. Go around and ask your users how they are doing. Often this information will let you know in advance of upcoming problems and changes. It also acts as a reality check.

* Magic Dust/Stray Cosmic Particles/etc...: This stuff is great when you HONESTLY have no idea what happened, or how you fixed it. It's very frustrating for a user to come to you with a problem, then you do the EXACT same steps they did, but it works! Then it's a good time for the Magic Dust!

* YOU!: You are your own greatest tool! The more experience you get, and the more time you spend with your system, the better you'll start getting the "feel" of your system. Many System Managers, although they can't explain it, often detect problems with their systems, even though they had no facts. Gut feelings is a great tool. Make sure to invest time on a daily basis to keep in touch by reviewing the Console Log, SMBWA, etc...

SUMMARY

I'm sure by now that your toolbox is full, if not, it will be shortly as you start finding by experience what you need in it. It's success greatly depends on YOUR involvement with it, and your system. I feel that by using your newly formed Toolbox, that you will find you are able to resolve problems quickly, and for some problems, before they ever become problems!

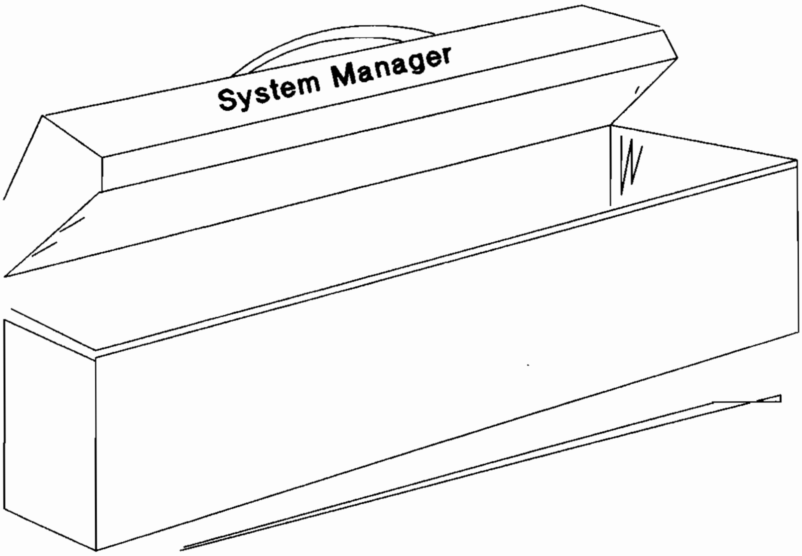
BIOGRAPHY

Isaac Blake is currently the System Manager and Technical Support Coordinator for the City of Tempe, Arizona. This includes the support of the 15 HP3000 computer systems, 300+ PC's, and networks.

His experience covers 17 years in the computer industry, with the last 13 years working on HP3000 systems. During this time he has gathered extensive broadbase knowledge of the HP3000 family most notably in System Management, Performance, Security, Operations, MPE, Migration, and User Support.

After working for HP as a Senior Support Engineer at the HP Western Response Center, he returned to the user community in 1987, and was elected to the INTEREX Board of Directors. As a Reserve Police officer, he has dealt with the investigation and prosecution of computer related crimes, and is a frequent speaker on Computer Security as well as the numerous articles he has wrote on this topic.

System Managers

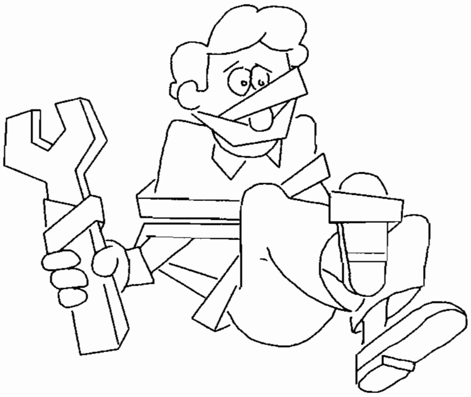


Toolbox

3118-8

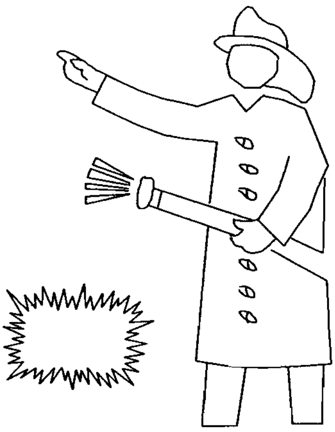
Is this You???

3118-9



Or maybe this...

Why a TOOLBOX?



This is YOUR TOOLBOX for YOUR SITE

ORGANIZING YOUR TOOLBOX

Immediate Use

Frequently Used

Occasionally Used

?

Basic Items for YOUR Toolbox

Current System Configuration

Network Layout

Latest TELESUP version

Reference Materials

Administrative Items

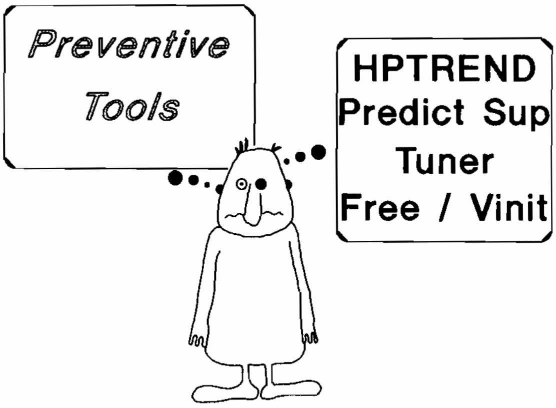
Hardware Tools

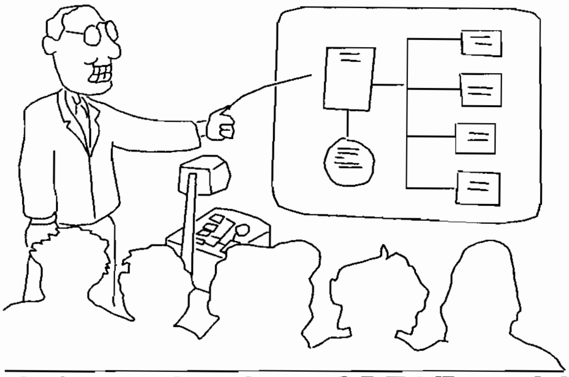
Software Utilities

REMEMBER

The time that you save...

3118-15


Preventive Tools
HPTREND Predict Sup Tuner Free / Vinit



Information is a GREAT tool !

OTHER TOOLS

Resources

Publications

What if ?

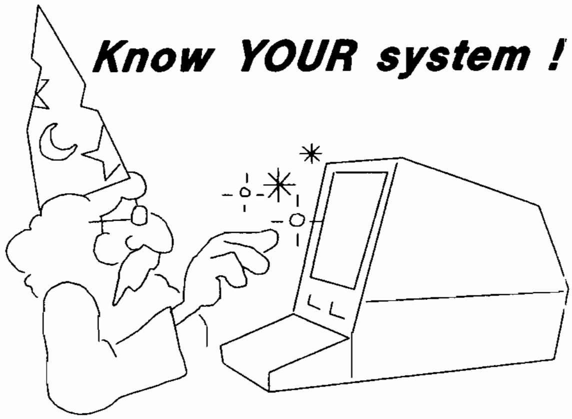
S.M.B.W.A.

Magic Dust

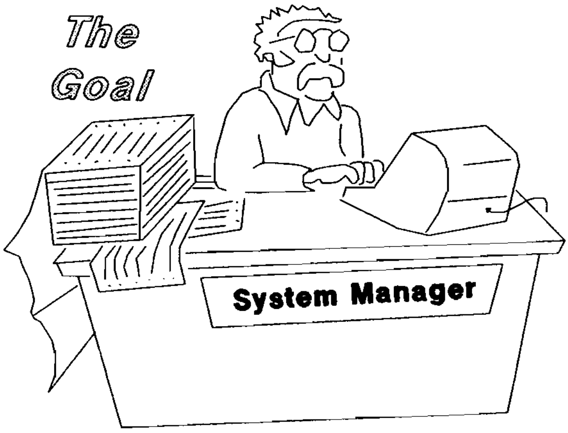
Stray Cosmic Particles

YOU!!!

Know YOUR system !



3118-19


The Goal
System Manager

TITLE:	System Performance Through Queue Tuning

AUTHOR: Warren Bunnell

ORBIT Software (USA),. Inc.

319 Diablo Road, Suite #218

Danville, CA 94526

415-837-4143

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 3119

TITLE:	System Performance/Management For
	Technical/Nontechnical/Personnel
AUTHOR:	Dave Jennings
	Facer-Sys. Performance Division
	PO Box 9802-231
	Austin, TX 78766
	512-440-8488

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 3120

Table Your Data Base!

Mike Mitchell

Hewlett-Packard Company
Vancouver Sales Office
10691 Shellbridge Way
Richmond, British Columbia V6X2W8

Summary

A table is the basic structure in a Relational Data Base model. Programs using this model execute their transactions with tables. On the surface, my title is a command to move to a Relational Data Base Model. It is, however, ambiguous. It is really enjoining you to chart three different courses for the use of Relational Data Base technology.

Webster's dictionary defines the verb "table" as, "to put forward for consideration." The command is to consider the possibility of adopting a Relational data model. On the other hand, to table is "to postpone a decision indefinitely." The command is a command to stay with the status quo. It is a call to inaction. Finally, to table is to "tabulate." The command becomes an imperative: "Go Relational. Now. Completely."

This paper examines Relational Data Base management using HP's ALLBASE/SQL. It provides sufficient background information to allow us to address the question implied by the title: What shall I do with my Data Bases? We do so in a real-world manner suggesting a rational course of action.

Introduction

Most computer based business applications rely on some formalized kind of Data Base management system or DBMS. If you own an HP3000 the chances are pretty good that you make use of the TurboIMAGE Data Base. If you aren't using a relational Data Base management system (RDBMS) you are probably under some pressure to do so. But, what IS a relational Data Base? Specifically, what is SQL? This paper will show you the "feel" and "taste" of SQL. It will allow you to answer the question implied by the title: What should I do with my Data Bases?

Data Base Models

The challenge of data management is that of modeling a real world situation, such as employees working for a company, into the abstract world of the computer. Not only do we need to reduce the employees to some kind of machine representation, we need to track their salaries, their skills, their current assignments and so on. Historically this was handled by flat files or keyed sequential files (often called indexed files) with a logical record structure imposed over them. For example, the first 2 bytes of a record are its "type", the next 20 are "surname", and so on. Individual programs were responsible for the interpretation and maintenance of the files. In effect, they owned the data. If the record structure had to be changed, a number of programs would be affected.

The situation became messier when separate data processing applications within a company required the same data. For example, a personnel department requires complete employee information while the project management group requires only employee skill level and work assignment data. While there was some data duplication, each software application had its own data files specific to its particular mission. Application maintenance was difficult, since a simple change in field width or record composition in one application resulted in inconsistent data in another.

The issue of data redundancy and inconsistency and the resulting application maintenance burden motivated the development of data base management systems. The vision was to provide system wide data management software that could be used by all software applications. The DBMS was to shield application programs from the invariable changes to data structure and content.

Three Data Base models are common in the business world today. They are presented here in chronological order of their development.

Model One: The Hierarchical Model

The first widely used commercial DBMS sought to model the real world in a hierarchical fashion. This was particularly well suited to organization management. The distinguishing mark of a hierarchical Data Base is that each node or "child" has only one "parent." A parent node can have may child nodes. Importantly, the parent-child relationships or paths are determined at Data Base design time and are not easily changed in a production environment.

Model Two: The CODASYL Network Model

The first standardized data model extended the hierarchical model by removing the restrictions of each child with only one parent. In a CODASYL network Data Base the relationship between parent and child nodes is many to many. Hewlett-Packard's TurboIMAGE is an example of a 2-level network model Data Base. As with the hierarchical model, the parent/child relationships are determined and fixed at design time. An application user needs to navigate through the fixed structure to get his data. A number of end user reporting tools make this easier, but limitations are still present.

Model Three: The Relational Model

This model takes an entirely different approach to the other two. The concept of child and parent nodes connected by fixed paths determined at design time gives way to a more intuitive implementation. The relational Data Base is seen as a series of tables. They are manipulated using the mathematically based operations of set unions, intersections and restrictions.

In the relational model, the relationship between different data is determined at inquiry time rather than design time. Access to the data is simplified. The data base navigation issues of the previous two models is replaced by a simple specification of what data is required. The relational Data Base (RDBMS) itself is mandated to figure out how to get the data. The resulting flexibility is appealing to data base users, programmers, designers, and Data Base Administrators.

Are These the Only Choices?

Other data management models are being developed. Their mission is to provide practical management of huge data pools. The resulting data management systems may be based on Entity-Relationship Models, Object-Oriented Data Bases or Hypermedia. A catch all DBMS, referred to simply as an Intelligent Data Base, has also been suggested. These systems are largely experimental. We will not go into any detail here, since currently they lack the commercial viability and availability of the relational model.

Taking a Closer Look: SQL and ALLBASE/SQL

History

In 1970 E. F. Codd published a paper which has become the foundation for the Relational Data Base model. Data is presented as a series of tables. They are manipulated using set oriented operations. Data relationships are determined at query time. The first commercial relational Data Bases became available in the early 1980's. Throughout the early 1980's, the Relational Data Base model did not generate a lot of excitement in commercial circles, primarily due to performance considerations. The tremendous flexibility and elegance of the model was not conducive to high transaction volumes. It doesn't matter how elegant the underlying data model is -- "same day response" for most queries is not acceptable in an on-line environment.

Throughout the 1980's significant research was carried out in the area of RDBMS performance and optimization. For example, ALLBASE/SQL, available on MPE/XL and HP/UX has had significant performance improvements over its last three major releases. These are due primarily to enhancements in the SQL optimizer and increased concurrence in a multi-user environment.

Structural Characteristics: Base Tables and Views

There is only one significant data structure in ALLBASE/SQL: the table. Tables seen by an end user or application program are of two kinds: base tables and views. Base tables correspond to physically stored data, while views are virtual tables. A view does not physically exist. Although is presented as a table, only the view definition is stored in the database. A view can be made up of one or more tables joined together on a common column value. The tables which make up a view can be either base tables or existing views. A view can also present a sub-set of a table to the end user. Thus with a view we can hide sensitive data such as employee salary levels.

Both base tables and views behave the same way. An inquiry program, for instance, does not need to know whether the query it is executing is founded on base tables or views. In fact, a Data Base Administrator could change a base table to a view with no impact on production software.

The phrase "only one significant data structure" has some subtle implications. Security, access groups, users, table definitions, view definitions, table dependencies and programmatic accesses are all stored in an internal dictionary called the System Catalog. The System Catalog is no more than a collection of SQL managed tables and views. A properly authorized Data Base user or program can inquire as to the structure of the Data Base using the same selection commands used to process application data.

Structural Characteristics: Files and Disks

ALLBASE/SQL on MPE/XL is physically made up of a number of files. The files are MPE/XL PRIV files, just as with TurboIMAGE. ALLBASE/SQL files are NOT, however, TurboIMAGE files. ALLBASE/SQL is a fully autonomous data management system. PRIV files are used since they are immune to operating system commands such as PURGE, RENAME or PRINT. Their content is thus secured.

The names given to MPE/XL's SQL data bases are arbitrary. Even within a Data Base, the individual file names are arbitrary. The actual data files are listed in the SQL System Catalog. Individual SQL files may contain System Catalog information, table data, index data or a combination of all three. The decision of file name and content is left to the Data Base Administrator. The sizes assigned to the data base physical files are also arbitrary. An SQL data base can start small and have additional disk space added to it as application requirements demand.

Relational Terminology

As with any specialized field, a variety of terminology and jargon has emerged. We've already used the terms Table (or base table) and View (or virtual table). Here are some others:

Column	A single data item within a row. This is like a field or item.
Join	Concatenating one or more tables together. The SELECT command provides this capability through the specification of column names and the WHERE clause.
Normalization	The process of removing structural inconsistencies in Data Base design. There are several stages of normalization. They are called first, second, third and forth normal forms. Additional flavors of normalization have been suggested.
Primary Key	The column or columns that will uniquely identify each row in a table. This should not be confused with an index, which is used for fast on-line access.
Referential Integrity	DBMS enforced logical inter-dependencies between tables and columns.
Relation	A fancy name for a table.
Restrict	Taking a subset of columns from a table. The SQL SELECT command provides this capability through specification of column names.
Row	A single line in a relational table. This is like a Record in traditional file processing. A row is also called a tuple in some of the literature.
Select	Taking a subset of rows from a table. In SQL the SELECT command provides a superset of this capability.

Data Manipulation in SQL

A data manipulation language (often called the DML) is the mechanism a Data Base management system provides to allow selection, creation, deletion and updating of data. TurboIMAGE uses a suite of intrinsics or system calls to accomplish this.

There are ten DML intrinsics in TurboIMAGE. In SQL there are four: SELECT, INSERT, DELETE and UPDATE. For example, assume we have two tables EMPLOYEE-INFO and EMPLOYEE-RATE:

EMPLOYEE-INFO Table	EMPLOYEE-RATE Table
Surname	Employee Number
First Name	BDU Rating
Employee Number	BDU Description
Salary	

These tables represent only a fragment of a hypothetical personnel application. Since our desire here is to show the primary SQL data manipulation commands, the application has been greatly simplified.

For clarity, the SQL commands are presented in upper case, table and column names are presented in lower case. SQL doesn't make such a distinction.

Selection of data from these tables is as follows:

```
SELECT
     surname, firstname, bdu-description
FROM
     employee-info, employee-rate
WHERE
     employee-info.employee-number = employee-rate.emp-number;
```

If you wanted to add data to the EMPLOYEE-INFO table, you would execute the SQL INSERT command:

```
INSERT INTO
     employee-info
VALUES
     ('Mitchell','Mike',38228);
```

Deletion of data requires the DELETE command. In this example, all employees with a BDU-RATING less than 5 are removed from the database:

```
DELETE
     employee-info
WHERE
     bdu < 5;
```

Changing existing data requires the UPDATE command:

```
UPDATE
     employee-info
SET
     bdu-rating = 8
WHERE
     emp-number = 38228;
```

Note that there are no physical restrictions as to which columns may or may not be updated. Traditional restrictions regarding updating indexes or primary keys are removed. Security constraints may restrict the execution of any of the SQL data manipulation commands.

Data Definition in SQL

The definition of the example tables would be like this:

```
     CREATE TABLE
          employee-info
          (surname              CHAR (40),
          firstname             CHAR (20),
          employee-number       INT
          salary                INT);

     CREATE TABLE
          employee-rate
          (employee-number      INT,
          bdu-rating            SMALLINT,
          bdu-description       CHAR (80) );
```

These again are simplifications. A number of SQL-enforced constraints could have been applied to the tables and their columns. These include the designation of NULL or NOT NULL columns (for the non-existence of column values), default values, column value uniqueness, primary key columns, foreign keys (for table inter-dependence) and check conditions (for restrictions on actual column values).

Embedded Data Base Calls and a Host Language

On an HP3000 running MPE/XL, the preceding examples could have been entered on-line, using an interactive tool called ISQL. ISQL is like a mixture of TurboIMAGE's QUERY and DBSCHEMA, but far easier to use.

These examples need not have been entered interactively at all. They could have just as easily been embedded into (say) a PASCAL program with only minor syntactic changes.

ALLBASE/SQL supports a number of programming languages including C, PASCAL, FORTRAN, and COBOL. Programmatic support of SQL is radically different than a non-relational DBMS such as TurboIMAGE. Recall that TurboIMAGE uses an intrinsic or procedure oriented calling mechanism. For example, looking up a particular employee's record would involve at least two

TurboIMAGE calls, DBFIND and DBGET. If the host or application language was PASCAL, the calls might look something like this:

```
     dbfind (base, list, data, status_array);
     if err_cd <> 0 then begin
          beam_up ();
          crash_and_burn := TRUE;
     end
     else begin
          dbget (base, list, data, status_array);
          if err_cd <> 0 then begin
               writeln ('Could not get your data!');
               .
               .
               .
          end;
     end;
```

On the other hand, if the host language was COBOL the calling syntax and flow control would have to adhere to COBOL conventions. If C or FORTRAN was being used, other syntactic changes would be required.

SQL uses an embedded syntax. Contrast the previous calls with the same task in SQL (example is written in PASCAL):

```
     EXEC SQL
          SELECT
              surname, firstname, bdu-rating
          INTO
              :surname, :givenname, :bdurate
          FROM
              employee-info
          WHERE
              employee-number = :emp_number;
```

The syntax of the SQL call is identical for each host language. Error checking can be made implicitly. The preceding example could have just as easily been taken from C, FORTRAN, COBOL or even a fourth generation tool such as HP's ALLBASE/4GL.

Any of the host language compilers would choke on SQL syntax. The embedded calling mechanism only works because an extra step is added to the program compilation process. A preprocessing step is carried out prior to host language compilation. The preprocessor's mandate is to produce legal host language code. The "EXEC SQL" tells the SQL preprocessor to begin translation into the appropriate host language. If there are no SQL errors, the preprocessed version of the program is passed into the host compiler.

Transactions: Logging and Recovery

A transaction, in data base terminology, refers to a logical unit of Data Base work, for example, the addition of an employee. In our example, an employee's existence affects at least two tables. In our example, the "add an employee" transaction requires two INSERT's be done to the data base, one to EMPLOYEE-INFO and one to EMPLOYEE-RATE.

Many data base management systems offer an optional transaction logging scheme. With ALLBASE/SQL you have no choice. You can't NOT log. As a Data Base Administrator or application designer you can elect to have two identical log files maintained. You can choose whether you want archival logging or not. The choices you make affect your data recovery options. You can not, however, decide to forego transaction logging.

SQL automatically starts a logical transaction when you "touch" the Data Base, for example, with a SELECT command. You may explicitly begin a transaction with the BEGIN WORK command. At any point in the life of the transaction you can issue a COMMIT WORK or a ROLLBACK WORK. COMMIT WORK makes any changes you have made to the data base permanent and visible to other users. ROLLBACK WORK automatically undoes all changes since the transaction began. If the application program you are running fails, or if the system itself crashes, a ROLLBACK WORK is automatically carried out by SQL.

ROLLBACK WORK is very powerful. For example:

```
     DELETE employee-info;
```

will empty the EMPLOYEE-INFO table. A subsequent SELECT would tell you that there are no records to look at. If you then issued a ROLLBACK WORK, the contents of EMPLOYEE-INFO would be restored.

Data Navigation and the Optimizer

Traditional Data Base Management Systems require you to specify the mechanism for connecting different components of the Data Base. This is called navigation. The navigation process can be complex if there are a large number of data items and sets or if inquiries tend to be adhoc. In TurboIMAGE, for example, you would specify the key items for each detail set. Then, you'd have to connect different data sets together in (hopefully) an efficient manner. Considerations include availability of keys and sizes of data sets involved. The efficiency of existing keys themselves is variable, depending on such things as master set capacity and the physical location of the data in the data sets. A number of end user oriented tools seek to optimize the data selection process in order to provide a good level of performance. A typical data processing application lacks the sophistication to take the all the variables into account.

With SQL, the Data Base User says WHAT and the DBMS figures out HOW, based on the current make up of the Data Base. If an index is added to a table, SQL will automatically reoptimize any Data Base calls, in order to ensure the most efficient access path.

By moving the responsibility of optimization from application programs several benefits are realized. First of all, every Data Base call is optimized. Data processing applications are not at the mercy of their development programmer. Program maintenance is simpler, since tricky coding techniques introduced for Data Base optimization are removed. Every program automatically benefits from system level enhancements to the DBMS optimizer. No recompliations are required. Again, the maintenance task is simplified. Finally, the optimizer allows a Data Base Administrator (DBA) more flexibility in carrying out the care and feeding of the Data Base.

Life as a Data Base Administrator

The administrative task with SQL is primarily one of monitoring and tuning. The Data Base Administrator (DBA) has to monitor disk space consumption and application use. Additional disk space can be added to an SQL data base incrementally and dynamically. Incrementally, in that you only need to add disk space as it's required. Dynamically, in that you can add disk space while the data base is actively being used by other applications and users.

As a DBA you can create or remove indexes dynamically. An index can potentially speed inquiry access at the cost of slightly increased INSERT and UPDATE times. The beauty of the data model is seen in that the existence or absence of keys makes no difference to application programs. There are no restrictions on updating a column which is part of an index. Indexes are merely available as a performance consideration. They are used or not used based on decisions made by the SQL optimizing software independent of specific application programs.

Since SQL uses SQL to manage itself, and since activities such as index creation and deletion can occur dynamically, these activities can be easily incorporated into application programs. A smart application program could create indexes prior to month end reporting and remove them when it's done. It would gain the "look up" advantages of an index without impacting other applications. Naturally, as a Data Base Administrator, you'd need to measure the performance with and without the indexes in place.

As alluded to above, part of the data base Administrator's role is to consider performance. Some of the considerations and tradeoffs include:

> Many indexes for fast accesses vs UPDATE and INSERT overhead.
>
> Commonly used tables could be combined into one physical file to take advantage of disk data transfers
>
> Indexes can similarly be combined with other indexes or with their associated tables. Performance benefits would be realized depending on the physical locality of the data.
>
> Indexes can be specified as CLUSTERED. For a clustered index, SQL tries to physically locate the data in the indexed table in key value groupings.

As you can see, there are a number of considerations. There is great flexibility. One of the major values an RDBMS offers is the increased flexibility over traditional models for the data base Administrator.

What about 4GL Tools?

Recall the purpose of this paper. We want to answer the question: What shall I do with my Data Bases? Our focus is the engine, not the chassis (i.e., no fourth generation language (4GL) tools are being discussed here). 4GL tools have made it extremely easy to use any underlying data model. Even flat files provide an acceptable data storage structure given a sufficiently sophisticated fourth generation language tool and a sufficiently small amount of data. The advantages of a relational data model with a 4GL are seen at design time and in high volume production, as previously discussed.

Table Your Data Base!

I have provided background information intended to give you the "feel" and "taste" of SQL. Although a lot of details have been left out, we have shown that it is a more flexible Data Base Management System than the traditional models. The flexibility is seen by End Users, Data Base Administrators and Application Programmers.

We are now in a position to address the original question. It was chosen specifically because it was ambiguous. That ambiguity reflects the most sensible course of action.

First of all: Table your Data Base! Some existing applications can be made relational. As your current software inventory depreciates and ages, consider rewriting parts of it using the relational model. If you are using TurboIMAGE, consider that the SQL interface into TurboIMAGE is now available.

Secondly: Table your Data Base! Leave it alone! A large number of existing applications are based on TurboIMAGE. They work very well. Adhere to the maxim: "If it isn't broken, don't fix it." Many third party turn key applications fall into this category. They are well thought out, have excellent end user and administrative tools, and they deliver a lot of performance.

Finally: Table your Data Base! If you have a relational Data Base like ALLBASE/SQL and some new applications to develop, use it. Use the best current tools. If you don't have a relational Data Base, get one, even a small one. This will allow you to understand the technology before committing a large number of resources to it. There are some very reasonably priced PC-based SQL products. This will allow you to try a small application (albeit PC-based) and begin developing some SQL skills.

What shall I do with my Data Bases? To sum it up in a sentence: Open your eyes; know your tools; make sensible choices: Table your Data Base!

The Automation of Software Testing

by Susan Maxwell
Hewlett-Packard Company
3410 Central Expressway
Santa Clara, CA 95051

ABSTRACT

Testing is a necessary evil, required to assure the quality of the software before it is released. It is also time-consuming and difficult. As a result, quality is often sacrificed to meet the schedule.

One solution to the problem of inadequate time for testing is to automate the testing process. In recent years, several testing tools have hit the market. This paper describes opportunities for automating the testing process, the functionality to look for in a testing tool, and what additional features must be available in the software development environment to automate the entire process. This paper also describes what the current tools cannot do for you, and may never do.

Software test automation can be a real benefit to the organization that understands its limits and its value. This paper lays the groundwork for that understanding.

INTRODUCTION

Testing gets a lot of attention from project managers as a problem area. It is the last hurdle between the developers and the date of completion. If any previous phase of software construction took longer than estimated, testing becomes the phase that risks the project's deadline. As a result, testing time may be sacrificed, and quality may suffer.

This paper describes how some aspects of the testing process can be pulled up in the schedule, to parallel the software development process, and the time for the actual execution of tests can be minimized with automation, so that good quality and timely delivery of software can be achieved.

THE TESTING PROCESS

The software testing process has four phases, similar to the those in the software construction process: a design phase, an implementation phase, an execution phase, and a maintenance phase. Each phase of the process is described below.

THE DESIGN PHASE

The design phase of developing a software test is usually embodied in a test plan. A test plan specifies what values are to be input to the software, and what results the software is expected to produce. This includes values that are at the limits of the application, values that the application is likely to see, and values that are outside of the application's limits.

Designing the test can occur, and really should occur, along with the design of the software being tested. As the software specifications become more concrete, so does the test plan. The actual test data can be prepared while the software is still vaporware, long before the software construction is complete.

Compiling the testing data at this early stage has added benefits. By looking more closely at how the software is expected to perform, holes in the software design may be discovered. If the software's behavior is undefined in response to certain input, this will show up in the detailed test plan. The hole in the design can be fixed at that point by the design team, rather than by a developer during implementation, who is ignorant of the overall design. The test plan also provides additional data for the initial schedule estimates -- more detailed data about the functionality requirements for the implementation phase, and the complexity of the testing phase.

In cases where the software specification is allowed to change, or encouraged to change due to feedback from users, some rework of the test plan may be necessary, but the benefits of producing the test plan during the software design phase should outweigh the cost of the rework.

THE IMPLEMENTATION PHASE

Implementation of the software test can occur in parallel with the implementation of the software. Implementation involves setting up the test data and the execution environment. This includes creating the test databases, creating the test input files, building the test environment from which to call procedures, and writing the scripts for any user interaction.

During the software implementation, the test plan, like the software design, should be kept up to date. For example, as developers write code, their knowledge of the input required to hit special paths, like those added for performance reasons, should be incorporated into the test plan.

THE EXECUTION PHASE

Execution of the test involves putting the software through its paces, and checking to see that the software behaved as expected. Putting the software through its paces is what most developers think of as testing -- actually running through the test data or the user interaction script. However, it is only half of the third phase of testing.

The other half of the execution phase, checking the expectations against the actual results, is one of the main reasons for testing to begin with. It answers important questions: Did I fix the old bug? Did I introduce another bug? Did someone else -- another developer or a new subsystem -- introduce a bug? The accuracy of the answers to these questions directly correlates to two aspects of the test process: the completeness of the test plan and its implementation, and the accuracy of the comparison of the expectations against the results.

THE MAINTENANCE PHASE

The tests for a piece of software are maintained just as the software is maintained. As new features are added to the software, new tests are added to exercised those features. As with the development of the software, there is a cycle through design, implementation, and execution for these new tests.

Even if the software itself does not change, the subsystems or operating systems or even the hardware platform may change around it. Some alterations may be needed to accommodate these changes. (Note that without execution of tests, these external changes would probably go unnoticed, until they caused the software to fail.)

Alterations in the tests or in the expectations of those tests, must be managed, just as software changes must be managed. Fortunately, the same tools used to manage software versions can be used for tests.

SUMMARY OF THE TESTING PROCESS

If there has been no design phase prior to the execution phase, executing the tests becomes "banging on the keys" -- playing with the features of the software and seeing if it breaks. Testing without a test plan may catch some bugs, but it has definite drawbacks:

No record is kept of what features have been tested, or how thoroughly they were exercised.

The same parts of the software are often exercised over and over again, wasting valuable testing time.

Without clear expectations of the results to be achieved, detection of bugs relies on immediate judgement of the human observer; defects may slip by unnoticed.

Since the testing is random, once a bug has been found, it may not be reproducible, and may therefore go unrepaired.

Even if a bug is found and fixed, there is no guarantee that it will not reintroduced in another version of the software; there is no process to insure that the next session of banging on the keys will cover the same functionality as the previous one.

Clearly, all phases of the testing process are necessary for testing to be valuable. The next sections of this paper detail each phase, and point out opportunities for automation in each.

AUTOMATING THE DESIGN PHASE

The primary deliverable of the testing design phase is the test plan. A complete plan covers the range of input the software will receive, and the expected results the software will produce. The types of input and results described in the test plan vary, depending on the type of software being tested.

INPUT VALUES AND EXPECTED RESULTS

When testing procedures or subprograms, such as those destined for an SL, XL, or COPYLIB, the values described in the test plan are those of the parameters and global data referenced by the procedure. This includes data read in from databases, or retrieved from intrinsics, other procedures or library routines. The expectations include the effects on those parameters, global variables and databases, as well as any value returned by the procedure itself. If the procedure affects system variables, disc files, the display, or other peripherals, the test plan includes those effects and input producing those effects.

When testing an interactive program, the values described in the test plan include all types of interaction with the application. Depending on the type of user interface, interactions may include the function keys, the fields on each form, the ENTER key, the options on pull-down menus, radio-buttons, check boxes, the individual commands and their parameters, mouse button movements, and all of the individual keystrokes recognized by the application. The expected effects may include new values in the fields, error messages, informational messages, the appearance of new menus, the disappearance of old ones, changes in color, sounds, and cursor movement.

The values passed to a batch application include the commands and data present in its input file(s), including any parameters such as the PARM and INFO string on the MPE RUN command, as well as any file equations, JCWs or other system variables used to control the operation of the application. The expectations include the output file(s) created, the state of the system variables, and any other output, such as might go to $STDLIST on MPE.

GENERATING INPUT VALUES

One way that the design phase of testing can be automated is to generate the input values. This does not mean generating every possible input value, but values within, at, and outside the limits for that particular variable or interaction. For example, if an application is expecting to read an integer between one and ten, the input values generated include a number between one and ten, one, ten, a number less than one, and a number greater than ten.

Generation consists of three phases: determining the different inputs to the software, determining the limits applied to that input, and then generating data values which fall within, at, and outside of those limits.

There are several potential sources for input values and their associated limits. When a dictionary is used to define all of the data structures in an application during the design phase of software development, it contains the range of values that those data elements are allowed to have. A schema or data definition file contains similar information for values coming in or out of a database used by the application. The data declarations, COBOL copy libraries, and include files also contain information about the range of values the data elements are allowed to have. User interface definition files, such as VPLUS formsfiles, are another source of limit information.

If a dictionary is not used to develop or design the software, and the software does not access a database, and the software does not use a user interface package, the information available depends on the language used to code the application. Languages such as ADA allow you to include a lot of information about the data types, but with older languages such as COBOL and Fortran much of the limit information is embedded in the code:

```
if x < 00000 or x > 99999
     then error
     else retrieve the record
```

Extraction of limit information becomes difficult to automate in these cases. It requires not only an understanding of the syntax of declarative statements in the language, but the language's entire syntax. When dictionaries, databases, and user interface packages contain the limit information, it is relatively simple to massage the data in their reports to extract the information needed (see **DATA REFORMATTING TOOLS** in the **<u>AUTOMATING THE IMPLEMENTATION PHASE</u>** section).

LIMITATIONS OF DATA GENERATION

Tools which generate test data as described above are inherently flawed. They are based on the implementation of the software specification, not the software specification itself. If the software omits a limit, the test values produced do not check the limit:

```
if x < 00000
     then error
     else retrieve the record
```

This omission could be in the design, as well as the implementation. The input values generated from the data specifications, the code, and the design documents are only as good as those sources. Reviewing the generated data is still needed to uncover flaws in the software design and its implementation.

In addition, even in the best case above, we have only automatically extracted the appropriate input values. Unfortunately, the expectation, the second half of designing the test, cannot be extracted from data definitions. It is tied to the logic of the software. To automatically generate the expectation is to implement those semantics -- the equivalent of writing the application automatically from the specifications. This aspect of the test plan cannot be automated.

BENEFITS OF AUTOMATING THE DESIGN PHASE

Although the test plan cannot be generated completely automatically, there is a definite benefit to automating the parts that can be automated. Since the plan is tied directly to the data definitions, changes in those data definitions can be easily incorporated into the test plan.

Also, the data definitions are likely to be complete, and therefore produce a more complete test plan.

AUTOMATING THE IMPLEMENTATION PHASE

Once the input values have been generated, they must be put into a format usable by the application. If the test plan is written or generated in a regular format, general data reformatting tools can automatically generate the test data and execution scripts needed for execution of the tests.

DATA REFORMATTING TOOLS

There are a number of data formatting tools available to help with test implementation. Today's screen editors have powerful macro capabilities, allowing users to massage data from one format to another using sets of editing commands. These macros can be created initially by "recording" a set of editing commands as they are executed in the editor, and can be edited to add logic or to delete erroneous commands.

Tools from the UNIX development environment are now available on the PC and other platforms: the file transformation utility AWK and the batch editor SED allow users familiar with C and regular expressions to massage data using scripts for this tools. SED scripts are essentially sequences of editor commands. AWK scripts are more like C programs, but with a simpler syntax.

Any of these tools can rearrange the input values, add syntactic features like commas, parentheses and spacing, and replace names and values used in the design by their implementation equivalents.

Given the software input values of the test plan as an input file, these reformatting tools can produce many different types of files used in the execution of the tests.

ASCII input data files for batch applications can be produced easily with most reformatting tools. If the data files contain binary data, such as COBOL packed decimal numbers or Pascal sets, a more specialized tool or a simple conversion program will be needed.

Setting up a database can be achieved by transforming the input values into a command file for loading the database. Databases today generally have a batch loading facility, and other utilities which can be used to aid the repetitive creation of test databases.

Job streams for executing batch programs can be created by transforming a job template file. If passwords change, the job template can be changed, and the test jobs recreated automatically. The batch transformation tool can generate the needed RUN commands with its own logic, or replace keywords in the template with actual data values.

Procedure calls to test library routines can be compiled from the test data incorporated into a simple program template. With today's advanced debuggers, procedure calls can be tested without putting actual procedure calls into the program template. Debuggers capable of calling procedures can be used to set the values of variables used by the procedure, call the routine, and check the results. This is automated at execution time by using a debugger command file or script, which can by produced like any other ASCII input file with a data transformation tool.

Scripts defining the interaction with the user interface may also be produced automatically. If the interactive application can be run on a terminal, the input values can be reformatted into command files for a terminal emulator. Today's terminal emulators have rich command languages which can be used for driving interactive applications. There are more specialized tools on the market today, other than terminal emulators, designed to drive interactive applications. These tools handle cases not supported by terminal emulators, such as driving PC applications. For these more specialized utilities, the input files may have a format which cannot be generated by file transformation tools, but a user interaction script can still be produced. This semi-automates the manual work of the test engineer when generating the driver tool's script in the required manner. This same script can be used if no adequate tool can be found to automatically drive the interactive application.

LIMITATIONS OF AUTOMATING IMPLEMENTATION

As with design, there are limitations to how much of implementation can be automated. Tools may assist in preparing the data format but some additional logic in the sequence of the data may need to be manually applied. For example, when creating a user interaction script for the input of data, mixing erroneous entries with accurate ones will not work if an accurate entry moves the user to a different menu in the application. The user interaction script must include returning to the entry menu after correct entries.

The impact of this limitation depends on several things: the logic capabilities of the script and macro languages, the expertise of the person writing the scripts and macros, the simplicity or regularity of the interface with the application, and the detail of the test plan. For example, if the input values generated for the test plan state which entries are erroneous (outside the limits) and which are correct, this information can be used by the reformatting tools to direct their logic.

BENEFITS OF AUTOMATING IMPLEMENTATION

Reformatting data automatically rather than manually allows the actual test data to be reproduced from the test plan at will. If the test plan is augmented or revised, the test data can be produced again with little effort. If the particulars of the testing environment or the tools used to automate execution change, or if the user interface, procedure parameter sequences, or database platform is altered during the software implementation, the macros or scripts can be altered and revised test data produced automatically.

In addition, expertise gained in using the file transformation tools can be applied in other areas of software development, speeding up any data transformation tasks. For example, the tools can be used in the testing design phase to extract information from database schema and data dictionary reports.

AUTOMATING THE EXECUTION PHASE

Once the execution script has been generated in the test implementation phase, executing the software becomes a simple matter of handing the script to the driver tool and running it. The other half of test execution, determining if the expectations have been met, is not as simple.

RECORDING THE RESULTS

When the software is driven automatically, the logging feature of the driver tool can be used to record the results. If a database is affected, a query on the database, or the database impact log or transaction log indicates how the software behaved. If any output files were produced, those also provide a record of the results of the software execution.

All of the data recorded during test execution may not be needed in order to determine whether the software met its expectations. Current dates and software version numbers are typical examples of data which is not important. The fact that a version number or date was produced is important, not its value. If the software being tested utilizes a user interface package, that package may produce a large number of escape sequences which are recorded in the log file. Most of these can be ignored; if the user interface package changes the escape sequences, the application will still function the same.

The data transformation utilities which were used in the implementation phase of testing can be used again here to mask out information deemed irrelevant to the software results. Escape sequences may be stripped out entirely, while dates and version numbers can be replaced with placeholders such as $DATE$ or #VERSION#. The masks used here must be written with care; wildcards used incorrectly could result in important data being stripped out along with the unimportant data. The masks should also be organized so that new masks may be added over time. A few unimportant data variations may not show up while the tests are being developed, but will appear during the maintenance of the product.

EVALUATING THE RESULTS

The first time the tests are executed, the results files must be examined closely to see that the expectations outlined in the test plan are met. This task must generally be performed manually, so that any results not matching expectations or insufficient test cases are identified. This is tedious, but fortunately need only be done once.

After this initial examination, subsequent evaluations become a matter of comparing the results from this test to those of the previous test, or an earlier version. File comparison tools can detect any differences, and a data transformation tool may be used to remove irrelevant differences from the file comparison report.

After the tool performs the comparison, the test engineer need only evaluate the differences, if there were any. Differences may result from new tests producing new output, or changes in underlying subsystems, or enhancements to the software, or defect introduction or repair. Judging whether or not the differences are acceptable becomes the primary task performed by the test engineer.

BENEFITS OF AUTOMATING EXECUTION

The most visible benefit of automating execution is the time saved. Driver tools and comparison tools can execute the software and spot discrepancies many times faster than a test engineer. One software project at Hewlett-Packard required a full <u>week</u> of engineering time to run the tests and evaluate the results. Once the test execution was automated, a more comprehensive set of tests for the same software could be executed overnight, requiring only <u>a half a day or less</u> of engineering time to evaluate the differences.

A long term benefit of automated execution is its ability to catch defects in the software and its underlying subsystems and hardware. Defects are found in testing rather than by users, and old problems do not reappear unnoticed. With less data to evaluate, the human judgements are more accurate.

AUTOMATING THE MAINTENANCE PHASE

Maintaining software tests is as complex as maintenance of the software involved. Good practices in the previous stages, like documentation and good organization, contribute to the success of maintaining tests, just as with the software. Keeping track of the results files, and the input files that produced them, is the key to the bulk of the testing which occurs during maintenance.

REGRESSION TESTING

Most testing during the maintenance phase of the software is regression testing, checking to see that new bugs were not introduced by defect repairs, or that changes in underlying subsystems did not surface as defects in the software. Regression testing requires a revision controller to keep track of the results files, the input files that produced them, and the version of the software which was executed. This becomes critical when multiple versions of the software are being used.

For example, consider a piece of software which has versions A and B. The input files used for version B cannot be used for version A because they use new functionality in version B that was not in the older version.

Now consider what happens if the software is using a database with versions 1 and 2. Version A went out with Version 1 of the database. Bugs came in on Version A, the bugs were fixed, tests were created, and a new Version B of the application exists. Meanwhile, the database has rolled to Version 2. Version A must be tested to run on version 2 of the database. If the tests are kept in a version controller, the test engineer can check out Version A of the tests, run Version A with the new database, and verify that Version A still works. Without version control, the test engineer probably either would be unable to do this, or would not bother.

BENEFITS OF AUTOMATING MAINTENANCE

Use of a version control tool in test maintenance reduces the overhead of keeping track of the changes in the software, underlying subsystems, and the corresponding tests' input files and results files. In addition, the version control tool also reduces the amount of storage media needed to keep track of different versions.

The user sees the benefits of the automated maintenance as well. Because old versions of the product can be verified to work with new subsystems, the user does not need to get a new copy of the software every time the database improves its performance. Also, with the increased effectiveness of defect detection due to automated execution of the tests, fewer defects are seen over the life of the software.

SUMMARY

As with software development, software testing has four phases: design, implementation, execution, and maintenance. Automation can be applied to all phases using common tools such as editors, debuggers, version controllers, and terminal emulators, or using more specialized tools to perform similar functions.

The advantages to automating the testing process are numerous, not only in the quality of the software but in time saved throughout the software's life. With fewer defects to address, automatic test execution and identification of discrepancies, the maintenance engineers are able to concentrate on responding to changes in user requirements and advances in technology. Given that maintenance consumes a large portion of most software development resources, the time and effort spent in automating the software testing process may be one of the wiser investments in the software industry today.

THE BIGGEST COMPUTER SECURITY THREAT

by Vladimir Volokh
VESOFT
1135 S. Beverly Dr.
Los Angeles, CA 90035 USA
(213) 282-0420

The problem of computer security was definitely not invented by software vendors -- just read the newspapers every day.

Computer crimes come in different flavors:

* simply reading sensitive data (prices, customer lists, etc.)
* modification of data (payroll rates, shipping information)
* sabotage (viruses, time bombs, intentional system crashes)
* software theft
* unauthorized computer use
* defense-related crimes
* and more...

Security-minded authors have written many books, as well as articles in HP-related publications, on this subject; we at VESOFT became involved in the HP 3000 security industry in its very infancy, presenting computer security papers at HP conferences in Berlin (1981), Copenhagen (1982), Anaheim (1984), and, most recently, at the INTEREX security seminar in 1989.

And yet not every HP3000 computer is secure! The word SECURITY is misleadingly simple, simple enough to make many people think that they have adequate system security without fully thinking out what HP 3000 security really entails.

The issue of computer security is actually very complex -- it involves:

* physical security (guards, dogs, locks)
* system set-up (accounts, groups, users, capabilities, access, etc.)
* LOGON security
* file system security (why does MPE have a :RELEASE command?)
* IMAGE security (have you EVER changed your database password?)
* application security (who is allowed to print checks?)
* data encryption (fields, files)
* LOGOFF security (can people just walk up to an unattended terminal and use it?)

* batch access
* back-up and disaster recovery
* and more... much more...

System security is every bit as much a primary concern of any DP department as the actual applications running on the machine.

SYSTEM SET-UP

Look at your accounting structure first:

* What accounts do you have (check it by using :REPORT X.@)?
* What groups (use :REPORT @.@)?
* What users (:LISTUSER @.@)?
* Which capabilities do each of them have (SM, OP, PM)?
* What kind of access (Read, Write... -- for ANY, AC, GU...)?
* Are all of these entities passworded, or only some of them?
* Are some of the existing passwords too short or too obvious?
* How often are they changed (if at all)?
* How many various levels of UDCs are set on your system?
* And if you rely on them, how easy is it to bypass them?

LOGON SECURITY

Logon to the HP seems to be quite secure with ACCOUNT, GROUP and USER passwords. Or is it? Look carefully:

* MPE error messages at logon time are too *"friendly"*
* Passwords are **readable** combinations of up to 8 ASCII characters
* They are either easy to guess or difficult to remember -- and users write them down (sometimes even stick them to the terminal)
* They are often shared or simply disclosed
* Seldom changed
* If users use the session name (:HELLO MARY,MGR.PAYROLL), it only looks better -- the session name isn't enforceable and the password is assigned to user (MGR) anyway
* Yes, the MPE password is **assigned**, so account manager is the first suspect (and all SM users too)
* Is it easy to enforce shifts (time restrictions on logon)?

- Can payroll be run in the computer room (from LDEV 20)?
- Or can it be done on the weekend?
- Can end-users ever see ":"? What can they do then?
- What is better: to forbid most MPE commands via *"clever"* UDCs or to let users execute only some commands and subsystems?
- And if you have a logon UDC which brings them into an application, how about some other applications (e.g. HPMAIL), some utilities? Should users constantly change their logon ID?

REMOTE ACCESS

Remote access to the computer is common nowdays: dial-up, DS, NS...

- Who knows your dial-up telephone number? Your former employees, current employees, telephone company workers, HP SEs...
- Simple question: what to do if a person leaves the company? (Change all passwords on the system, unplug dial-up forever, request to change your dial-up number...)
- We have a horror story to tell you: one of our customers did change their dial-up number, but... the telephone company set call-forwarding onto it (you know the message -- *"The number has been changed, the new number is..."*)
- And if there are two or more computers linked together, can any programmer access the production HP/960 from the development HP/42?

LOGOFF SECURITY

Logoff is also a problem: you should realize that unattended sessions constitute a major threat to system security. The more sessions you have (it can be hundreds on XL) the less control you have.

- Remember that an unattended terminal is a convenient way for some people to use your system without logging-on.
- Also, if the session is left on after hours and keeps some files open these files might not be backed up.

BATCH SECURITY

Batch security is as important as on-line, but...

- MPE requires passwords to be included in job cards, so a typical job card looks like this :JOB FULLDUMP,MANAGER/SECRET.SYS/QWERTY
- This makes passwords easy to read by unauthorized people and difficult to change on a regular basis by people responsible for system security (it might be you)
- There are some other important things built into streams -- lockwords, database and/or application passwords, etc.

- The situation is somewhat better if all of your streams are in groups with X:ANY,R:GU access -- but try to verify this

IMAGE SECURITY

IMAGE security had better be good -- that's where our most important data usually is. However

- Passwords (up to 63 of them) create the appearance of good protection of the base, sets, and entries

- But... these passwords are often built into sources -- intrinsic DBOPEN requires this; source code is compiled and guess what? IMAGE passwords are **never** changed! The situation is so bad and continues for so long, that HP users seldom recognize this kind of danger.

- It's even worse when using some application packages -- all customers of this package have the same password. Would you buy a car with the same key for everybody else who buys the same car?

- Some system managers sense something wrong in this area and set a lockword on QUERY. It's better than nothing, but what about other database retrieval tools or custom written programs?

FILE SYSTEM SECURITY

File system security in general is very important. A couple of questions come to mind:

- How many files on your HP are released?

- Even worse -- are these files in PM groups?

- And how do you :SECURE hundreds of them? Are you the *"creator"*?

- Which files were accessed on your HP over the weekend?

- How many programs, and which ones, have PM capability?

- Is it possible to :FCOPY the object code of your programs in ;CHAR mode and see all the 'built-ins'?

- Do you like the recent ACD (Access Control Definition) enhancement for MPE/V file system, which, in short, links particular users to the file?

- If so, before using ACDs, think about selection of these files later -- they will be as invisible as :RELEASEd files; think also about setting ACDs for groups of files, saving ACDs after editing text files, etc.

Having said all of the above let's ask ourselves:

What is **the biggest** computer security threat?

And it seems that the problem lies in the wrong approach to the risk management on the part of DP personnel. As long as system managers continue to count on users' ignorance, on end-users being *"good"*, on having **only one** dial-up line (yes, we've heard this one too) and such, company assets -- and some people's resumes -- will be in danger.

TurboIMAGE vs. HPSQL, A (Performance) Comparison

Charles Sullivan
RunningMate Software
3001 I Street
Sacramento, California 95816
(916)444-9304x226

INTRODUCTION

The relational database model has been around since 1970, but relational databases have barely established a toehold in the HP3000 computer world. Most of us have grown comfortable with IMAGE (and now TurboIMAGE) because it is robust, familiar, and easy to use.

But looking into the future, it seems clear that moving to SQL, even on the HP3000, will benefit everybody. Hewlett-Packard has stated that it plans to upgrade HPSQL until a completely distributed relational database, one that allows "transparent" access to any datum anywhere on a network, is achieved. This will be the database equivalent of Nirvana.

Theoretically, relational is wonderful. But in practice, SQL implementations often cannot compete with other database systems when high throughput is crucial. Hewlett-Packard claims that HPSQL is getting better compared to TurboIMAGE, but that TurboIMAGE still has a comfortable performance advantage. On MPE-XL 1.2, HPSQL performs at about 53 percent of the level of TurboIMAGE, and on MPE-XL 2.0, HPSQL runs at about 79 percent the speed.

As usual, a single number does not reveal very much about performance. I will try to uncover a few more details about HPSQL and its performance as it compares to TurboIMAGE.

A VERY BRIEF RELATIONAL HISTORY

The relational database model has been around since 1970 when E.F. Codd published the revolutionary "A Relational Model of Data for Large Shared Data Banks" in June of that year. Although Codd was working for IBM at the time, it was a small, upstart company named Oracle that actually brought the first relational database management system (RDBMS) to market--in 1979. IBM finally released its major RDBMS product, DB2, in 1983. Since that time, the majority of database products sold on mainframes and minicomputers have been relational. Oracle, Ingres, Sybase, Informix, IBM, Tandem, Digital Equipment, and Hewlett-Packard all sell RDBMS products.

A VERY BRIEF TURBOIMAGE HISTORY

TurboIMAGE, which was originally called IMAGE, was conceived in the summer of 1971 and was released to the world in the fall of 1974 for the HP3000 Model CX. IMAGE became IMAGE-B in 1978 because of internal structural changes. Further internal changes begat a name change, and TurboIMAGE was released in 1986. TurboIMAGE/XL for the Hewlett-Packard Precision Architecture machines was available when the first HP3000 Series 930 was shipped in 1988. In the 16 years of its existence, IMAGE has been part of almost every HP3000 computer that was sold. However, TurboIMAGE has been recently "unbundled," so it is not a certainty that the fortunes of TurboIMAGE and the HP3000 will be so tightly coupled in the future.

THE 12 FIDELITY RULES FOR A RELATIONAL DATABASE

Dr. Codd in the mid-1980's developed a famous list of 12 rules which must be met by a database system before it can be considered fully relational. To comprehend the sweeping vision contained in the RDBMS model, one should be familiar with these twelve rules.

1. **THE INFORMATION RULE. Information is logically represented by values stored in tables which consist of rows and columns.**
2. **GUARANTEED ACCESS RULE. Each datum is accessible by resorting to a combination of table name, primary key value, and column name.**
3. **SYSTEMATIC TREATMENT OF NULL VALUES. [This has turned out to be an extremely difficult rule to satisfy.]**
4. **DYNAMIC ON-LINE CATALOG BASED ON THE RELATIONAL MODEL.**
5. **COMPREHENSIVE DATA SUB-LANGUAGE RULE. This language must allow data definition and view definition, data manipulation, integrity constraints, security authorization.**
6. **VIEW UPDATING RULE. Views can be updated, when it is logically possible and consistent to do so.**
7. **HIGH-LEVEL INSERT, UPDATE, DELETE. Data can be manipulated by set operations.**
8. **PHYSICAL DATA INDEPENDENCE. Physical storage is separate from logical definition of data.**
9. **LOGICAL DATA INDEPENDENCE. Programs are unimpaired when "lossless" database changes are made.**
10. **INTEGRITY INDEPENDENCE. Integrity constraints are definable in the relational data sublanguage and storable in the catalog, NOT in programs. ENTITY INTEGRITY: Primary keys cannot have null values. REFERENTIAL INTEGRITY: For each distinct non-null foreign key value in a RDBMS, there must exist a matching primary key value from the same domain.**
11. **DISTRIBUTION INDEPENDENCE. Databases can be distributed around a network without affecting programs.**
12. **NON-SUBVERSION RULE. Using a record-at-a-time language, you cannot subvert the integrity rules and constraints.**

Most, if not all, relational database systems fail to satisfy at least one of these rules. Here is where Hewlett-Packard's HPSQL is deficient: Rule 2) HPSQL allows two identical records to be written to a table; Rule 3) like most systems, HPSQL does not treat nulls in a completely consistent manner; Rule 6) HPSQL does not allow view updating; Rule 9) HPSQL partially fulfills this rule; Rule 10) HPSQL supports entity integrity but not referential integrity.

DATABASE CREATION

It is beyond the scope of this paper to detail the differences between creating an HPSQL database and a TurboIMAGE database. But a few observations are in order. First, using DBSCHEMA to create a TurboIMAGE database is a very well-defined process; there are very few "different" ways of creating the database. Second, creating an HPSQL database can be much more complicated than using DBSCHEMA. But this is to be expected, since an RDBMS tries to be more all-encompassing than does TurboIMAGE.

One very odd thing about HPSQL is that you do not specify the maximum number of records for a table; when a table fills up, you expand it. By the way, it is not at all obvious whether there is an easy way to determine the current number of records in a specific table; serially reading the table seems to be the only way to do so.

DATABASE LOADING

I created an MPE file with one thousand 254-byte records in it. Using ISQL (Interactive SQL) in a batch job, I loaded the HPSQL database by executing the "LOAD FROM EXTERNAL" command 149 times. This took a very long time (over 80 minutes on a Series 925). However, at least this task could be done without writing a program. Using Pascal, I wrote a small program which loaded the equivalent TurboIMAGE database in about 30 minutes, quite a difference. (As a yardstick, writing the same amount of data to an MPE flat file took less than four minutes!)

If you are willing to write a program to load your data into HPSQL databases, you can achieve performance which is very close to that achieved by a TurboIMAGE database. Using an embedded-SQL statement such as the following

```
EXEC SQL INSERT INTO PurchDB.Parts
                     (number,data1,data2,data3)
              VALUES (:number,:data1,:data2,:data3)
```

you can make HPSQL achieve about 80% the throughput of TurboIMAGE, if you seldom issue the

```
EXEC SQL COMMIT WORK
```

statement.

When you make HPSQL "commit" your work, it appears that all database changes get physically written to disc. Thus, the programmer has a great deal of control over the performance of an HPSQL database. In the context of "loading" a database, the programmer can probably be unconcerned about a paucity of COMMIT WORK statements because if the system crashes, you simply restart the program. But data integrity concerns become much more important when you have many users modifying data at the same time (see the next section).

HPSQL INSERT VS. TURBOIMAGE DBPUT

Most HP3000 programmers can code TurboIMAGE procedure calls in their sleep. Here is an example Pascal statement which attempts to add a record to a dataset:

```
DBPUT(base,dataset,mode1,status,list,buffer);
```

The INSERT statement performs the same work for the HPSQL database. After executing the INSERT statement, the program might choose to make the newly written data a permanent part of the database with the COMMIT WORK statement. HPSQL does not require the program to issue a COMMIT after every database modification, but at some point a COMMIT WORK must be done, otherwise all uncommitted modifications are automatically rolled back. Within a multi-user, on-line, transaction-processing environment, a COMMIT would probably be issued after every "transaction," a transaction consisting of one or more database modifications. In a relational database, a COMMIT will either completely succeed or completely fail; thus the database is always in a logically consistent state. This "guarantee" of logical consistency is one major difference between relational and non-relational systems.

Deciding how often to commit database transactions can have a huge effect on HPSQL performance. On a Series 925 running MPE-XL 2.0, TurboIMAGE executed 50,000 DBPUTs in just 896 seconds. As you can see from the table below, HPSQL always took longer. When 50,000 COMMITs were done, HPSQL required 5,133 seconds to INSERT 50,000 records; but when just 50 COMMITs were made, then only 1,094 seconds were necessary.

HPSQL INSERT PERFORMANCE

Commit Frequency (Inserts/Commit)	Elapsed Seconds
1000	1094
500	1207
100	1335
50	1416
10	2011
5	2649
4	3237
3	3156
2	3519
1	5113

SERIAL DATABASE READ

After HPSQL's poor performance in the previous section, I was expecting TurboIMAGE to do just as well in this serial read test. Pitting a native mode program and TurboIMAGE against ISQL and the "SELECT" command was going to be a slaughter, or so I thought.

A sample ISQL command which scans the database without selecting anything could be

```
SELECT * FROM PurchDB.Parts WHERE Number <> Number;
```

The asterisk is equivalent to the "@" list in TurboIMAGE. Using Pascal, the procedure statement for serially reading the dataset is

```
DBGET (base,dataset,mode2,status,list,buffer,dummyarg);
```

On the Series 925, the results showed that HPSQL was much faster: HPSQL required about 160 seconds while TurboIMAGE needed about 290 seconds to read the same amount of data. (Opening the TurboIMAGE dataset in privileged mode and using MR-NOBUF FREADs, the elapsed time was about 70 seconds.)

Apparently, when TurboIMAGE was ported from MPE to MPE-XL, most of the internal algorithms were retained. During a serial read, the overhead of searching and managing some internal buffers appears to be quite expensive. This probably accounts for TurboIMAGE's unexpected slowness.

RANDOM DATABASE READ

Having tested sequential reads, which are extremely important for batch processing, I now wanted to investigate the workhorse of on-line, end-user processing: random access. For this test, each record in the test databases consisted of a 10-byte key plus 960 more bytes of data. The HPSQL database was simple: one table and one index. The HPSQL creation commands were:

```
CREATE PUBLIC TABLE PurchDB.Parts
  (Number      CHAR(10)      NOT NULL,
   Data1       CHAR(240)     NOT NULL,
   Data2       CHAR(240)     NOT NULL,
   Data3       CHAR(240)     NOT NULL,
   Data4       CHAR(240)     NOT NULL)
IN WarehFS;

CREATE INDEX PartNumIndex ON PurchDB.Parts (Number);
```

I tried two different TurboIMAGE databases. The first closely approximated the HPSQL one in physical appearance because I used a detail dataset and an automatic master:

```
SETS:
  NAME: Key, Automatic;
  ENTRY: Number(1);

  NAME: Data, Detail;
  ENTRY: Number(Key),Data1,Data2,Data3,Data4;
```

The second TurboIMAGE database was for maximum speed of retrieval; it consisted of just one hashed dataset:

```
SETS:
  NAME: Data, Manual;
  ENTRY: Number(0),Data1,Data2,Data3,Data4;
```

It took only a few minutes to write a Pascal program for the TurboIMAGE benchmarks. And I am happy to report that writing the HPSQL test did not take too long, either. I simply took one of the short programs supplied by Hewlett-Packard in the SAMPLEDB.SYS group and modified it to fit my needs. However, I did find it necessary to add to the program the following statement:

```
EXEC SQL LOCK TABLE PurchDB.Parts IN SHARE MODE;
```

because the program kept overflowing an internal SQL buffer. (In the last few years HPSQL has developed some elaborate locking strategies in order to improve performance. Relational systems promised to make locking easier for programmers, but HPSQL has become just about as complicated as TurboIMAGE in this respect.)

Extracting a record from the TruboIMAGE database required, in the more complicated case of having both an automatic and a detail dataset, the following Pascal code:

```
DBFIND (base,detailset,mode1,status,item,argument);
DBGET (base,detailset,mode5,status,list,buffer,dummyarg);
```

HPSQL required the following code:

```
EXEC SQL SELECT Number,Data1,Data2,Data3,Data4
          INTO :Number,:Data1,:Data2,:Data3,:Data4
          FROM PurchDB.Parts
         WHERE Number = :Number;
```

Note that you cannot explicitly reference an SQL index when you select data; the above statement will find the same data whether or not the index exists. To a programmer, this is the beauty of relational: you do not have to write code that navigates via pointers through a database.

Despite its "defeat" in the serial read test, TurboIMAGE won this battle. Looking at elapsed times, HPSQL performed nearly as well as TurboIMAGE, but HPSQL needed three times as much CPU as did TurboIMAGE when processing the same data. Why did HPSQL perform so "poorly" in this random test after doing so well in the serial test? Well, the serial test required only one SELECT statement to scan about 150,000 records. But the random test needed 4,000 SELECT statements to find 4,000 records. Obviously, there is a relatively large amount of interpretation needed each time an SQL SELECT statement is processed.

HPSQL UPDATE VS. TURBOIMAGE DBUPDATE

When updating existing data, TurboIMAGE requires that you first get the existing record with a DBGET before you issue the DBUPDATE:

```
DBGET (base,detaset,mode,status,list,buffer,argument);
DBUPDATE (base,detaset,mode1,status,list,buffer);
```

On the other hand, HPSQL does not require such a prefetch, so the update operation can be very efficient, although in an on-line environment, an SQL SELECT will normally precede an HPSQL UPDATE. The HPSQL statement for an UPDATE looks like this:

```
EXEC SQL UPDATE PurchDB.Parts
             SET data1 = :new_data1
           WHERE number = :number;
```

Of course, whenever modifications are made to an HPSQL database, a COMMIT WORK must eventually be done to save the changes. Once again, the more often a COMMIT is used, the slower the performance of the HPSQL database, as you can from the table below.

HPSQL UPDATE PERFORMANCE

Commit Frequency (Updates/Commit)	Elapsed Seconds
1000	291
500	292
100	288
50	290
10	319
5	359
4	396
3	439
2	517
1	788

TurboIMAGE required 431 seconds for the equivalent test. Once again, HPSQL is not as fast--in most applications--as TurboIMAGE, but the UPDATE vs. DBUPDATE test was much closer than the INSERT vs. DBPUT result.

THINGS THAT BOTHER ME ABOUT HPSQL

After writing 50,000 records to the HPSQL database, I wanted to benchmark the delete function of ISQL this way:

```
DELETE FROM PurchDB.Parts WHERE Number = Number;
```

Unfortunately, the log file filled up and aborted my test after about 5 minutes. So I built the log file much, much larger. This time ISQL blew up because of a shared memory allocation failure within some internal table. So then I tried deleting a subset of the table; I received the same error message. So I just gave up. Needless to say, performing the same function against a TurboIMAGE database using either Query or a custom program would have been very straightforward.

We are using HPSQL in one of our (small) production systems and discovered that whenever you use the ORDER BY clause of the SELECT statement, that there must exist within the HPSQL database enough temporary storage to perform the sort function. So if you have multiple HPSQL databases, you need to allocate a lot of extra space just to allow for the ORDER BY clause. If your SELECT command joins two tables with the ORDER BY clause, disc space requirements can become quite large. Is this a plot to sell disc drives?

Most of the other complaints--so far--revolve around system administration requirements. This is probably because everything is so new about HPSQL, but I also suspect that some of our complaints are because some parts of HPSQL are just not "friendly."

CONCLUSION

The great relational debate has always been about performance. For many years, people said that a relational database would never be competitive with more established database systems because the performance of an RDBMS was so slow--the first implementations of relational systems were very pedestrian. And, judging from my tests, I would conclude that choosing TurboIMAGE over HPSQL can indeed be justified for performance reasons, especially if you have some of the performance utilities that make TurboIMAGE even faster.

But choosing to jump on the RDBMS bandwagon would not be a mistake, because there are so many advantages to doing so: reduced data redundancy, improved data integrity, easier database maintenance, reduced programming effort, and less complex logical structure (the end-user sees nothing but columns and rows). TurboIMAGE is a great system and will be around for many years, but the logical consistency of Dr. Codd's model is such that relational will continue rapidly winning converts, even on the HP3000.

Procedure Exits on MPE XL

Rajesh Desai
Jamie Odell
Commericial Systems Devision
Hewlett-Packard
19447 Pruneridge Ave.
Cupertino, CA 95014

May 15, 1990

1 Introduction

The computer industry is rapidly evolving towards open systems and architecture. Major suppliers of proprietary systems are now attempting to increase the arsenal of software solutions available on their systems. They are increasingly looking to third parties and independent contractors for assistance. The open standards of UNIX systems facilitate this process of distributed development of software very well, but it raises a host of issues for proprietary systems like MPE XL.

On MPE XL, the internal structure and working of the operating system is information which is not widely disseminated. System tables, their relation to each other, internal functionality and its uses are clouded in a shroud of mystery. This makes it difficult for non-HP developers to integrate their software with MPE XL. The consequent loss of tight coupling with the underlying system leads to performance degradation, unsupportable, fragile products and wide use of the proscribed practice of using privileged mode. The Architected Interface product for Operating System (AIF/OS) released on MPE XL Version 2.1 addresses the problem of intercepting internal data and exposing internal functionality. It also was a first in being an interface to be called only from privileged mode and assuring developers a limited support for their use of privileged mode.

MPE XL is a powerful system, with a rich set of functionality. These extensive capabilities are targeted towards the general user and may require customization for specific value added systems and applications. A lot of products existing on MPE V depend upon some sort of low level customization. This requires an interception of internal control flow as opposed to the interception of internal data. IBM users have long been familiar with this kind of functionality, which goes under the banner of User Exits. The Architected Interfaces Facility for Procedure Exits (AIF/PE) is being developed precisely towards alleviating this problem.

AIF/PE provides a way for users to specify a procedure(s) to be executed whenever some other user/system procedure is called. This specification by the user can change dynamically during the lifetime of a system, requiring no reboot or relink, and will affect all program files, currently loaded as well as those loaded subsequent to the specification. It will intercept all external calls made to the specified procedure and in certain cases, all internal calls as well. It is a powerful tool to effectively replace internal procedures by user written code and as such should be employed with great circumspection. It is slated to be released on version 4.0 of MPE XL.

The next section places this product in its proper perspective, enumerating the kind of applications it can be used for. The third section describes the features of AIF/PE. The fourth section provides a brief overview of its external specifications. The fifth section describes a couple of examples in detail.

2 Perspective

As mentioned in the introduction, a multipurpose system like MPE XL requires a host of applications to harness its power. Some of these applications require the interception of internal procedures which AIF/PE provides.

2.1 IBM User Exits

IBM systems have for a long time provided the means to their SEs to customize certain parts of their operating systems for the end user's requirements. This customization mainly deals with the introduction of user code into the base operating system. This code effectively replaces existing procedures which were mainly stub procedures. Its chief utility lies in the customization of an entire system towards a particular requirement. AIF/PE on the other hand, allows this customization on a product specific basis, so that different applications can view the system from their own standpoint.

2.2 Relinking

Major releases of MPE XL do not occur everyday. A bug reported against a particular version has to wait for the next release for general distribution. An alternative to this has been the creation of patch tapes. Patch tapes require in general a relinking of the base system. This is an expensive process and requires a reboot. AIF/PE could provide a short term work around to this laborious task, by stubbing out the old, defective procedure with a new procedure. These pseudo-patches can be developed by HP Support personnel at the user site, thus improving the turnaround time on critical fixes at a slight cost to performance. Moreover, these pseudo-patches can be installed and removed dynamically with minimal impact on system performance.

2.3 Online Installation

The provision of xl.pub.sys on MPE XL machines has been a tremendous help in enabling non-OS products to have a systemwide impact. It allows them to develop code which would be in the binding environment of every program on the system. The user just has to introduce code into xl.pub.sys. However, this requires that no one is currently loaded through xl.pub.sys - in essence, the system must be quiesced. In certain instances, this requirement could be dropped by using AIF/PE instead.

2.4 Error Recovery Testing

Any software developed on top of other systems has some code for recovery from errors returned by the other subsystems. This is error recovery code. Similarly, a robust software system will have a lot of internal error checking and recovery. These errors are all unexpected errors, which may take care of timing windows or corrupt system states. It is pretty rare that these areas of the code can be executed during normal regression testing, because it is highly unreachable. AIF/PE could be used to artificially generate the appropriate error messages and reach the recovery code. This should add more robustness to the product.

2.5 Enhanced Functionality

There are a number of products on MPE XL which enhance currently existing features of MPE XL - in the areas of security, backup, data management, job scheduling, to name a few. For instance, security packages would enforce more security than that supplied by MPE XL. This would generally require a close monitoring and interception of all security related access on a machine. This can be provided using AIF/PE.

2.6 Tracing

Complex systems like MPE XL have a large number of parameters, like disc space, main memory, size of various tables, etc. These parameters can be fine tuned to obtain better behavior for specific applications. But, the process of tuning requires a large amount of data on the behavior of systems. This data frequently is composed of traces of procedures called, amount of time spent in various procedures and the common parameter sequences passed. The collection of this data is referred to as Tracing. AIF/PE should make it easy to collect trace information on external calls and in some cases, internal calls too.

2.7 Logging

MPE XL provides various kinds of logging facilities. These facilities are targeted towards the generic user base and may not always be sufficient. Enhancing the kind and detail of information being logged is a recurring request faced by HP. The AIF/PE facility should make it easier for HP to help developers to write tools to improve the content of user defined logs.

2.8 Proprietary Information Disclosure

The calling sequences of internal procedures are proprietary in nature. Products using privileged mode sometimes call internal procedures and they obtain the information from standard channels in HP, established towards this end. However, with AIF/PE, some of the critical internal procedures will be documented. These procedures may change from version to version, but the purchasers of AIF/PE will be informed of changes. This is another first for MPE XL, in that, internal procedures are being made available in a supported fashion.

3 Product Features

This section proceeds to discuss AIF/PE in more detail. It describes the principal features of this product.

3.1 Intercepting Procedure Calls

MPE XL provides two principal mechanisms by which procedures are invoked. One of them is an internal call. This mechanism is used when the caller and the called are both located in the same module at link time. The other mechanism is an external call. This is employed to call procedures which were unresolved at the time of linking. When the program file (or the library) is loaded, these external references are resolved (for eg.. references to procedures in nl.pub.sys).

AIF/PE provides the guarantee that certain designated procedures will ALWAYS be intercepted, whether the call is internal or external. Once, AIF/PE intercepts the call, it will pass control to appropriate user specified procedures. The procedures can be designated through two mechanisms. HP will itself decide and designate certain procedures in nl.pub.sys, xl.pub.sys and other HP software, as per the requirements of the third parties. Third parties can themselves also designate their own user procedures to be interceptable. In this case they will be treated exactly like HP designated procedures.

AIF/PE provides the guarantee that for ANY procedure residing in a library loaded on the system (user library as well as system library), ALL external calls to it will be intercepted. Thus, any references to a particular procedure from outside the library, can always be trapped.

The intercepted procedure will henceforth be referred to as the target procedure.

3.2 Enabling Handlers

The user must, in order for their own code to be invoked whenever the target is called, specify its own procedure as an handler for a particular target. This specification is called a binding of the handler. A handler will be allowed to be of two types - invocation and termination. Invocation handlers are called before the target execution and termination handlers are called after the target finishes executing.

AIF/PE intercepts all procedures specified to be targets. It then scans down the list of invocation handlers and calls them, then it calls the target and finally the list of termination handlers.

3.3 Stubbing of procedures

The user's handler procedure will be able to decide at run time whether to invoke the target or not. For example, a security product may intercept a

:Hello command and decide that it should not allow the logon, in which case it would stub the call out. Unconditional stubbing out corresponds to the case of a procedure replacement. In case of some unexpected, inconsistent state, the handler can even escape out of its code. The escape would then be propagated to other handlers and even to the calling procedure if so required. In case the target itself escapes then this would be propagated to the termination handler.

3.4 Multiple handlers

As mentioned earlier AIF/PE allows a product-specific customization, as opposed to a systemwide customization. Each product can intercept its set of procedures and bind their own handlers. A single product could have multiple handlers attached to it, at invocation and/or termination. This allows third parties much more flexibility. They do not have to mandate that they cannot coexist with some other product, always a sensitive thing to mention at a sales pitch.

3.5 Prioritization of handlers

As mentioned in the above section, multiple handlers could be attached to the same target. The order of invocation of handlers is very crucial. The order would determine how multiple products installed simultaneously influence each other's behavior. The user can define the priority at which their handler should be invoked. They can specify whether they want their handler to be the first one or the last one or some pre-ordained priority.

3.6 Access to parameters

The invocation handler code will have access to all the parameters passed in by the user to the target. The handler can examine the parameter sequence and modify it as desired. This requires the handler to know the calling sequence of the target, which would have to be obtained through proprietary channels, unless specified in the AIF/PE documentation.

The termination handlers will be called upon the termination of the target procedure. For targets returning functional results, the handlers will be able to examine and modify the functional results. For targets escaping out, the handlers will be able to propagate the escape and change the escape code, or stub the escape out. In any case, they can initiate their own escape if so required. Note that, the parameter sequence will not be available to these handlers.

3.7 Target Encapsulation

In certain kinds of application, the user would desire to logically encapsulate the target procedure within its own handler. That is, the handler is called before the target, examines the parameters and modifies them, calls the target and then examines the results of the target before returning to the user. This effect of encapsulating a target is obtained by binding an invocation and a termination handler at the same priority on the target. AIF/PE then guarantees that either both the handlers or neither of them will be invoked. As for passing information across the two handlers, this can be managed using the process-local SOM-local DP space.

3.8 Dynamic binding/unbinding

The binding and unbinding of handlers will be completely dynamic. Unlike, other similar products on the market, AIF/PE will not require the user to relink, reboot or even quiesce the system, while handlers are being enabled and disabled. There will be a brief period of slowdown on the system, when the new handler is being bound/unbound.

3.9 Security Issues

User developed handlers will be called whenever a particular target is called. This transfer of code to the handler is done, irrespective of the process which is calling the target and the current privilege level of the process. In fact, the user code could run on the interrupt control stack, ports facility processes or progen's stack. This is a major first in the short life of MPE XL, in that user code will now be allowed to run at privilege levels 0 and 1. Until now, only the code residing in xl.pub.sys and nl.pub.sys could run at 0 and 1.

3.10 Supportability

There will be a complete range of support tools available, to indicate what handlers are currently bound to which targets, which targets are always intercepted, who has bound what handlers, etc. This should make it easier for support personnel to trouble-shoot AIF/PE based products being used at end user sites.

4 External Specifications

In order for the user to specify which procedure exits are going to be used, procedures similar to the following would be needed.

4.1 Enabling Libraries

The first thing a user must do to enable procedure exits, is tell the system in which libraries these procedures live, and how to treat the procedure exits in these libraries. The user would do this by calling a procedure similar to *PROCEDURE PEBindLib*. This procedure would need the following information:

- library name - This file that contains the procedure exits.
- library type - This would indicate specific information about how this library should be treated. For instance, does this library contain instrumented procedures?
- priority - This parameter would allow the user to specify what priority should be given to the handlers within this library. The user could indicate that priority was not important, or that these procedure exits must be executed prior to any other exits enabled on the same procedures. Obviously, only one library can obtain a specific priority. Programmers should try to make their applications not depend on getting a specific priority if at all possible.
- unsat procedure - This would be the name of a procedure that all unresolved externals for this library should be bound to. This is similar to the unsat= parameter on the CI run command. An example would be to specify *debug* as the unsatisfied procedure, so any unresolved externals would invoke the system debugger.

Additionally, a procedure to unbind the library would exist.

4.2 Enabling Procedure Exits

With the library enabled, the user can now specify which routines in the system are to be targeted for procedure exits. Also, the user must specify which procedure exits to use from the enabled library. The user would call a procedure similar to *PROCEDURE PEBindHandler*. This procedure would need at least the following information:

- handler procedure name - This would be the name of the procedure exit, including its library name.
- target procedure name - This would be the name of the routine being handled by the above procedure.
- handler type - This would indicate if this handler should be invoked upon entry to the target procedure or upon exiting the target procedure.

Additionally, a procedure to unbind the exit library would also exist.

4.3 Exit Management

Because several applications would be able to apply their own procedure exits to the same targets, applications will need to get information about the condition of the targets. This is done by examining the data structures used by the system to keep track of procedure exits. The following is a list of some of the information that exit management could provide to procedure exit users:

- priority - the assigned priority of a procedure exit
- procedure address - the address of the entry point of a procedure exit
- number of targets - the number of targets a particular procedure exit is handling
- list of targets - a list of all the target names this procedure exit is handling
- number of invocation handlers - the number of invocation handlers enabled on a particular target procedure
- list of invocation handlers - a list of all the invocation handlers enabled on a particular target procedure
- number of termination handlers - same as number of invocation handlers, but for termination handlers
- list of termination handlers - same as list of invocation handlers, but for termination handlers
- library name - name of library in which a procedure lives

5 Examples

The following are a couple of example uses for procedure exits.

5.1 Wild Carded File Equations

By enabling an invocation procedure exit on file open, an application could catch every file open on the system. The procedure exit could then examine the file name, compare it to a list of special file equations, and then pass file open a modified file name. The following diagram illustrates the execution:

```
+-------------+                                 +-------------+
| Open a file |      Normal Execution           |             |
| called 'foo'|-------------------------------->| FOPEN 'foo' |
|             |                                 |             |
+-------------+                                 +-------------+


+-------------+                                  +-------------+
| Open a file |      Execution with a            |             |
| called 'foo'|-----+ Procedure Exit     +------>| FOPEN 'moo' |
|             |     |                    |       |             |
+-------------+     |   +-------------+  |       +-------------+
                    |   | Translate   |  |
                    +-->|any file f@  |--+
                        |    to m@    |
                        +-------------+
```

In this example, the application tries to open a file called 'foo'. Under normal circumstances, the application calls file open with the file name, and file open opens the file. The application user can specify a normal file equation, such as:

```
FILE FOO = MOO
```

This will make the file system open a file called 'moo', instead of 'foo'. But what if the user does not specifically know the file will be called. Furthermore, suppose the user wants all files that are in group 'odell' to be replaced with files from the group 'desai.' The user could enter file equations for each file, if the names are known, and if the number of files does not exceed the file equation table. But it would be more convenient, if the user could just tell the application to replace all files from the group 'odell' with files from the group 'desai.' The user wants to write a file equation like this:

```
FILE @.ODELL=@.DESAI
```

This feature would be nice, but it does not exist. Wild carded file equations would be relatively simple to implement, however, if the application writer could use procedure exits. Since all calls to file open are intercepted by a procedure exit, the procedure exit just needs to look for the file equation (the details are left for the designer of such a utility), and modify the file name appropriately. This could all be done with the user being totally unaware of what is happening.

Imagine trying to compile a program that includes several other files. Now lets say that the included files need to be modified. The programmer could use wild carded file equations to point the compiler to the new include files in a local work group, leaving the original files untouched. This could be very useful if several people are working on the same group of files or if the user must use different versions of the same files. Each version of the files could be kept in separate groups. The programmer could user wild carded file equations to point the compiler to the correct group.

5.2 Intelligent Make Tool

Another example of a useful procedure exit enabled on file open is an intelligent make tool. This procedure exit would be enabled on file open every time a user wanted to link a program. When the linker tried to open an object file, the procedure exit would be invoked. The following diagram illustrates how the utility might work:

PROGRAM FOO is made up of the following object modules:

- yfoo1 (4/4/90)
- yfoo2 (4/4/90)
- yfoo3 (4/4/90)

The source for each object module comes from the files:

- ofoo1 (4/3/90)
- ofoo2 (4/5/90)
- ofoo3 (4/4/90)

The dates next to the file names indicates when each file was last modified. The source file ofoo2 is newer than the object file yfoo2 and, the user wants to build the most up-to-date version of the program.

The procedure exit first checks the creation date of the object file, and compares it to the source file used to create the object. If the object is older than the source, the procedure exit can force the object file to be recompiled prior to linking it into the program. In the above example, the procedure exit would decide to recompile the module ofoo2 before trying to link the program.

Now combine this example with the previous example. Suppose the user wants to be able to use this smart make utility along with wild carded file equations. This is not an unreasonable request, and is possible. Remember that the user wants wild carded file names to be used on every file open. In this case, wild carded file equations can be installed as a handler before the make utility and again after the make utility.

The following diagram illustrates the importance of placing the wild carded file equation handler in both places:

```
Wild Carded File Equations:
---------------------------

o@.odell = o@.desai
y@.odell = y@.desai

Note: The convention is files beginning with O are source code,
      and files beginning with Y are object files.


Case 1: Wild Carded File Equations Handler is Bound Before Make Utility to
File Opens

+----------------+ Try to open     +---------------+     +-----------------+
| Link YFOO.ODELL|   YFOO          | YFOO.ODELL    |     |YFOO.DESAI is    |
| to get         +---------------->|is replaced    +---->|older than OFOO. |
| FOO.ODELL      |                 |with YFOO.DESAI|     |ODELL            |
+----------------+                 +---------------+     +--------+--------+
                                                                  |
                                                                  V
                                   +---------------+     +-----------------+
                   Result is       | Build FOO.    |     |Recompile OFOO.  |
                  Not What We <----+ODELL with     |<----+  OFOO.ODELL to  |
                   Expected        |YFOO.ODELL     |     |get YFOO.ODELL   |
                                   +---------------+     +-----------------+
```

Case 2: Wild Carded File Equations Handler is Bound Before and After Make Utility to File Opens

```
+----------------+ Try to open      +---------------+      +----------------+
| Link YFOO.ODELL|   YFOO           | YFOO.ODELL    |      |YFOO.DESAI is   |
| to get         +----------------->|is replaced    +----->|older than OFOO.|
| FOO.ODELL      |                  |with YFOO.DESAI|      |DESAI           |
+----------------+                  +---------------+      +-------+--------+
                                                                   |
                                                                   V
                                    +---------------+      +----------------+
                    Result is       | Build FOO.    |      |Recompile OFOO. |
                    What We    <----+ODELL with     |<-----+ OFOO.DESAIto   |
                    Expected        |YFOO.DESAI     |      |get YFOO.DESAI  |
                                    +---------------+      +----------------+
```

By installing the wild carded file equations exits in both places, the user will be assured that any file opened as a result of the make utility will also get the benefit of wild carded file names. Normally, a procedure exit library will be bound in a specific order. Any procedure in that library cannot be exited by procedures in libraries bound before it. By binding wild carded file equations in two places, before and after the make utility, the user is assured that all file opens be exited.

6 Summary

In summary, the AIF/PE product should substantially address a host of issues for developing code on MPE XL systems. It should go a significant way towards improving the development environment of MPE XL. It will provide a powerful capability for dynamic interception of control flow, providing a foil to the AIF/OS and AIF/MI products which allow the interception of data. It is targeted for the version 4.0 of MPE/XL.

TITLE: Using MPE/XL Debugging Tools

AUTHOR: Bruce Toback

OPT, Inc.

10681 Foothill Blvd., Suite 201

Rancho Cucamonga, CA 91730

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 3132

When to Use Imsam / When to Use Omnidex (When to Use Image)

by

Suzanne Harmon
AH Computer Services, Inc.
8210 Terrace Drive
El Cerrito, CA 94530
415 525-5070

One of the biggest hurdles I find with people's ability to determine when to use Image search items versus Imsam keys versus Omnidex keyworded fields, is understanding, and keeping constantly in mind, the following:

Image Search Items:	Access by the value, the whole value and nothing but the value. All pieces of a key must be physically adjacent.
Imsam Keys:	Exactly like KSAM. Access via one or more characters of the key value, starting in the first position of the key and moving left to right, exactly positional. The key value may be made up of physically adjacent data or it may be made up of a byte or more here and a byte or more there (composite keys). Imsam, however, treats the bytes as though they were physically adjacent in the sequence you have defined them.
Omnidex Keywords:	Parses the keyworded field into "words" (ie. John J. Smith parses into three words - John, J and Smith). You can then search for one or more words in one or more keyworded fields using Boolean combinations. Non positional. Omnidex also allows you to create composite keyworded fields. Like Imsam, they are defined by you using a byte or more here and a byte or more there.

In many cases it is non-trivial deciding how to "key" the data sets in a system. However, there are a few standard cases where it can be pretty clear cut.

1. Always make your most frequently accessed unique key on an Image Master and Image search item. Because: An Image keyed retrieval on a master data set is faster than any other "keyed" retrieval.

2. For data sets which are logical masters but do not require Image or Omnidex relationships with other data sets, consider making them stand alone details with an Imsam key (plus other Imsam and/or Omnidex keys as needed). Because: Putting or updating can frequently be faster, and you can save large amounts of disc space because you do not need to leave the free space for performance on a detail that you do on a master.

3. If you have more than one unique key in a master data set (ie. employee number and social security number), make the less frequently accessed on an Imsam key. Because: Unique keys don't usually make sense in Omnidex and are less performance effective in Omnidex.

4. Imsam keys allow you to retrieve records in sorted Imsam key sequence (like KSAM). Therefore they are a good replacement for sorted keys. For example, Customer # sorted on Order Date in an Order Header data set could be replaced with an Imsam composite key made up of Customer # and Order Date. This effectively gives you four keys - Customer #, Customer # + Order Year, Customer # + Order Year and Month and Customer # + Order Date. It also allows you to retrieve records on any one of the four options in sorted order by Customer # and Order Date. It is much more performance effective than a sorted Image path if you add records to the data base in other than sorted key sequence (ie Order Date within Customer #).

5. Imsam keys are your best bet for partial key retrievals, provided you do not need to do the partial retrieval in conjunction with other values or fields (see 4 above for a good example of this). Using partial key retrievals, and stopping at the appropriate point, is also very useful for range retrievals. Partial retrievals and range retrievals are not as efficient in Omnidex. However, this is one of the largest areas of confusion between Omnidex and Imsam.

6. Because we have always used KSAM to do generic name retrievals, etc., we automatically assume that using Imsam with partial key retrieval is the obvious transition. However, frequently using Omnidex as a replacement for these kinds of retrievals is far more desireable and effective. In the case of name fields, or any textual fields, Omnidex is a much better choice.

7. The most important consideration when thinking about Omnidex is: Do I want to inquire on values in this field in combination or in conjunction with values in other fields? If the answer is Yes, then I probably should make this an Omnidex keyworded field.

8. The next most important consideration when thinking about Omnidex is: Do I need to be able to inquire on parsed words within the field with positional independence? If so, then this should be an Omnidex keyworded field.

9. A field can be an Image search item and/or an Imsam key (or part of a composite Imsam key) and/or an Omnidex keyworded field (or part of a composite Omnidex keyworded field). There are cases where this makes sense.

10. One of the most important things you need to keep straight (and it ain't easy), is how do I access records based on the keying decisions I have made?

-Retrieval on Image search items is done via the standard Image intrinsics.

-Retrieval on Imsam keys can be regular or discrete. Regular retrieval involves a preliminary call which sets up the pointer to the first qualifying record and retrieves it for you; this is followed by subsequent calls which allow you to retrieve one record at a time moving backwards or forwards. Discrete mode does not return a record to you but returns the full Imsam key entry. For example, if I do a partial key retrieval of "90" on a date field it will return to me values of 900101, 900102, etc. Since one I/O retrieves many Imsam keys, this can be used very effectively to rapidly get account balances, etc. For example, if you create a composite key of department, account, account balance, and then do a partial retrieval by department, you can get all the account balances for hundreds of accounts with just a few I/O's.

-Retrieval on an Omnidex keyworded field builds a list of records which qualified. The list consists of Image search items if there is a master data set in the domain, relative record numbers if it is a detail only domain. These search items or relative record numbers can then be used as the basis for Image and/or Imsam retrievals against the data base.

11. Keeping in mind the retrieval considerations, think about the advantages gained by effectively combining Image, Imsam and Omnidex keying. For example, if my customer master has an Image search item of customer number, I can Omnidex such fields as Name, Address, City, State, Zip, Region, VIP code, etc. On my Order Header set (which is a detail data set), I may want to make an Imsam composite key, as we discussed above, of Customer # and Order Date. Then if I want to find all the orders this month for VIP customers in California, Oregon and Washington, I can first do an Omnidex retrieval on State = CA+OR+WA and VIP code = Y; then I can use the Customer ids returned in the list from this Omnidex retrieval concatenated with 9005 (if this is May of 1990) to do an Imsam retrieval on the Order Headers.

12. With the new multi-find capabilities in Omnidex, you can use the list of Customer ids returned in 10) above, for example, as the keyword values used to search an entirely different domain, a sales history domain for example, even if it is in another data base (Customer # would have to be an Omnidex keyworded field in the other domain).

The important thing to remember is that before you start thinking about what to key and how, you must think through carefully how data will be retrieved and with what frequency. It is good to make a diagram which deals with data logically, instead of physically, as a starting point. For example: Customer data must be retrieved by Customer # constantly; Customer data must be retrieved by Customer Name constantly; Order Headers must be retrieved by year and month weekly; and so on.

Next, examine the amount of data, volatility of data, and machine resources available, to determine if the overhead of keying is justified. For retrievals that are done nightly or weekly, is deferred indexing a possibility (ie. indexing certain fields using a nightly batch run)?

Finally, don't be afraid to try things even though they may not work out for the best. It is very easy to change Imsam and Omnidex keying. If you have a data base transformation utility it is also easy to change Image search items. You can rarely be an immediate success at anything, and making the correct choices about keying is no exception. Allow yourself to try things even though they may not be right the first time.

Your Own Private Program Library

Author: Gerrit Altena
A & B Automatisering
Fazantenkamp 226
3607 CL Maarssen
The Netherlands
+31-346571664/-346566509

Introduction

In nearly every programming language, there is the possibility at compile-time to supplement the file with source-lines from another file or library.

In Cobol this is accomplished by the COPY-statement, in SPL by the $INCLUDE, other languages have there own method.

What does the include or copy means for a source-program?

It is a method to add source-lines to your program, source-lines which have been used and debugged before; therefor the time to write and test the program is reduced.

However every programmer has his/hers little routines, which are invariably included in the program she/he writes.

These routines are usually copied from one program to the other, not because these routines are not good, but because for one or other reason they are not or can not be used department-wide.

If these little routines can be stored in a library for your own private use, then the copying is not necessary anymore and these nice little routines can easily used over and over again. Even when you change them, changing the old programs is an easy task.

The concept

Before we start with a solution to the problem, let's look at the elements of a compilation.

A compiler is a program, which converts a file with source-lines into a file with machine-readable instructions (of course still to be prepped, but that goes without saying).

The file with source-lines is a simple ASCII-file with text, created by any text-editor; the text is formatted as required by the compiler.

If eq the Cobol-compiler finds a COPY-statement, it opens the copylib-file, reads the module with the source-lines from it, merges them with the source-file and at the end of the module continues reading from the source-file.

To repeat, from a second ASCII-file source-lines are merged with the first file, based on instructions in the first file, prior to the conversion to machine-readable instructions.

The question now is, can this merging of two source-lines, based on instructions be done before the compiler is used?

Yes, of course this can be done.

The solution

From the above the requirements for the merging of tho files can be established:

1. setup a file with the routines,

2. define an instruction, which will start the copy of source-lines,

3. write a little program, which will execute the instruction.

The following is an example to a practical solution of the above requirements:

1. set up a KSAM-file with a record-length of 96 bytes and a key of 6 bytes, starting at position 91; duplicate keys are allowed,

```
:RUN KSAMUTIL.PUB.SYS
  BUILD lib;
  KEYFILE=libk;
  KEY=b,91,6,,DUP;
  REC=-96,85,F,ASCII;
  DISC=2000,1,1
```

2. start up the editor (or your favourite text-processor) and type the source-lines in; put for every group of

related source-lines the same key for access in columns 91 thru 96,

3. fcopy the file from the previous step to the KSAM-file,

```
    :FCOPY FROM=slib;TO=lib
```

   the fcopy will automatically erase the contents of the KSAM-file,

4. write a little program with a sequential input-file, a sequential output-file and a KSAM input-file.

   for Cobol:

```
  SELECT input   ASSIGN "input".
  SELECT output  ASSIGN "output".
  SELECT ksam    ASSIGN "ksam"
    ORGANIZATION IS INDEXED
    ACCESS       IS DYNAMIC
    RECORD KEY   IS ks-module-name WITH DUPLICATES
    FILE STATUS  IS stat-item.

FD input.
01 input-rec.
   05 instruction         pic x(06).
   05 input-module        pic x(06).
   05 filler              pic x(68).

FD output.
01 output-rec.
   05 filler              pic x(80).

FD ksam.
01 ksam-rec.
   05 ksam-line           pic x(80).
   05 filler              pic x(10).
   05 ks-module-name      pic x(06).
```

5. The specifications for the program are:

   a. read a record from the sequential input-file

```
      read-input.
         READ input AT END go to end-of-input.
```

   b. if no $COPY is present, starting in column 1, write the record to the sequential output-file and return to step a.

```
         IF instruction NOT = "$COPY " then
            WRITE output-rec FROM input-rec
            GO TO read-input
```

```
          END-IF.
```

c. use the module-name after the $COPY line to point to the first record of that module in the KSAM-file.

```
          MOVE input-module TO ks-module-name.
          START ksam KEY IS EQUAL TO ks-module-name.
```

d. read the KSAM-file on ascending key.

```
       read-ksam.
          READ ksam.
```

e. if the module-name in the KSAM-file is the same as the one from step c. write the line to the sequential output-file, continue with step d.

```
          IF input-module = ks-module-name then
             WRITE output-rec FROM ksam-rec
             GO TO read-ksam
          END-IF.
```

f. go to step a.

```
          GO TO read-input.
```

6. the sequential output-file is an ASCII file, which can be used by the compiler for conversion to machine-readable intructions.

In the above intructions no mention is made of the language, although the sample used is in Cobol. This is indeed not necessary, because the same principle may be used with any language.

Conclusion

Shown are the principles and the most important statements for you to create and use your own private program library. This paper may lead to more use of program libraries and thus less time for coding and testing the program.

MAKING OTHER PEOPLE'S PROGRAMS DO WHAT THEY WERE NEVER INTENDED TO DO

by Eugene Volokh
VESOFT, Inc.
1135 S. Beverly Dr.
Los Angeles, CA 90035
(213) 282-0420

THE PROBLEM

We first got the idea for the MPEX hook in 1980, when one of our earliest users complained to us about the time it took him to get into MPEX. Whenever he was in EDITOR and wanted to use MPEX, he'd have to /KEEP the file, exit EDITOR, get into MPEX, do the command, exit MPEX, re-enter EDITOR, and re-/TEXT the file -- a lot of work, especially on his overloaded Series III (remember those?).

We were thus faced with a substantial problem. What our customer really wanted was a change to EDITOR -- some way in which he could execute MPEX commands directly from within EDITOR, without exiting or re-entering, /TEXTing or /KEEPing. He wanted us to modify somebody else's program.

Unfortunately, we did not have the EDITOR source code; however, even if we had it and modified it to suit our needs, we'd have to repeat this modification every time a new version of EDITOR came out (and re-send this new version to all of our customers). Furthermore, the same sort of feature would be needed in TDP, QUERY, etc. -- even if we had the source code to these programs, we wouldn't want to modify them all (and then re-modify them for each subsequent version).

Now, if we couldn't modify the source code, could we modify the object code? Perhaps find out which locations needed to be patched to implement this feature, much like HP sometimes sends out patches to fix certain MPE bugs? There were several reasons why we could not do this:

* Patching object code -- especially someone else's -- is hard. Object code is very hard to understand, and often it's difficult to tell if a patch you make might have an unexpected side-effect. (I say this now, in 1990 -- in 1980, it was even harder for me to deal with.)

* While patches can be used to delete chunks of code (by branching around them) or to make small changes, they cannot readily be used for **additions**. It's very difficult to insert code into a segment, and even more difficult to add calls to external procedures that the segment doesn't already call. To do this, you'd almost have to write a "program file editor" program that could manipulate program files, and though I know how to do it now, I didn't know how to do it then, and don't want to do it now even though I know how.

- Patches would have to be generated for every new version of the patched program that comes out, and we'd have to start almost from scratch for every such version (since the locations of the various pieces of code in the program and in each segment are likely to change quite radically, and the entire internal structure of this part of the program might change).

All in all, just patching object code was dangerous and difficult.

TRAPPING PROCEDURE CALLS

Fortunately, there was an alternative to patching code directly, an alternative that was pioneered (to the best of my knowledge) by Bob Green of Robelle. (Even if he didn't originate it himself, he's certainly the one from whom I adapted it.)

For space and performances reasons, the files handled by Bob's QEDIT text editor had their own special internal format; a program (like a compiler) that expected normal EDITOR-generated files would be quite surprised to get a QEDIT file. But, since QEDIT aimed to be substantially faster (as well as more powerful) than EDITOR, Bob didn't want to have to convert the QEDIT file to EDITOR format before each compile.

The thing that Bob took advantage of -- and I eventually did too -- is that **programs are not self-contained.** All of their dealings with the outside world -- with disc files, with the terminal, etc. -- are done through **intrinsics** (or some other system SL procedures). If only we could cause the programs to call our own procedures that would **look** like the system intrinsics but actually do our own stuff, too (e.g. pretend that QEDIT files are actually normal EDITOR files, or process user input before the program gets it in order to possibly execute it as an MPEX command), we could change the program's behavior without its even noticing. (The way that we would make the programs call our own procedures is by moving the programs into a separate group and putting procedures with the same names as intrinsics into the group SL.) For instance, if we could replace the READX called by a program by our own procedure that:

- accepts **exactly the same** parameters as the real READX;
- calls the real READX;
- checks to see if the user's input starts with a "%" character, and if so, passes it to MPEX to be executed as an MPEX command;
- returns **exactly the same** values as the real READX (including the condition code)

then a user will be able to execute MPEX commands from that program by just prefixing his input with a "%"; the program itself would not have to be patched, since all of the logic will be in our SL procedure.

Note how this approach avoids the problems of object code patching:

- We don't have to read object code, since all that we need to do is emulate the calling sequence of a well-documented procedure.
- We can easily add new functionality, since the SL procedure can be of almost unlimited size.

* We probably won't have to worry about new versions of the program, since no matter what changes are made to the program, the program will probably still call READX in the same context and for the same purpose as it did before.

Of course, there are also some limitations with this approach:

* We can only alter those aspects of the program's behavior that are accomplished through external procedure calls -- internal calculations and checks will often be beyond our reach. For example, we can implement a multi-line REDO facility in EDITOR (since we can, by intercepting terminal input calls, record all the input that the user's given us and replace "redo" commands by the appropriate line of input), but we can't input, say, a new feature on a /CHANGE or /LIST command because we don't have any access to the EDITOR work file and all of the work-file-related tables that EDITOR keeps.

* Though we will know all the **parameters** of the procedure call, we may have very limited information about its **purpose** -- for instance, is this READX call intended to prompt for a command (in which case we want to process commands prefixed by a "%"), or for a line of text being added to the file (in which case we don't want this).

And, there are some practical difficulties that we need to overcome to make this approach fully workable -- more about them in the pages ahead.

WRITING THE PROCEDURE

The best way of discussing things further, I think, is to walk through a very simple example of "hooking" a program that you can actually do as you read the paper. Unfortunately, it'll be of limited use (since it was chosen for simplicity rather than utility), but it might still be somewhat impressive -- we'll "teach" LISTDIR5 to honor MPE commands prefixed by ":"s, so you can, for instance, say ":ALTSEC xxx" or ":NEWGROUP xxx" or something like that when prompted with the LISTDIR5 ">" prompt.

The first question that we must ask is

which system intrinsic call can we intercept to get the job done?

The answer to this is quite simple -- the READX intrinsic. Our plan of attack will be:

* Write a procedure with exactly the same calling sequence as READX.

* Call the real READX to get the input from the user.

* Check the input to see if it starts with a ":" -- if so, execute it as an MPE command using the COMMAND intrinsic.

* Return exactly the same results as READX would.

So, let's begin:

```
    $CONTROL USLINIT, SUBPROGRAM, SEGMENT=MY'READX
    BEGIN
    INTEGER PROCEDURE READX (BUFFER, LEN);
    VALUE LEN;
    ARRAY BUFFER;
    INTEGER LEN;
    BEGIN
```

The very first thing that you notice is: this is written in SPL. What, you say that you don't know SPL? Well, that's perfectly understandable, but unfortunately there are two crucial things that a hook procedure needs to be able to do that just cannot be done in some languages:

- Accept as input virtually any kind of parameter -- word address or byte address, by value or by reference.

- Return a condition code.

To the best of my knowledge, only SPL and PASCAL procedures can take by-value parameters (like READX's LEN parameter); in MPE/V, only SPL procedures can return a condition code (though MPE/XL's HPSETCCODE intrinsic permits other languages to do this on MPE/XL).

However, with the following procedure:

```
    PROCEDURE VESETCCODE (I << 0 = CCG, 1 = CCL, 2 = CCE >>);
    VALUE I;
    INTEGER I;
    BEGIN
    INTEGER ARRAY Q(*)=Q+0;
    Q(-Q(0)-1).(6:2):=I;
    END;
```

you can set the condition code from, say, a PASCAL procedure, as long as you call it (VESETCCODE) from the hook procedure itself and **not** from any of the procedures called from within it.

Armed with VESETCCODE, there's no reason why you can't write hook procedures in PASCAL on MPE/V (in fact, I'll even use it in my SPL examples) though I think that you still can't do them in any other language.

OK, back to our sample procedure. Note that we created a procedure header that **exactly** corresponds to the calling sequence of the READX intrinsic. **Each** of the parameters must match exactly, both in **type** and in **mode** (by value/by reference); the return value must be exactly the right type, and if the procedure we're intercepting is an OPTION VARIABLE procedure, so must ours be. (PASCAL programmers: you can still hook OPTION VARIABLE procedures if you realize that an OPTION VARIABLE procedure is just the same as a normal one but has an extra by-value parameter at the end that contains the OPTION VARIABLE bit mask.)

Now, let's continue:

```
    $CONTROL USLINIT, SUBPROGRAM, SEGMENT=MY'READX
```

```
BEGIN
INTEGER PROCEDURE READX (BUFFER, LEN);
VALUE LEN;
ARRAY BUFFER;
INTEGER LEN;
BEGIN
INTRINSIC READX;
BYTE ARRAY BUFFER'B(*)=BUFFER;
INTEGER LEN'READ;
BYTE ARRAY TEMP'CMD(0:255);
INTEGER CIERR;
INTEGER FSERR;

LEN'READ:=READX (BUFFER, LEN);
IF > THEN VESETCCODE (0)
ELSE IF < THEN VESETCCODE (1)
ELSE
  BEGIN
  IF LEN'READ<>0 AND BUFFER'B=":" THEN
    BEGIN
    MOVE TEMP'CMD:=BUFFER'B(1),(LEN'READ-1);
    TEMP'CMD(LEN'READ-1):=%15;  << carriage return >>
    COMMAND (TEMP'CMD, CIERR, FSERR);
    END;
  VESETCCODE (2);
  END;
READX:=LEN'READ;
END;
```

As you see, we call READX, check the condition code, set our own return condition code appropriately, and if the read succeeded and the input line starts with a ":", call the COMMAND intrinsic.

What is wrong with this picture? Well, there are three problems:

* First, and most important of all (I'm sure you noticed this), we have our procedure called READX calling the intrinsic READX. You'd think that since you declared READX as an intrinsic, the compiler will recognize that you want to call the READX intrinsic in the system SL.

  This, unfortunately, is not the case. When the linker sees the call to READX, it views it as a recursive call to our own procedure and not a call to the READX intrinsic. (To make matters worse, the SPL compiler will not flag the "INTRINSIC READX" declaration as a duplicate symbol error.) In fact, we will find that this -- how to call the real procedure from our hook procedure -- is one of the more substantial problems that we face.

* Secondly, note the `MOVE TEMP'CMD:=BUFFER'B(1),(LEN'READ-1)` statement -- why is it wrong? Because the way that the READX intrinsic is defined, its result (which we put into LEN'READ) may be the number of bytes or the number of **words** read (depending on whether the LEN parameter was negative or positive). Actually, we might discover that LISTDIR5 always passes a negative LEN parameter and thus the READX result will always be in bytes, but we don't want to count on that (especially if we want the hook procedure to be general). The rule is thus that you

must be prepared for any possible set of input parameters and any possible result returned by the intrinsic.

In other words, instead of LEN'READ, we should have said `(IF LEN<0 THEN LEN'READ ELSE 2*LEN'READ)`.

* Thirdly, what happens when a command that's prefixed by a ":" is input? Indeed, it will be executed as an MPE command, but it will then be returned to LISTDIR5 as the result of the read -- LISTDIR5 will see it as an invalid command, and will output a nasty message.

  This is important to remember -- when you intercept a procedure call, from the program's point of view the call still completes, and the program will act upon the data returned by the hook procedure. In this case, we should make sure that the data returned to LISTDIR5 is such that LISTDIR5 will do as little with it as possible -- in LISTDIR5's case, returning an empty string (just as if the user hit return). For this, we'd have to set the function result to 0 (0 characters read), but we'd also have to make sure that the buffer returned to the program is in the same state as it was when our procedure was called, since programs often calculate the length of the data input not by the READX result, but by the position of some terminating character (e.g. a carriage return) that they filled the buffer with.

CALLING THE REAL PROCEDURE

Let's get back for a moment to the first problem mentioned in the above list -- if we call the intrinsic READX from our READX procedure, we get an infinite loop. What can we do about this?

Well, there are three possible solutions:

* Since we're putting our READX in a group or account SL anyway, we can put an "intermediary" procedure called, say, INT'READX into the system SL (or any SL higher than the one in which our READX is) -- our READX can call INT'READX, which will then call READX. Since SL's are always searched in the order group, account, system, a call to READX from an INT'READX that's in the system SL will **not** call back to our group/account SL READX, but rather go to the real READX in the system SL.

  The INT'READX procedure might then look something like this:

```
INTEGER PROCEDURE INT'READX (BUFFER, LEN);
VALUE LEN;
ARRAY BUFFER;
INTEGER LEN;
BEGIN
INTRINSIC READX;
INT'READX:=READX (BUFFER, LEN);
IF > THEN VESETCCODE (0)
ELSE IF < THEN VESETCCODE (1)
ELSE VESETCCODE (2);
```

```
END;
```

* Another alternative is to have our READX call the real READX using the LOADPROC intrinsic. Among other things, LOADPROC lets you specify that you want to load the procedure from the system SL, so we won't get into the same recursive loop that we would have had if our READX just tried to call READX directly.

  I won't go into any more detail as to how this is done, but I do want to point out that one problem with this approach is where to put the plabel returned by the LOADPROC procedure so that we don't have to re-LOADPROC for every procedure call. Actually, for intercepting READX we can afford to re-LOADPROC the real READX every time our READX is called because subsequent LOADPROCs of a procedure that's already been LOADPROCed take only a few milliseconds; however, if we're intercepting a more time-critical procedure, like FREAD, we'd have to be sure to LOADPROC it only once, in which case we'd need some global storage to keep the plabel. More about the global storage problem later.

* One other alternative that's worth considering is somewhat more difficult to do but cures a very substantial disadvantage present in the first two solutions.

  Say that you want to hook a number of programs all over your system, for instance to call your own DBOPEN replacement procedure instead of the DBOPEN intrinsic (which we do in our SECURITY/3000 VEOPEN module), or to call a replacement COMMAND procedure (which we do in our SECURITY/3000 STREAMX module). If you want to intercept these calls using procedures named DBOPEN and COMMAND, you'd have to put these procedures into local group or account SLs in every group or account in which the programs to be hooked reside. This can prove quite cumbersome, especially when it comes time to install a new version of your procedures -- you might have to replace dozen of SLs in dozens of different accounts. The trouble, of course, is that you can't put your own hook procedures into the system SL, since they have the same names as the real intrinsics.

  The way that you can get around this problem is by actually **patching** all the programs to be hooked to call not a procedure called DBOPEN, but rather one called, say, VEOPEN. Then, you can put the VEOPEN procedure into the system SL, since it will no longer conflict with the real DBOPEN -- furthermore, since it is called VEOPEN, there'll be no problem with it calling the real DBOPEN without threat of recursion. When a new version of VEOPEN, incidentally, is installed, you won't have to re-patch all the programs, but only replace the module in the system SL. (On the other hand, whenever you roll in a new version of a patched program, you'd have to re-patch it.)

  Patching the programs might at first glance seem difficult, but it actually isn't. All program files (I'm speaking here of MPE/V and CM programs) contain at a well-defined place an "external reference list", which is a list of the names of all the SL procedures that they call (together with some other information). Simply by replacing the procedure name "DBOPEN" by the name "VEOPEN" you can make the procedure call VEOPEN instead of DBOPEN. (Note that the two procedure names are intentionally chosen to be the same length.) The layout of this table is described in Chapter 10 of the MPE/V System Tables Manual -- it's not that hard to write a program that modifies it. It's not much more difficult (though it is extra work) to write a program that modifies the external reference list of an SL segment.

All in all, we've found patching to work quite well for us, but the additional cost of writing a program to patch the external reference list might make it a rather expensive solution for some.

GLOBAL STORAGE

So far, with the READX and INT'READX procedures, we've done pretty much what needs to be done to get our new-and-improved LISTDIR5 to work. All we need to do is:

- Copy LISTDIR5 into our own group (it'll have to have PM capability, but that's just because LISTDIR5 itself needs PM).
- Add the READX procedure (as finally corrected) to the group SL.
- Add the INT'READX procedure to the account or system SL.
- :RUN LISTDIR5.ourgroup;LIB=G

If we've done everything right, our toy should work just fine; we can even move other programs (e.g. DBUTIL) into our group and have the very same procedure work for them.

Unfortunately, one of the reasons why this was so easy (you did think it was easy, didn't you?) is because the problem that we set for our ourselves was quite easy. The feature that we wanted to implement could be implemented entirely within one READX call; we didn't need to save any information from one call to the next.

What if we did need to save information this way? For instance, if we wanted to implement a multi-line REDO, we'd have to save some information (e.g. the file number of the REDO file) from one READX call to another – we'd also need to be able to tell when our READX is called for the first time, so that we can initialize this information. (Actually, a number of useful features -- like SECURITY/3000's VEOPEN and STREAMX's interception of the COMMAND intrinsic -- can be implemented without using global storage, but many other features can't be.)

SL procedures in MPE/V are **not** allowed to have global storage of their own -- if you try to add a procedure that uses global or OWN variables to an SL, it will fail. Procedures that have the cooperation of the caller (like the V/3000 intrinsics) can get around this by having the caller pass them an array that contains the data that they need to preserve from call to call (e.g. the V/3000 VCOMAREA array), but we don't have that luxury since we must remain scrupulously compatible with the calling sequences of the procedures we're intercepting.

Where can we put our global data? There are a number of places in which MPE lets us keep information that won't vanish from one procedure call to the next, but all of them have their own problems:

- Files and extra data segments -- you can put a lot of data here, but it's rather slow to access (even a few milliseconds per call can be slow when we're intercepting a frequently called procedure like FREAD, though it can be acceptable for, say, READX, DBOPEN, or COMMAND, which take much longer anyway). Furthermore, you still have to find a place to keep the file number or extra data segment index!

* JCWs -- these are also rather slow, and they can only contain a single word each. Furthermore, they're session-local rather than process-local, so multiple hooked processes in the same session might have trouble.

* The "DL-to-DB" area of the stack -- this can be accessed very quickly (since it's just as much part of your stack as your procedure-local variables), but is often already used by the hooked program (especially if it calls V/3000 or uses PASCAL's "heap" mechanism). There are a few words a little bit below DB (DB-10 through DB-1) that are often not used by most programs, especially programs written in SPL, but again it's possible that the program you're trying to hook uses them. This is especially relevant if you're trying to write a general-purpose hook routine that is supposed to work for all programs -- in fact, the first version of our MPEX hook routine used one of these DL-negative words until we ran into a program that wanted to use it, too.

As you see, this is not a pretty sight. There are things you can do with one or more above mechanisms that might work in your case (especially if speed is not a problem), but there doesn't seem to be a very good general solution.

The best solution (pioneered by Bob Green) is somewhat difficult to implement but ultimately far superior to any of the above. As we mentioned before, Bob's QEDIT text editor was written with efficiency very much in mind, and when he decided to have compilers read QEDIT files, it was very important that they do this as fast as possible. One of the key procedures that he needed to intercept was the FREAD intrinsic (which often takes only a couple of milliseconds), so the access to his global storage had to be as fast as possible. He pretty much had to have all the global storage be kept in the stack.

To understand how this approach works, one has to realize what a program file contains. A program file is essentially a blueprint for the loader that describes how to load the process. It contains information on all the code segments (which is to go into the CSTX), the names of all the external references (which are to be loaded from SLs), and an image of what the process's stack is to look like when it starts up. All the initial values of all of the program's global variables are kept here, and when the loader loads the program, it allocates the right amount of global area (the size of the global area is also kept in the program file) and fills it with these initial values.

Bob was already patching the program file's external reference list (see the discussion above), so he decided to expand the program's initial global values area to include room for his own global values. Since the program by definition didn't use any of the global area beyond what it thought was available, his storage wouldn't collide with the program's storage; and, he could add as much space as he needed (keeping in mind, however, that if the program already used a lot of stack space, this might cause stack overflow problems).

So the the general plan -- again, you might want to look at Chapter 10 of your System Tables Manual for this -- was:

* Modify the "global area size" word in record 0 to indicate that there is more global area.

* Insert as many records as needed after the global area (and before the code segment area) in the program file, initializing them to whatever values you wanted to initialize them to. The insertion, of course, has to be done by creating a new copy of the program and copying all the data from the old program.

* Modify the record numbers (in record 0) of the segment area, external reference list area, and entry list area to reflect the fact that we've inserted records.

* Set the record number of the FPMAP area to 0, since unfortunately the FPMAP area of the program contains a lot of internal pointers with record numbers, and rather than readjusting them all, you should probably just tell the system that there is no FPMAP in this program.

For example, if the old program used 5000 words of stack space and you wanted to have 256 words of your own, you'd change the global area size to 5256, insert 2 records (256 words, 128 words per record) at the end of the global area, and increment the segment area, external reference list area, and entry list area record numbers by 2. It seems like a fairly complicated manipulation, but it really isn't; armed with Chapter 10 of the System Tables Reference Manual, you can do it quite easily.

There is one more problem to be dealt with. You run the patched program and it has 256 extra words of global area; but how does your hook procedure know where those words are? You can't just hardcode the address into the procedure, since you'd like it to work for various programs (and in any event the end of the global area of even one program will probably change from version to version). Instead, here's what you can do:

* When you insert your global area, make sure that the part that you want to use starts on, say, a 128-word boundary. For instance, in our 5000-word-global-area program, you'd make sure that your 256-word global area starts at word #5120 (the first multiple of 128 after 5000) -- thus, you'd expand the global area to 5376 words and just waste the 120 words between words 5000 and 5119.

* Set the first few words of the non-waste part of the data you insert into the program to some unique pattern that's not likely to appear in a normal program's global area. (Remember that the data that you insert into the copy of the global area in the program file will make its way into the program's stack.)

* In your hook procedure, try to find this unique pattern by looking at word 0, then word 128, then word 256, etc.

This way, you find your global area by the unique pattern that you've initialized its first few words to, but you don't have to check every word in your stack (which would take too long) because you know that your global area starts on a 128-word boundary. An example of this in SPL might be:

```
INTEGER POINTER IP;
@IP:=0;  << make IP point to DB+0 >>
WHILE IP(0)<>123 AND IP(1)<>456 AND IP(2)<>789 AND IP(3)<>555
DO
  @IP:=@IP(128);
```

Note that your unique sequence must be a sequence that's highly unlikely to ever appear in the program's stack; if, for instance, you choose a normal piece of text, it's possible (though unlikely) that this piece of text will somehow appear in the program's stack at a 128-word boundary (perhaps input from the terminal or a file) and will thus make you find the wrong area. I use a fairly unlikely sequence of 5 words, many of which represent unprintable ASCII characters.

WHICH PROCEDURES TO PATCH

The preceding discussion assumed that you knew exactly which procedure is to be patched, e.g. READX in LISTDIR5. Unfortunately, things aren't always quite this simple.

Most tasks can be performed by a program in different ways. Some programs, for instance, use READX to read from the terminal, but others (like SPOOK) use READ, and others (like EDITOR) use FREAD. When writing our "MPEX hook" procedures, we wanted to work with all of these programs, so we needed to hook all of the procedures. Hooking READ was quite simple, since it is very simple to READX, but dealing with FREAD was more difficult, because it was used by EDITOR to read both from the terminal and from files. We wanted to have terminal input that was prefixed by "%" be executed as MPEX commands, and we wanted to save terminal input in the multi-line REDO history, but we obviously didn't want this done to, say, lines from the file that we were /TEXTing in.

Our first thought was to call FGETINFO inside each execution of our FREAD hook procedure to see if we were reading from the terminal or not, but this was far too inefficient -- imagine calling FGETINFO for each line of a 10,000-line long file. Instead, we found ourselves having to hook FOPEN calls just so that we can check once per file open whether we were opening $STDIN or $STDINX, and recording this information for each file -- then our FREAD hook could just look into this array of flags to see if this was a terminal file or not.

Similar problems arise when programs use other mechanisms for reading from the terminal -- programs written in PASCAL often use PASCAL compiler library routines to do terminal I/O; these routines can be quite difficult to hook simply because, unlike intrinsics, their calling sequences are undocumented.

The problem of FREADs from the terminal vs. FREADs from files is actually a symptom of a greater problem -- what we really want to distinguish is not terminal vs. file input, but rather input of commands (which might come from files, e.g. /USE files) from reading of data (which might come from the terminal, e.g. in /ADD mode). We really want to distinguish FREAD calls based on what EDITOR intends to do with the data read, which unfortunately we cannot do, since the essence of the problem is that EDITOR isn't cooperating with us and isn't telling us anything about what it's doing.

We might try to tell which FREAD call is which by looking at where in the program the FREAD is being called (we can get this information from our hook FREAD procedure's stack marker), and seeing if it's one of those locations in which EDITOR does command input; unfortunately, this leaves us with almost all the problems that would be involved in directly patching the program's code -- we'd have to read the object code to find all the right locations to patch, they would only apply to this particular program, and they would have to be recalculated for each new version of the program.

Fortunately, sometimes, you can get information as to the "purpose" of a call in surprising places -- for instance, you may find that a particular program reads command input by passing a read length of -80 to READX but reads /ADD-mode input by passing a read length of -255, and thus use the read length to distinguish the two kinds of input.

To keep your hook procedure general, you might even want to keep the expected read length as a value in the global area that the program that you use to hook other program files inserts, and have this "hooking program" prompt for what expected read length is to be put into the global area. This way, you can, at the time you hook a program, communicate various special attributes of the hooked program to the hook procedure that will be called from this program.

CONCLUSION

In part, this paper is more a discussion of an interesting type of problem solved in an interesting way than a blueprint for your own development -- not everyone has the needs described in this paper or the means to satisfy these needs.

However, various people in the HP 3000 community have, more or less independently, used these techniques to accomplish some very valuable things:

* Robelle has gotten compilers to read QEDIT-format files.

* Various people have intercepted IMAGE calls to instead go to their own extended-IMAGE utilities.

* VESOFT has used hooking to implement MPEX command execution and multi-line REDO from EDITOR, TDP, etc., to allow an SM user running some such editor to save files across account boundaries, to preserve the ACDs of files being edited, to implement an IMAGE database security system by hooking DBOPEN, and to intercept COMMAND intrinsic calls and to route executions of the STREAM command through STREAMX.

There are a couple of relatively simple things that come to mind that you might do yourself:

* If you have your own internal data storage format, you can hook your favorite text editor to be able to properly read those files.

* If you want to disallow people to execute certain MPE commands from a program that normally allows MPE command execution (e.g. EDITOR), you can hook it to reject those commands.

* If you want to implement a control-Y trap in a program, you can hook some procedure that the program calls at the very beginning and have your hook procedure arm the control-Y trap.

If you really want to do something substantial, I believe that you can hook QUERY to handle MPE and KSAM files by intercepting all the DBxxx calls to make the MPE and KSAM files look like IMAGE databases. This would be truly a feat.

TITLE: Automated Testing: One Company's Experience

AUTHOR: Eugene Volokh

Vesoft

1135 S. Beverly Drive

Los Angeles, CA 90035

213-282-0420

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 3142

POSIX And X Windows On MPE XL

Scott Cressler
Ryk Langdon

Hewlett-Packard
19447 Pruneridge Avenue
Cupertino, CA 95014

As part of HP's commitment to open systems computing, two application programming interface standards, POSIX and X Windows, will be implemented on MPE XL. POSIX is the acronym for Portable Operating System Interface for Computer Environments, the International Association of Electrical and Electronic Engineers (IEEE) 1003.1 Standard. POSIX defines a programmatic operating system interface as well as some of the behavior of the functions it contains. The X Windows System, maintained by the Massachusetts Institute of Technology, defines primitives as well as higher level tools to support a windowing environment. Both standards are currently core application interfaces for programmers writing open systems applications and their use is expected to grow rapidly during the 1990's.

The addition of POSIX and X Windows to MPE XL will permit HP3000 users to take advantage of a wider range of application solutions to meet their computing needs. Applications that conform to these standards will be more easily ported to the HP3000. For HP3000 users, the benefits of these new interfaces will be two-fold. First, application portablility will mean an increase in the number of applications available for the HP3000. Second, POSIX and X Windows will afford an expanded functionality set on the HP3000 for application writers.

The intent of this paper is to introduce POSIX and X Windows and their enhancement of the MPE XL environment. While all interfaces and features of these standards will not be covered, examples will be used to introduce major functional components and their use. The discussion will present the new open systems options available to HP3000 application writers and the differentiation between MPE XL and the features offered by these standards.

POSIX

MPE XL will provide the IEEE 1003.1 standard for operating system interface functions. The goal of this standard is to provide a set of functions which provide application portability in the area of operating system interfaces. If an application interfaces with a system only through the POSIX interfaces, porting an it to another POSIX system should require only that the application be recompiled on the new system.

The POSIX interface functions are similar to MPE XL intrinsics in that they are programmatic interfaces to the operating system for application developers. Several examples of MPE XL intrinsics and POSIX functions are made in the discussion below to assist in understanding POSIX. However, there are many aspects of POSIX that are conceptually quite different than MPE XL. These features offer significant new potential for MPE XL applications.

This discussion of POSIX will cover four of the functional areas into which the standard naturally falls. These are Process Management, Signals, the Hierarchical Directory, and File System.

POSIX TERMINOLOGY

The definition of the following terms are provided as a reference for the POSIX discussion:

Directory File -- A file which is used in implementing the POSIX hierarchical directory structure. Entries are created in this file which point to files and other directory files.

Environment Variables -- These are strings which are passed in to a program when it is executed. They are similar to MPE XL Command Interpreter (CI) variables and JCWs, but they are local to a process and are not associated with a CI.

File Modes -- The file modes are a set of bits which define the access security of a file. If the file is executable, there are also two bits which determine whether the UID and/or GID of the file is to be used as the UID and/or GID of the process executing this file.

Group ID (GID) -- Each user on the system is a member of a group. The GID is a number which identifies that group. Each MPE XL account is assigned a unique GID when POSIX is installed. The GID is used when determining access to files.

Process Group ID (PGID) -- This is a number which identifies the group of processes of which a process is a member. A group of processes can be sent a signal using this identifier.

Process ID (PID) -- A number assigned to a process which is unique among all processes on the system. This is very similar to the MPE XL Process ID Number (PIN).

Signal -- This is an interrupt which is sent from one process to one or more other processes.

User ID (UID) -- This is a number which identifies a user to POSIX. This is similar to the MPE XL user and account name combination. In fact, UIDs are assigned when POSIX is installed so that each MPE XL user has a unique UID. The UID is used when determining access to files. It is also used when determining to which processes a process can send signals.

PROCESS MANAGEMENT

POSIX defines several functions which allow an application to manage processes. Although there is still only one kind of process on MPE XL, the style of process management used by the POSIX interfaces is significantly different from that available through the MPE XL intrinsics.

Create A Process

POSIX defines a function called *fork*() which is used to create a process. It might be better to call this "cloning" a process, because the child process is an exact copy of the parent. File opens are duplicated for the child. When the child is activated, both the child and the parent will return from the *fork*() call and continue their execution from this point. The only difference between the parent and the child is the return value of the *fork*() function. *fork*() returns zero to the child and the process ID of the child to the parent. After returning from *fork*(), these values can be used to specify different processing for the child and the parent.

Execute A Program In A Process

The child process can follow a significantly different path than the parent by replacing the program it is running with another program. This procedure of executing another program within a process is available through a set of functions called the *exec* family. Rather than creating a process in which to run a new program, the *exec* functions replace the executing environment of the existing process with a new program.

The combination of a *fork*() followed by an *exec* function is analogous in effect to the CREATEPROCESS intrinsic. The POSIX sequence, however, retains file opens of the parent process. It also allows the flexibility to separate the functions of creating a process and executing a program in that process.

Wait For Process Termination

A process can use POSIX to manage itself and its children. Through the *wait*() and *waitpid*() functions, a process can wait for one or more of its children to terminate. It can then determine why the child terminated and take actions based on the result.

Process Information

POSIX functions exist that can be used by a process to obtain information about its parent, such as the parent's process ID (*getppid*()). There are several functions that allow a process to determine information about itself, including its own UID, GID and user name. A process can also modify its UID and GID with the *setuid*() and *setgid*() functions, respectively, assuming that the process passes the appropriate security checks.

Environment Variables

POSIX provides for the definition of environment variables for each process. The *getenv*() function can be used to obtain the value of these environment variables.

SIGNALS

Signals Concepts

Signals are a feature of POSIX which are used primarily for interprocess communication. One process can send a signal to one or more of the processes in its process tree. Signals provide only simple interprocess communication. They do not allow messages to be sent, but they can be used to inform a process of a certain event. In addition, they can be used by one process to control the execution of another process (there are signals to tell a process to stop and to continue its execution) or to terminate it.

POSIX defines several different signals, including *SIGCHLD* (child process terminated or stopped), *SIGABRT* (abnormal termination), and *SIGALRM* (timeout). Each time that a signal is received by a POSIX process, there is an associated action that will be taken. Each signal also has a default action, of which there are four: abnormal termination of the process, ignore the signal, stop the process, and continue the process if stopped.

The action for a signal is determined in one of three ways. A process may specify that the default action for a signal is to be used. A process may specify that the action for the signal is to ignored it. A process may also specify that the action is to process the signal with a handling routine which the application has previously installed. A signal handling routine is conceptually similar to a subsystem break trap handling routine as installed through the XCONTRAP intrinsic.

Sending Signals To Processes

The function provided by POSIX for sending signals to other processes is called *kill*(). It has this name because the default action for most signals is to terminate the target process (the process being sent the signal). The process will not be terminated if it has set the action for that signal to ignore the signal, if the process has installed a signal handling routine for that signal, or if the process is blocking the signal (see below for details on signal action and blocking). *kill*() can be used to send a signal to one process or all the processes in a specified process group.

Manipulating Signal Action

POSIX provides a function, *sigaction*(), which allows a process to set the action that is to be taken upon receipt of a particular signal. One of the possible actions is to ignore it. Alternatively, the process can install a handling routine which will be called if the signal is received. When a signal is sent to the process for which a handler has been installed, the process will "catch" the signal and execute the handler.

sigaction() can also be used to specify that the default action is to be taken for a signal. This would ordinarily be done when a process had previously used *sigaction*() to specify that either the signal received was to be ignored or a signal handler is to be called.

Manipulating Sets Of Signals

Certain POSIX signal functions (*e.g.*, *sigprocmask*() for signal blocking as discussed below), use the concept of a set of signals. A signal set is simply a data structure which is used by an application to define one or more signals. There are several POSIX functions that provide the caller with the means to manipulate signal sets. These functions can be used to initialize a set to all POSIX signals (*sigfillset*()) or to none of the signals (*sigemptyset*()). Functions are also available to add a signal to a set (*sigaddset*()), to delete a signal from a set (*sigdelset*()), and to test whether a signal is a member of a set (*sigismember*()).

Manipulating Signal Blocking

Most signals can be blocked by a process. The processing of a blocked signal will be deferred until the process unblocks it. At that time, all occurrences of the blocked signal will be sent to the process.

The POSIX *sigprocmask*() function allows a process to perform one of several blocking related functions. With this function, the process can add a set of signals to those currently being blocked, delete a set of signals from those currently being blocked, or specify a new set of signals to be blocked. *sigprocmask*() can also be used to retrieve the current set of signals being blocked, with or without modifying the existing set.

An Example Of Signal Sets And Blocking

The following sequence of function calls can be used to block all signals except one with a sequence of function calls. First, the process would make a call to *sigfillset*() to fill its signal set data structure with all signals. Then a call to *sigdelset*() would be used to specify one particular signal to be deleted from the signal set. Finally, a call to *sigprocmask*() would cause the process to block the resulting set of signals.

Examining Pending Signals

The POSIX *sigpending*() function allows a process to determine which signals are pending for the process. A pending signal is one which has been sent to a process that has stopped or is blocking the signal.

Waiting For A Signal

By calling the POSIX *sigsuspend*() function, a process can suspend its processing until a signal is received. In addition, the process can use *sigsuspend*() to specify a set of signals which are to be blocked until the process becomes active again. This feature can be used to limit the signals which will reactivate the process. The process will resume processing when it receives a signal for which it has installed a signal handler. If a signal whose action is to terminate the process is received, the process will terminate.

A process can pause its execution without setting the blocking mask. The POSIX *pause*() function provides this functionality. In fact, *pause*() is exactly the same as calling the *sigsuspend*() function without a blocking mask.

Scheduling An Alarm

A special function is provided by POSIX that allows a process to schedule the alarm signal to be sent to itself. After a specified number of seconds have elapsed, the system will send the process the alarm signal. Depending upon the action set for the alarm signal, the signal will either be caught and a previously installed signal handling routine invoked, be ignored, or will terminate itself (this is default action for the alarm signal).

An Example Of Using An Alarm

An application may need to have a process suspend its execution and wait, for some maximum amount of time, for a specific signal to be received. First, the process would install a signal handler for the desired signal. The process can then schedule an alarm signal by using *alarm*() and use either *sigsuspend*() or *pause*() to suspend the process' execution. If the proper signal is received within the time limit, the process will resume and call the signal handler. When the time limit has expired, however, the alarm signal will be sent to the process, the process will resume and then take whatever action had been set for the alarm signal.

Delaying Process Execution

POSIX provides a function called *sleep*() which allows a process to suspend execution for a specified number of seconds. If a signal is received during that time whose action is to execute a handler, the process will be reactivated. If a signal whose action is to terminate the process is received, the process will terminate.

This function is similar to the MPE XL PAUSE intrinsic except that a process which has called *sleep*() will be reactivated if a signal with a handler is received.

HIERARCHICAL DIRECTORY

Directory Concepts

POSIX provides a hierarchical directory into which files are organized. The POSIX directory consists of "directory files" that may contain entries for other directory files or for other types of files. A file is said to be "located" in a given directory file if there is an entry in the directory file which points to the file. The POSIX directory has a tree-like structure and there is a single directory file that is the root of the tree.

A POSIX directory file is analagous to a group in the MPE XL directory, which contains entries for files which are in that MPE XL group. MPE XL groups and accounts differ from POSIX directory files in that MPE XL accounts can only contain entries for groups, and MPE XL groups can only contain entries for files. In POSIX, each directory file can contain entries for other directory files as well as other kinds of files. An entry that is a directory file may itself have entries that are directory files (as well as other files). This characteristic gives rise to a structure with an arbitrary number of levels in the POSIX directory hierarchy.

The POSIX hierarchical directory provides more granular file organization than the MPE XL directory. That is, POSIX files and directories can be arranged to reflect an arbitrary logical relationship according to the user's needs. Related files can be placed in a single directory file and related directory files can be organised together in one parent directory. If a large group of related files needs to be divided into smaller groups, the files can be placed into separate directory files that are entries in a higher level directory.

POSIX file and directory file names are allowed to be longer than MPE XL file names, are case sensitive (*e.g.*, *abc* is a different file than *Abc*), and may contain the underbar, "_", and period, ".", characters, in addition to alphanumeric characters.

Files in the POSIX directory are located using a path name. The path name defines the route through the POSIX directory to a file. An example of a path name is */accting/smith/statusfile*. The leading slash, "/", indicates that the system is to start at the root directory file when attempting to locate the file. In this example, the system will look for the *accting* entry in the root directory. It will use that entry to find the *accting* directory file. The *smith* entry in the *accting* directory file will be used to find the *statusfile* file.

If the path name does not begin with a slash, the system will start looking at a location called the current working directory (CWD). This is a path name for each process that is prepended to all path names which do not begin with a slash. For example, if the CWD of a process were */accting*, the path *smith/statusfile* would find the file */accting/smith/statusfile*.

POSIX imposes security checks on each directory and other file types. This security is based on the UID and GID of the user attempting to access the file. This is similar to the fact that MPE XL file security is based in part on the MPE XL user name and account of the user accessing the file.

Directory Operations

POSIX provides several functions which allow an application to query directory files. Opening a directory is similar to opening a file. An application calls the *opendir*() function to open the directory file, returning a structure called a directory pointer. This directory pointer is used to refer to the opened directory file. The application can then sequentially read entries from the file, using *readdir*(). *rewinddir*() will allow the application to start reading from the beginning of the directory file. Finally, *closedir*() is called to close the directory file.

An analogy on MPE XL is to perform a LISTF programmatically, write the output to a file, and then open and parse the file.

Make A Directory

Directory files can be created using the *mkdir*() function. The file modes of the directory file may be specified at creation. Creating a directory file is conceptually similar to creating an MPE XL group, except that POSIX directory files can be created beneath other directory files, whereas only files can be created within MPE XL groups.

Remove A Directory

A directory file is deleted using the *rmdir*() function. The directory file must be empty to be deleted.

FILE SYSTEM

Open A File

Access to files through POSIX is similar to MPE XL. First, the file must be opened and then a file descriptor, similar to an MPE XL file number, is used for all further access. The POSIX *open*() function is used to open files for read, write, read/write, append or creation access. If the file being opened exists, the *open*() function will open it. Otherwise, *open*() will create a new file.

Create Or Rewrite A File

In addition to *open*(), the POSIX *creat*() function can be used to either create a new file or overwrite an existing one. Both *open()* and *creat*() return file descriptors if successful.

Close A File

Files which have been opened using the POSIX *open*() or *creat*() functions can be closed by the *close*() function.

Byte-Stream Files

The default type of file created through *open*() and *creat*() is something called a byte-stream file. The MPE XL concept of a file record is not imposed by the POSIX file system on this type of file. Instead, all data in the file is treated as a linear array, or stream, of bytes. Any number of bytes can be read from or written to the file.

FIFO Files

Although byte-stream files are the default file type created through the POSIX interfaces, they are not the only type of file which may be created. Another type of file in POSIX is a FIFO file. FIFO stands for "first in, first out". A FIFO file acts like a "first in, first out" queue in which the first data written to the file will be the first read out. POSIX FIFO files are very similar to MPE XL message files in that data from a FIFO file is removed from the file after it is read.

Link To A File

POSIX treats the name of a file differently than MPE XL. In MPE XL, the name of a file is an important part of its identity. Although the file can be renamed into a different group, it can only be reached by one name. In POSIX, a path name is simply a way to reach the data in a file and there can be more than one way to reach the same data.

The *link*() function provides for this multiple access by using a file's path name to create an entry (or link) in a directory file. *link()* can create such an entry in a directory other than the one in which the file was originally created. Files are just as accessible through path names created using *link*() as they are through the original path name. A file can be created, a second link added using *link*(), and the original access removed (using *unlink*() as described below). The only means of accessing the file would then be through the newly created link.

The concept of a POSIX link is similar to an MPE XL file equation in that they both allow a file to be found through a name which is different from the name with which the file was created. However, the similarity stops there. A file equation is temporary, session-local and may be disallowed in an HPFOPEN. POSIX links are actual directory paths to the file data. They are visible and accessible to other users on the system and are as permanent. They are as real as the original path to the file when it was created.

Remove A File

Files may be deleted using the *unlink*() function of POSIX. As the name implies, the main purpose of this function is to remove links to a file. If the link being removed is the last link to the file, the file space of the data is removed.

Rename A File

The POSIX *rename*() function allows an application to rename a file from one path name to another. This is essentially the same result as a *link*() to the new name and an *unlink*() from the old one.

rename(), though executed programmatically, is similar in function to the MPE XL RENAME command.

Get File Status

POSIX provides a pair of functions which allow an application to obtain POSIX information about a file. *stat*() and *fstat*() return such information as the file modes and the UID and GID of the file owner. The difference between these two functions is that *fstat*() takes a file descriptor of an open file, whereas *stat*() takes a path name.

The analogous intrinsics on MPE XL to the POSIX *stat*() and *fstat*() functions are FLABELINFO and FFILEINFO which return information about MPE XL files.

Change File Security

The POSIX *chmod*() function can be used to change the file modes of a file. The access security part of the file modes is separated into the access allowed to three types of users. These are the owner of the file, users which are in the same POSIX group as the owner of the file, and any other user. Access for read, write and execute can be allowed or disallowed for each of these user types. In the case of directory files, execute access controls the ability to search the directory.

Change File Owner

The user that owns a file, can change the user ID and/or POSIX group ID of a file using the *chown*() function. These identifiers, combined with the file modes, determine access to a file.

Create A Pipe

POSIX defines a concept known as a pipe. A pipe is similar to a FIFO file, except that it is transient and does not involve a file. The *pipe*() function is used to create pipes, which are especially useful as a way for a parent and child process to communicate. Once a pipe is created, the process will have two file descriptors - one for writing to the pipe and one for reading from the pipe. If a child process is then created through the *fork*() function, the child will have duplicates of these file descriptors. The parent may then write data to the pipe which can be read by the child, and vice versa, thus providing a process-to-process communication mechanism.

Read From And Write To Files

POSIX provides two functions for transferring data to and from files: *read*() and *write*(). The *read*() function reads a specified number of bytes from an open file into a user-supplied buffer. The *write*() function writes from a user buffer to an open file. Both of these functions operate on the file starting at the current position in the file associated with the file descriptor. These functions operate on byte-stream files, FIFO files, and pipes.

Reposition The Read/Write File Offset

By default, an application will read or write sequentially through a file. If an application must access bytes in a file in a non-sequential manner, it may use the POSIX *lseek*() function. This function adjusts the position in the file for the next *read*() or *write*().

Alter File Attributes

While a file is open, it may be manipulated in several ways by a POSIX function called *fcntl*(), which stands for "file control". One interesting use of this function is for file locking. The file locking provided by POSIX is more granular than that provided by MPE XL. Any portion of the file, from one byte to the entire file, may be locked. The MPE XL FLOCK intrinsic, however, only allows for locking of the entire file. *fcntl*() allows the application to lock a portion of the file with a shared lock (usually for reading) or an exclusive lock (usually for writing). *fcntl*() can also be used to unlock a portion of the file.

There is a significant difference between *fcntl*() file locking and MPE XL file locking using the FLOCK intrinsic. FLOCK will prevent access to a locked file through any intrinsic using what is known as "enforced" file locking. With this type of locking, once the data is locked by one process, a second process is unable to access the data until the lock is removed. *fcntl*(), however, provides only for "advisory" locking. With this method, the act of locking a portion of the file *does not prevent* another process from writing to or reading the locked data. With advisory locking, locked data is prevented by convention. A second process must use *fcntl*() to check the data for locks and then delay its access until it performs a check and the initial lock is no longer present.

Another use of *fcntl*() is to mark an open of a file to be closed if the process that owns the file later calls an *exec* function.

X WINDOWS

X Windows (or simply "X") is a programmatic interface standard for creating and utilizing graphical user interfaces on high-resolution, bit-mapped displays. The architecture of X Windows allows an application to create a graphical user interface without knowledge of the hardware used for the display. This makes the application more portable because it does not have to contain machine-dependent display information.

X Windows employs what is known as a client/server architecture, in which a software "server" handles requests for specific services from one or more "client" applications. An application using X is called an X client application. The process managing a bit-mapped display for use by X Windows applications is called an X display server. This separation of client and server functions allows display specific processing to be confined to the display server software. It is possible for several client applications to utilize the same display server simultaneously.

The client and server communicate user interface requests and information employing a protocol that can be used across a network. This allows X to support distributed applications. An application using X to control the user interface does not have to be executing on the same machine to which the graphic display is physically connected.

The client/server nature of X is particularly useful for an application which executes on MPE XL. By using X Windows, an application can be executing on MPE XL and presenting a graphical user interface on a display attached to another machine which is managed by an X display server. There are several options for such display server support, including the HP 9000/300 workstations and the HP 700/X family of X Windows Terminals.

There is a basic library of X Windows functions called Xlib. The functions in this library allow an application to make a connection with a display server. The application can then create a window on the display and perform graphic operations in the window, such as drawing circles and text. Xlib also allows an application to take user input from that window, through the keyboard and other input devices such as a mouse. This input and pertinent information are communicated to the application through the use of events. The main part of an X Windows application is a loop which gets the next event from X and processes it.

There is another library that is commonly used in conjunction with Xlib, since the functions in Xlib are primitive. That library, called the Xt Toolkit, allows an application to create an X user interface without having to write code to manipulate every aspect of the interface, as with Xlib. The application uses Xt to create and use something called a widget. A widget is the code necessary to manage a part of a user interface, such as a push button.

The use of Xt and widgets allows an application developer to create and manage an X user interface at a "higher level" than with Xlib alone. For example, rather than drawing an oval for a push button using Xlib and managing all interactions with that button, the developer can create and activate a push button widget. The widget manages the interaction between the user and the push button; the application just has to react when the widget determines that the button has been pushed.

The Xlib library allows an application to produce a distributed, graphical user interface. The

Xt library uses Xlib to allow an application to create and use widgets. Both of these libraries were designed to be "policy free", that is, not imposing any stylistic constraints on the interface. However, it is desirable to have a consistent interface style to make applications easier to develop and use. There is a library of widgets that has been developed by the Open Software Foundation (of which Hewlett-Packard is an active standards participant) called OSF/Motif, that provides this capability. When this set of widgets is used with Xlib and Xt, an application developer can produce a particular kind of user interface. This user interface can easily be made consistent in its appearance and behavior with other applications using OSF/Motif.

XLIB

The following discussion of Xlib is a preliminary description to help understanding some of the concepts and component of Xlib. The books listed in the Bibliography offer more complete descriptions.

Connecting To A Server

Before an application can use X Windows, it must make a connection to an X display server. An X display server is a process which manages a bit-mapped display for use by X Windows applications. The *XOpenDisplay*() function is called with the name of the display to open this connection. If this connection is made successfully, a structure describing the connection is returned from *XOpenDisplay*(). This structure is used by other X functions to communicate with the server.

Window Manipulation

A window, as far as the X Windows library is concerned, is a rectangular area of a bit-mapped screen. An X application can create a window, which communicates to the server that the application will use that area for its user interface. To make the window visible, the application needs to call an X function which "maps" the window. This creates a request to the X display server to display the window on its bit-mapped display.

Drawing Text And Graphics

Once an application has mapped a window it may draw to the window. First, however, it needs to create something called a graphics context (GC). A GC is basically a data structure which contains information about performing graphics, such as color and line width. By creating a GC, the application prepares the X server for having the application draw in the window. The application can then call the X drawing functions, passing them the GC and the window identifier. X functions are provided to draw arcs, text strings, lines, points, rectangles and pictures.

Event Handling

After initialization, the main function of an X application is to process X events. An event is notification sent by the X display server to the application that the user has interacted with the user interface of the application. When the user clicks the mouse button while the pointer is within one of the application's windows, an event is generated. Typing on the keyboard generates events as well. Events can also be generated indirectly by actions performed by the user on windows of other applications. For example, an "exposure event" is sent to an application if all or part of one of its windows has been exposed. A window is said to be exposed if another window which was obscuring it is moved.

Responding to these events is the way X client applications interact with a user. For example, the application might define that if a user clicks his or her mouse button in a window, a certain action is to take place. The application contains a loop which gets the next X event, using *XNextEvent*(), and if the event is a *ButtonPress* event, performs that action.

User Customization

X Windows provides a set of routines called the resource manager. Resources are pieces of information which are used to customize an X client application, such as colors, fonts and window sizes. An application usually has default values for this information, but the user can specify his or her own default values. This is done by putting entries in a file called the X defaults file. These are called defaults because they will be used for each invocation of the application unless they are overridden (*e.g.*, through input to the application as it is executed). The application can use the resource manager functions to retrieve the user default for a given resource.

XT TOOLKIT

Initialize The Toolkit

Before any Xt functions are called, an application must call *XtInitialize*(). This function will make a connection to an X display server and get the necessary resources for the application. It also creates a shell widget to form an interface between the remaining set of widgets making up the application's interface and the window manager.

Create A Widget

Widgets are organized into widget classes. Xt provides a function, *XtCreateWidget*(), to create an instance of a widget. Each widget created is an instance of that class of widget and has attributes in common with other widgets of that class.

"Realize" A Widget

The *XtRealizeWidget*() function is used to create the window in which the widget resides. Each widget needs its own window into which it can draw and from which it can accept input.

Manage A Widget

The widget must then be registered with a parent widget that will manage the widget. A parent widget manages such things as the size and position of its child (associated) widgets. There are several Xt functions that arrange for a widget to be managed.

Event Handling And Callbacks

Once a widget has been created, realized and its management has been arranged, the application may start handling events. Since many events will be handled by widgets, Xt provides a function which assigns events to the widgets which need to handle them. This is called dispatching the events and the function is *XtDispatchEvent*().

Since the main part of an X application is a loop which gets an event and dispatches it, Xt provides a routine called *XtMainLoop*() to provide this function.

Although the behavior of a widget is managed by the widget code, the application still needs some way to react to user input. This is accomplished through two features of Xt, event handlers and callback functions. An event handler is a user function which is called whenever a specified event occurs. This function is registered with Xt through a call to *XtAddEventHandler*(). This way, an application can intercept one or more events and react to them as appropriate. Callback functions are a feature supported by some widgets. Some widgets have lists of functions which are called when a particular condition occurs for the widget. For instance, many widgets allow an application to define a list of application-supplied procedures which are called when the widget is destroyed. This permits the client application to react when a widget is being destroyed.

Destroy A Widget

Widgets may be destroyed when no longer needed using *XtDestroyWidget*().

OSF/MOTIF

OSF/Motif is a set of widgets developed in conjunction with the Open Software Foundation. They feature a three-dimensional appearance. When used with the OSF/Motif Style Guide, these widgets can be used to create an application with a user interface that is consistent with other OSF/Motif applications. Consistent interfaces prevent users from having to relearn common application interactions across applications. This reduces the training time required for the user to learn new applications and allows users to easily move between applications. This becomes increasingly important as users have access to applications across computer systems. This section will briefly describe each class of widget provided.

Shell Widgets

Shell widgets are a class of widgets which are mainly used as an interface between an application and the window manager. Some of the shell widgets provided by OSF/Motif are widgets to support dialogs, menus, and general visible shells.

Display Widgets

Display widgets are widgets which are used either to display data to the user or to interact directly with the user in fairly straightforward ways. OSF/Motif provides a large set of display widgets for use in creating a user interface.

Several button widgets are provided to take user action commands. An *ArrowButton* is a button in the shape of a triangle which can be used as a directional arrow (*e.g.*, to allow the user to increase or decrease some value). A *DrawnButton* is a simple, empty button which may be drawn in by the application to create a user defined button. *DrawnButton* widget instances accept user input, but do not give visual feedback. A *PushButton* widget looks and acts like a push button. It can be labelled with text or graphics and changes its highlighting and shadowing when "pressed" so that it appears to depress. The *ToggleButton* consists of a text or graphics label displayed next to a small button. When the *ToggleButton* is pressed, it appears to stay pressed. This widget is used when the user is allowed to make a choice which needs to stay chosen.

A *Label* widget is used to display text or graphics but not to take any input.

A *List* widget allows the client application to present the user with a list of items from which he or she can choose.

The *ScrollBar* widget allows the user to indicate a relative position in some data. It consists of two arrow-shaped buttons at opposite ends of a narrow rectangular region. The user presses on these buttons to scroll in small steps. There is a smaller rectangle in the rectangular region which is called the slider and is used to show and directly effect the relative position in the data.

The *Text* widget allows an application to present a user with an editable field of text.

Container Widgets

Container widgets are widget classes which are designed to contain child widgets or display data. They are composed of other widgets.

The *DrawingArea* widget provides an application with an area to display graphics or text. It provides the ability for the application to define a callback to be called when the contents need to be redrawn.

The *ScrolledWindow* widget is used to display and allow a user to scroll through some data. The *MainWindow* widget is useful to an application as its main window. It is basically a *ScrolledWindow* with options for a menu bar and a command window. A command window is an area for entering commands which maintains a history list of the commands entered.

A *RowColumn* widget simply organizes its child widgets in rows and columns.

The *Scale* widget can be used by an application to communicate the value of some item within a range of values. It can also be manipulated by the user to indicate a desired value.

Dialog Widgets

Dialog widgets are used by a client application to interact with a user. The application should use a dialog to inform or warn the user of some condition, obtain data from the user, allow the user to specify configurations, or allow the user to make a selection from a list. Dialogs are provided to produce forms, file selection dialogs, error boxes, and more. There is a large number of dialog widgets available and the OSF/Motif documentation provides more detail on them.

Menu Widgets

OSF/Motif provides functions to simplify the creation of user interface components used to make selections among assorted items. These components include menu bars, pull down menus, and popup menus.

Gadgets

A gadget is essentially a widget without a window. This allows a gadget to operate with much better performance than its widget counterpart. One drawback of gadgets is that they can not be customized individually by a user since they do not support resources from defaults files. Instead, they obtain their resource values from their parent widget. Several gadgets are provided by OSF/Motif which provide the same functionality as the corresponding primitive widget. Gadgets are provided to create such frequently used user interface components as ***ArrowButtons***, ***Labels***, ***PushButtons*** and ***ToggleButtons***.

CONCLUSIONS

POSIX on MPE XL will provide a rich, expanded set of programmatically available functionality to MPE XL Value Added Business partners. This new functionality will include a hierarchical directory, a byte-stream file type, a different way for an application to manage processes, and a simple form of user interprocess communication (signals). POSIX byte-stream files will provide a standard data format and the ability to use files across systems. POSIX will also provide an industry standard, open interface to operating system functionality on MPE XL. This will allow for increased application portability, and the increased availability of open application solutions on MPE XL.

X Windows on MPE XL will provide an industry standard programmatic interface for implementing graphical user interfaces on high resolution, bit-mapped displays. It will allow for distributed applications, utilizing a variety of display hardware, that can be developed without the need to address display hardware specifics. Though the X Windows routines of Xlib and Xt libraries do not impose a particular interface style, consistent appearance and behavior are available using OSF/Motif. X Windows thus offers the benefits of a standard interface, while providing great flexibility for application architecture, display options, and interface style.

BIBLIOGRAPHY

Hewlett-Packard, "HP OSF/Motif Programmer's Reference", Hewlett-Packard Company, 1989.

Institute of Electrical and Electronics Engineers, "IEEE Standard Portable Operating System Interface for Computer Environments", IEEE, Inc., 1988.

Nye, Adrian, "Xlib Programming Manual, Version 11", O'Reilly & Associates, Inc., 1988.

Young, Douglas A., "X Window Systems Programming and Applications with Xt", Prentice Hall, 1989.

Analyzing MPE XL Performance: What is Normal?

Doug Grumann, Marie Weston

Hewlett-Packard
Performance Technology Center
8050 Foothills Blvd
Roseville, CA 95678

When HP Precision Architecture systems running the MPE XL Operating system were first released, there were few software tools available which system managers could use to understand the performance of their systems. Luckily HP3000/900 series system managers now have more complete solutions at their fingertips. HP tools such as Glance and LaserRX are examples of reliable system performance monitors. Without the help such tools, a system manager can only make wild guesses when attempting to address performance problems.

There exist different performance tools to use when doing different types of performance management on your system. HP Glance is an example of a tool that lets the system manager know "what is going on right now". It is used in a diagnostic role to analyze performance problems on line. HP LaserRX is a system management tool which can analyze long-term trends in performance.

HP also offers custom performance consulting services. One of these services, HP Snapshot, was used to generate the data we're presenting today. The Snapshot tools take a comprehensive 1 hour collection on a system, and generate detailed performance reports.

Performance Terms

On any computer system, "throughput" can be defined as the amount of useful work being accomplished by the users. Throughput is quite hard to quantify. The definition varies on every system, and also may vary for different users on the same system. Though it is hard to measure throughput, there are indicators which can be measured on every system that strongly correlate to throughput.

The most useful indicator of Throughput for interactive workloads is Response Time. Response time is the time it takes for the system to complete a user's transaction. The problem with response time is that it is a very subjective measure of system performance: fast response times on one system may be considered too slow on another depending on the workload.

Does a transaction complete when the first characters are written back to the terminal in response to the completed read, or is a transaction complete when all the writes have finished and another read is posted? In different environments, these metrics can have varying importance. We call the metrics Time to First Response, and Time to Prompt.

Most interactive applications will post a terminal read and then wait for user input. This read can be a block-mode read or a character-mode read. When the user hits Carriage Return or Enter, the terminal read completes. The time between when the read is posted and when it completes is called Think Time, because this is the time the user spent "thinking" before starting a transaction.

There is an interesting mathematical relation between Think Time, Throughput, and Response Time. It is a derivative of "Little's Law", and looks like this:

$$R = (N / X) - Z$$

Where R = Response Time, N = the number of users on the system, X = the Throughput of the system, and Z = the average Think Time. For example, if your system had 120 users with an average think time of 10 seconds, and the system's throughput was 10 transactions per second, then your average response time would be = (120/10)-10, or 2 seconds per transaction.

Most system managers want to minimize response time and maximize transaction throughput. What are the factors that inhibit quick response times, and how many transactions can be completed on a particular system?

If there were no competition for system resources, response time would be limited only by system and application constraints. Most of our customers' HP3000/900 systems, however, are not single-user systems and their response times will be affected by competition for system resources. These resources can become "bottlenecks". It is tricky to define exactly what constitutes a bottleneck, but one can generalize that a bottleneck is the computer resource that the majority of the transactions are waiting on most of the time. On a heavily loaded system, it is usually not difficult to pinpoint a bottlenecked resource, given an effective performance monitoring tool.

Bottlenecks

On XL systems, one can define three major physical categories of potential Bottlenecks. They are: CPU, Memory, and I/O. We'll explain each in turn:

A CPU bottleneck occurs when there is no CPU idle time on the system, and applications are waiting on the CPU for time to run. This translates into processes in the "ready" state piling up in MPE XL's dispatch queue. It is important to realize that over a long interval (say, an hour) there may be lots of CPU idle time, but there could still be a CPU bottleneck IF within that period transactions were being delayed because of lack of CPU availability.

A Memory bottleneck occurs when processes cannot run efficiently due to memory limitations. Contention for memory space will force code and data to be "swapped out" to disk more often and processes will have to spend more time waiting on disk I/O. Since disk access is much slower than memory access, applications will run more slowly.

An I/O bottleneck is when the I/O system cannot keep up with the throughput demands of processes running. Queues develop on the I/O devices, and processes are forced to wait longer for their I/O to complete. In a truly I/O bottlenecked system, response times will be slow and the CPU will consistently have significant amounts of idle time. Disk devices are often the culprit in I/O bottleneck situations because disk access is shared among all processes active on a system, but MagTape, LAN, and Terminal I/O bottlenecks are possible too.

Because of the structure of MPE XL, CPU is the most common resource to be bottlenecked, which was not the case with MPE/V. Improved prefetching algorithms, mapped file access, and I/O subsystem improvements reduced the need for I/O bandwidth in XL. Since applications spend less time waiting for I/O, they place proportionately more demand on the CPU.

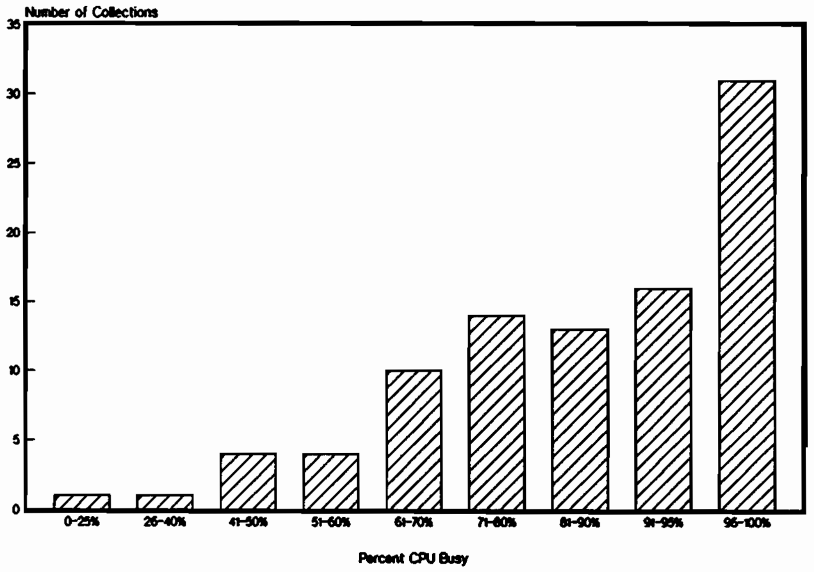
A special case bottleneck type which doesn't fit into the above categories is a "Lock" or "Logical" bottleneck. A Lock bottleneck is caused by many processes single-threading themselves on a software resource, often a Semaphore in MPE XL. A good example of this is Database locking. If many users need access to a database, and because of the application only one user can access it at a time, response time may be slow because processes are queued up waiting for the database to be unlocked.

There are several ways to lessen the severity of a bottleneck. We can reduce the amount of work being done on the system, which is not very likely to happen. We can increase the capacity of the overloaded system resource by upgrading the CPU, Memory, or I/O system hardware. This can be expensive! Finally, we can make our workloads more efficient in the area that we are bottlenecked, by reducing the demands on the bottlenecked resource per transaction. This final alternative can be the most rewarding. We will discuss techniques we can use to accomplish this at the end of this paper.

All performance tools are designed to help system managers determine where their systems are bottlenecked by collecting and presenting different metrics of system performance. Although we cannot discuss every available metric in detail, we have chosen to present a number of these metrics which represent each category of potential bottlenecks. In the following section, you'll see a graph of the distribution of values for each metric, based on statistics compiled from a database that contains detailed performance data from over 100 HP customer sites.

To avoid skewing the averages, we have included data from only those sites using an HP3000 950, 955, or 960. The samples represent averages over a one hour collection, which typically took place during high usage periods in the midmorning or midafternoon. Although this data was collected and compiled with specialized Hewlett Packard consulting software, there are various performance monitoring tools available on the market that will collect some or all of these metrics.

CPU Busy (Percent)



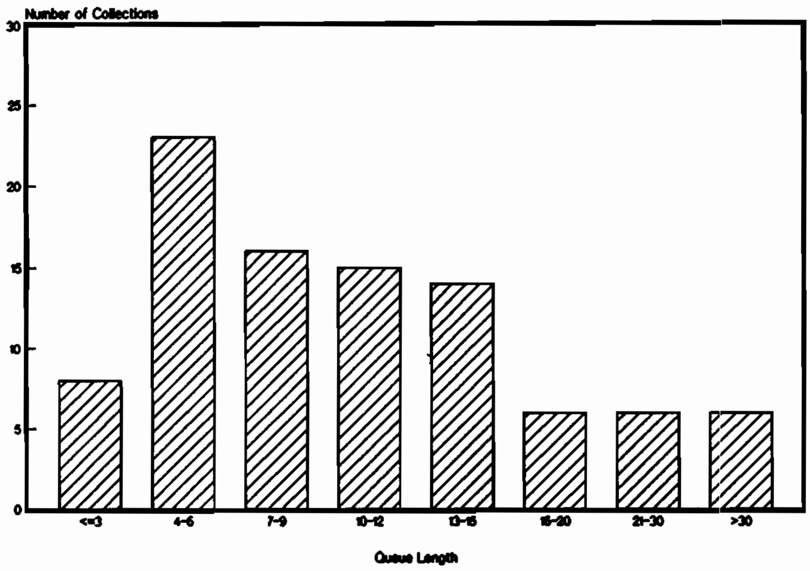
Percent CPU Busy

Average CPU Busy : 82.89%
Median CPU Busy : 89.47%

This metric shows the average percentage of CPU utilization for the duration of the collection. CPU utilization is the percentage of time that the CPU is either working on behalf of the system users or for the operating system. When this percentage is consistently close to 100%, this may indicate a CPU bottleneck. In other words, system performance may suffer because too many processes are demanding too much CPU time simultaneously. This could also indicate a memory problem.

The data shows that about half of the customer systems measured in our sample are above the 90% average CPU Busy level. These customers may be experiencing periods of poor response time due to excessive queuing delays during periods of high activity. Another possibility is that these customers may be simultaneously running low priority batch jobs that use up any extra CPU time and online response is not suffering from the high CPU utilization.

Average Dispatch Queue Length



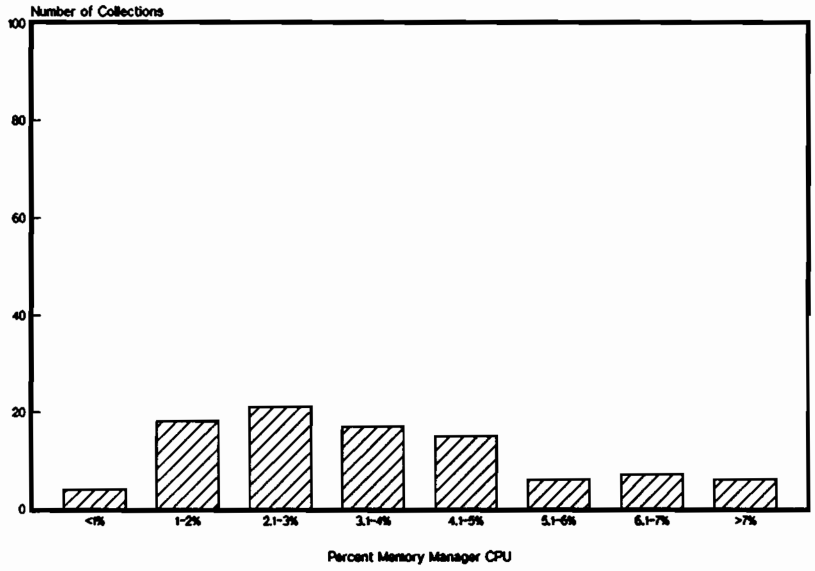
Average Dispatch Queue Length

Average length (across collections) : 13.8
Median length (across collections) : 9.0

This metric shows the average length of the dispatcher queue for the duration of the collection. The dispatcher queue is an ordered list of processes that are currently waiting for CPU time. Possible CPU saturation may exist when there are consistently more than 10 processes waiting for CPU.

Many customer systems in our sample have an average dispatch queue length of over 10. Since we saw that the average and median CPU utilization levels were fairly high–around 90 percent, we can surmise that the high CPU utilization percentage increases the length of the dispatch queue.

Memory Manager Percent CPU Busy



Memory Manager Percent CPU Busy

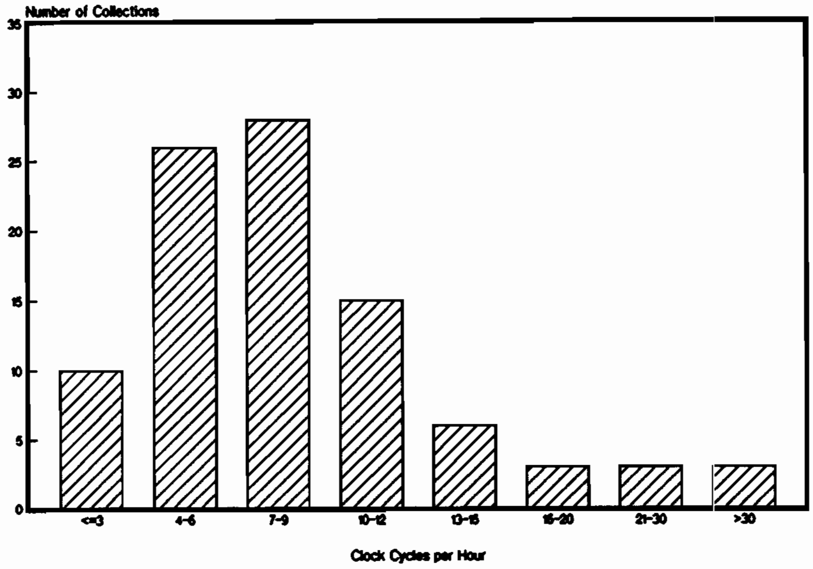
Average Memory Manager CPU : 3.55%
Median Memory Manager CPU : 3.25%

This metric shows the average percentage of CPU utilization on behalf of the memory manager for the duration of the collection. The memory manager is part of the operating system and it is responsible for keeping track of all available and in-use memory pages. It is also responsible for allocating pages to needy processes, deallocating pages when processes no longer need memory, and managing page swapping between main memory and disk when there is not enough memory to accommodate all processes.

If memory becomes constrained, it can increase overhead in CPU and disk utilization. When many processes are competing for a small amount of space, more CPU is needed by the memory manager to manage that space and more time is spent swapping pages to and from the disk, resulting in degraded performance.

Generally, memory may be a bottleneck when the average memory manager CPU utilization is over 8 percent. Our data suggests that most of our customers have a sufficient amount of memory (the sample average is 115 megabytes) since the average and median memory manager CPU are between 3 and 4 percent.

Clock Cycles / Hour



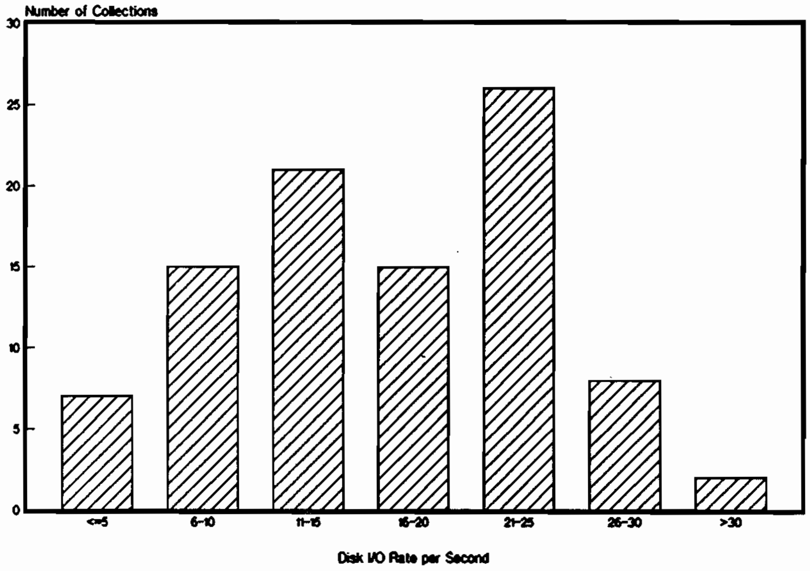
Clock Cycles / hour

Average Clock Cycles : 9.38 /hour
Median Clock Cycles : 8.00 /hour

This metric shows the average number of clock cycles per hour. A clock cycle occurs each time the memory manager completes a cycle through memory looking for recoverable overlay candidates (ROCs). In other words, the memory manager continuously cycles through memory to find pages that have not been referenced recently and, each time it completes one cycle through memory, the number of clock cycles is incremented. Too many processes running can decrease the number of ROCs and increase the number of clock cycles as the memory manager works harder to find space. If the clock cycle rate is consistently higher than 25 per hour, the system may be memory bound. Another term for this is "thrashing".

Once again our data suggests that most of our customers have sufficient memory since the average and median clock cycles per hour are far below the threshold of 25 cycles per hour.

Disk I/O Rate Per Second



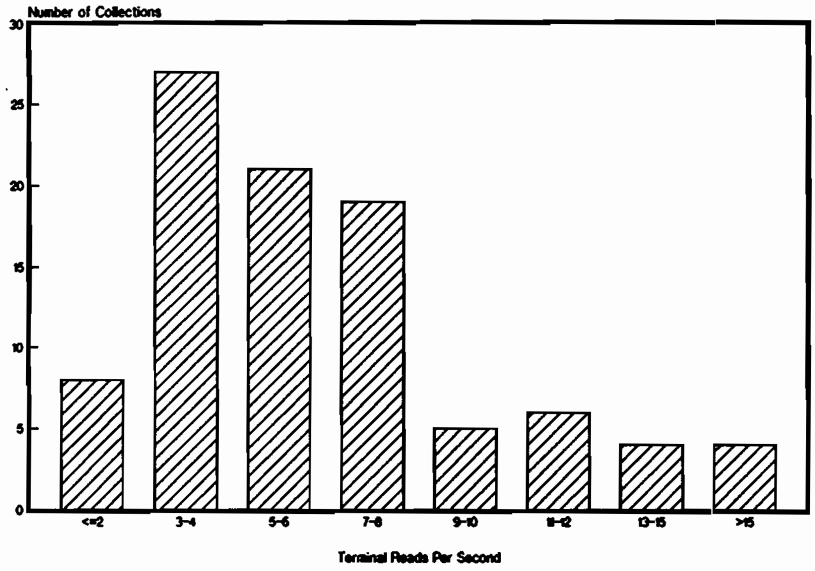
Overall Disk I/O / second

Average Disk I/O Rate : 16.37 /second
Median Disk I/O Rate : 15.90 /second

This metric is the number of disk I/O transfers per second. When there are too many processes performing disk I/O, these processes will experience poorer response time from the disk since all of the I/O requests must be queued for each disk. On MPE XL systems, this usually will not be a problem due to the way the memory manager "caches" portions of the disk space into main memory. However, if the disk rate does climb too high and begins affecting system performance, you may want to examine the workload to spread out those processes that access the disk extensively. One point to remember is that as new disks are added to the system, the disk I/O rate can climb without creating an I/O bottleneck.

We generally use 25 disk I/Os per second as a threshold for the overall disk I/O rate. The sample average for the total number of disks is 12. Our data shows that disk I/O is probably not a bottleneck for most of our XL customers.

Terminal Reads Per Second

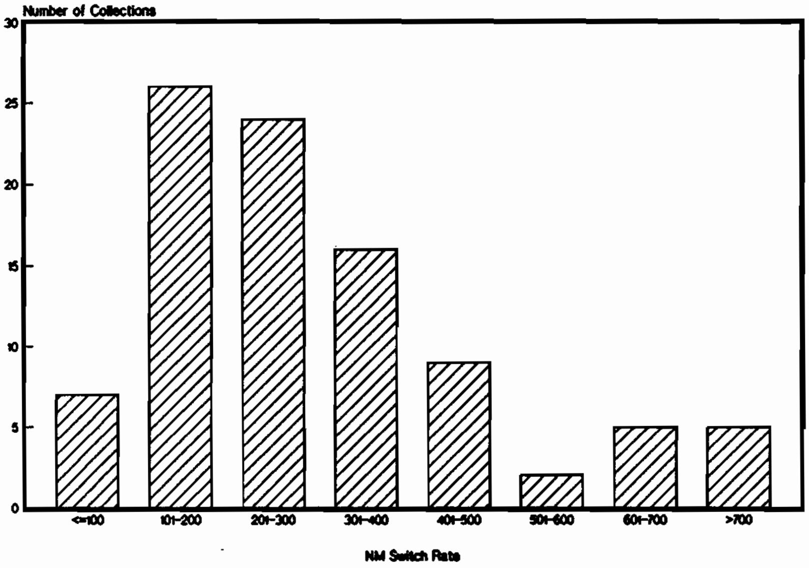


Terminal Reads / Second

Average Terminal Read Rate : 6.25 /second
Median Terminal Read Rate : 5.20 /second

This is one way to measure the transaction rate or throughput. A high transaction rate may be an indication that your system is performing well. With many applications, the time from one terminal read until the next terminal read is the time needed to complete one transaction. You may find it useful to periodically measure the terminal read rate to see how it compares with CPU utilization and response time.

Native Mode Switches per Second



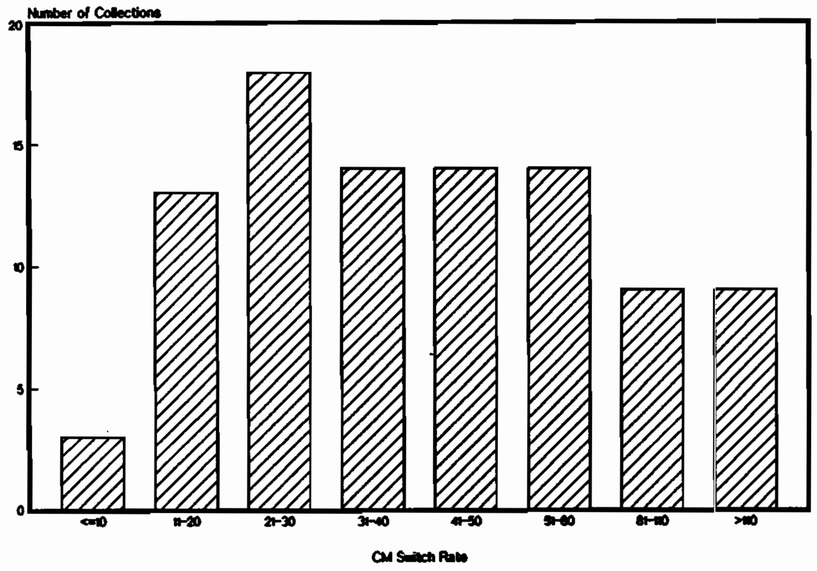
Native Mode Switches / Second

Average NM Switch Rate : 301.56 /second
Median NM Switch Rate : 270.10 /second

This metric is the number of times per second that compatibility mode processes switch to native mode. Since each switch requires a substantial amount of CPU time, a high switch rate can cause poor performance due to CPU saturation. In cases like this, native mode migration may be helpful in lowering the switch rate and the amount of CPU time required by the application.

The data shows that most of our customers in the sample have an NM switch rate of about 300 per second. Generally, when this rate climbs above 500 for a Series 950 and 650 for a Series 955, system performance may suffer.

Compatibility Mode Switches per Second



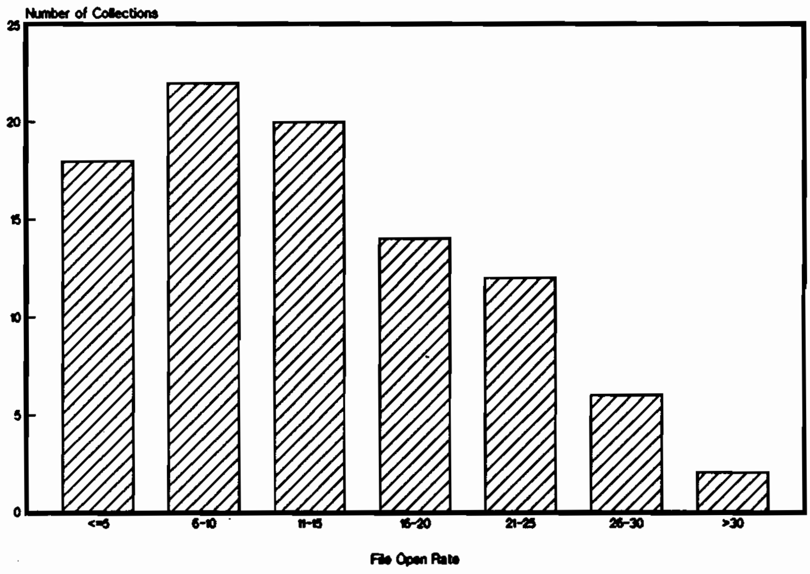
Compatibility Mode Switches / Second

Average CM Switch Rate : 50.57 /second
Median CM Switch Rate : 38.60 /second

This metric is the number of times per second that native mode processes switch to compatibility mode. As mentioned above, each switch requires CPU time, and a high switch rate can lead to performance problems. In cases where native mode processes are using services that are still in compatibility mode, it may be more beneficial to leave the original program in compatibility mode to keep the switch rate down and lower the total amount of CPU required by the application. An example of this would be KSAM applications. Since KSAM files must still be accessed in compatibility mode until MPE XL Release 2.1, we found some cases were native mode migrations actually hurt the performance of the application. Also, message files, circular files, and RIO files are still managed in compatibility mode.

Compared to the averages for Native Mode switches, which were around 300, these rates average considerably less. However, the CPU time required to switch to CM is considerably higher than an NM switch. You may want to monitor the CPU requirements of your NM applications that frequently access any CM system or user services.

File Opens per Second



File Open Rate / Second

Average File Open Rate : 12.93 /second
Median File Open Rate : 11.30 /second

This metric is the average number of files opened per second. Since opening a file takes a substantial amount of CPU time, an excessive file open rate can lead to CPU saturation. This problem may be alleviated by more efficient coding practices or adjusting the workload. A large file open rate may be indicative of an excessive process creation rate which is determined by the application structure. When possible, it is more efficient to use the Activate and Suspend intrinsics to control child processes as opposed to continually calling Createprocess.

Our data shows that most customers average about 11-12 file opens per second. You may want to periodically measure this rate on your system to see how it affects CPU utilization.

Adjusting your workload

We have seen that on many systems, high CPU utilization rates and long dispatch queues are the norm. Although these are accurate averages for the high-end HP3000 systems surveyed, they are guaranteed NOT to be "normal" for *your* system! Every system has its own unique combination of hardware, software, and user environments. It's very important that system managers concerned about performance get a feel for what is "normal" for their own shop. Track performance over an extended period to determine how your workload varies according to the time of day, and day of the week. This way, you'll learn to anticipate bottleneck situations that may cause your users grief, and you'll know what applications and resources on which to focus.

Begin by looking at your application mix, and the amount of the scarce resource consumed by each application. There is no sense in tuning an application that isn't contributing to a bottleneck! Some useful questions to ask yourself might be:

- At what times do the users feel the system is performing poorly?
- What applications are running during periods of high CPU utilization?
- How much batch activity is there during periods of high CPU utilization, and can the jobs be rescheduled?
- Is CPU time equally distributed among online applications?
- When can batch applications be scheduled without impacting interactive users?

Once you've become familiar with the performance characteristics of your own system, you'll know whether you're in a severe bottleneck situation. If this is the case, what can you do about it? This paper is not intended to be a "tips and techniques" forum, but there are general steps that one can take towards modifying workloads to alleviate bottlenecked resources.

- Redistribute your workload over different time periods (possibly by deferring batch activity) to avoid performance "spikes".
- Investigate problem applications: Should a CM program be converted to NM or be run through the Object Code Translator? Note that if an application spends most of its CPU time in system code (intrinsic calls), then converting the user code to NM may have little effect. Some NM programs may be better off in CM, if they are always requiring CM services (like Message or CM KSAM file access).
- Tune the dispatch queues to ensure even distribution of CPU time.
- Distribute disk activity as evenly as possible over the available disk drives. This would usually involve moving frequently used files off heavily utilized discs.
- If you have a Logical bottleneck, isolate what software resource is the cause of the problem, and change the application accordingly. Determine if file or database locking is occurring unnecessarily.

Again, remember that it's up to *you* to figure out what's normal on *your* system. Keep in mind that what's normal today won't be normal tomorrow, so you'll need to monitor trends over time. Spend your performance tuning time wisely: Concentrate on understanding your system and focus on bottleneck situations that result in degradation of throughput and/or response time. Proactive performance tuning will pay off in the long run!

Performance Contracts: Service Level Agreements in the Real World

By Rex Backman
Performance Technology Center
Hewlett-Packard Company
Roseville, California

Managing computer system performance is done most frequently after the fact. That is to say, we as users of computer systems do not usually think about the performance of our computer systems until the proverbial "straw has broken the camel's back". In the case of computer system performance the "straw" could be several things: a new program that is poorly designed, bad data base design, or worst of all diminished performance perception by our customers (i.e. our users). What ever the cause, something has happened that has made the previously translucent performance of our computer system quite visible for all to see. Unfortunately this is not the type of visibility most System Managers or Performance Analysts want. Activity at this point is geared toward fixing the performance problem in a reactionary role. Not the best of situations but something that all of us can relate to. Situations such as these can be resolved permanently. The answer is a Service Level Agreement.

A little common sense sprinkled with communication and negotiation skills will eliminate these negative reactionary performance problems. The solution is to create a pact that states the expectations of our customers relating to system performance. This is a Service Level Agreement (SLA). Simply put it is a contract between the supplier and a consumer of system services. The service provided is explained; expectations are stated regarding processing volume, system availability, etc; obligations of the supplier are set. It is a simple concept and with management commitment will make performance management a team oriented pro-active activity.

The Service Level Agreement is a written agreement between the supplier of the system services and the consumer of said services. As in any contract, the SLA spells out in enough detail what both parties expect. Thus, expectations are understood. No ambiguities regarding those nebulous performance terms such as response time and throughput. Think about it...makes life a lot simpler if all players are playing by the same rules. "Five second response time for each Cost Accounting update transaction" means exactly "Five seconds...". All parties have eliminated the possibility for confusion. No more "it was OK yesterday, but today seems a littler slower, what is wrong?" type calls to the Service Desk. Instead the absolute definitions in the SLA can be used to insure that "seems a little slow today.." is phrased properly as "I'm concerned the LaserRX graph for our Cost Accounting Application shows the response time exceeding five seconds towards the end of the month. Do we need to plan an upgrade or should we perhaps migrate to Native Mode ?".

OK, now that we understand the big picture of Service Level Agreements, let us get down to the details.

What is in a Service Level Agreement?

The Service Level Agreement states in sufficient detail the following:

a. Who is this agreement with. The supplier and the consumer of the service need to be defined. Identify the organizations involved. What organization is providing the service ? What organization is going to be the consumers of this service ?

b. What is being provided. This could range from "HPDesk Services" to "General Accounting System Services". The point here is to be precise. It is imperative that both parties understand what the provided services entail. By having a clear and precise definition of the services, the execution of the SLA during the contract duration will be much smoother.

c. The Consumers expectations defined in sufficient detail such that there is no misunderstanding in the volume, timeliness and availability of the service. It can get tricky here, but again define the SLA in terms of what is the optimum level of service for the organizations involved.

Volume defines the amount of work that the service will require. For interactive terminal based applications this is usually the number of "transactions" that will be executed. Remember to properly define what is meant by a "transaction". For batch oriented applications volume could be measured in the number of jobs submitted or scheduled. Care should be taken if your business is cyclical in nature. Take into consideration the weekly, monthly, or quarterly spikes that many systems do see. A fatal mistake would be not to identify and anticipate cyclical trends in system volume. It is important at this point to also pinpoint as accurately as possible any future changes in volume. While system capacity may be adequate for the SLA being negotiated, perhaps a new product introduction or a new store will be opening up; such that these events will cause volume to increase. Having knowledge of increased volume fits into the pro-active SLA methodology. Knowing about it allows for the parties to plan together on how increased volume is going to be resolved.

Timeliness defines and sets expectations regarding response time and batch throughput. For the interactive terminal based applications the unit of measure is response time per transaction. The caveat is to be absolutely sure that a "transaction" is defined. The "transaction" then becomes the unit of work that can be measured and reported on. The aforementioned metric response time can be one of the key indicators in your SLA. Batch throughput can be measured in overall elapsed time for the jobs defined in the SLA. The parties involved in the SLA must define the acceptable levels of response time and throughput at this point. These data points will then be used later for Management reporting to gauge how well the service is on the system defined in the SLA. Precision is required in defining both measures of timeliness. "85% of all Order Processing updates must complete with 5 seconds" or "The General Accounting month end close job must complete on the fourth day of the month by 10:00am" are examples of the precision needed when defining timeliness. At all costs avoid the terms "fast", "slow", "within an adequate time frame". Terms such as these can only lead to multiple meanings and interpretations which benefit none of the parties in the SLA.

Availability states when the service will be provided. This can be scoped in several ways. System uptime could be the component of choice. The parties involved may agree that the Personnel S/70 be up Monday thru Friday from 8:00am until 5:00pm excluding holidays. Another way to view availability is at the logical system level. Stating when the Cost Accounting database will be available or when the Order Processing Entry program is accessible work as well as the physical system choice. Factor in methods for handling weekend application processing that may be in conflict with system backup and maintenance strategies.

The primary message here is to understand the expectations of both the supplier and the consumer. With this new found understanding set forth a system of operation that meets the needs for volume, timeliness, and availability.

d. The limits of the service. All systems have logical and physical limits. Be sure that the agreed upon expectations fit into the actual limits of the operating environment. For example, a saturated S/949 will not be able to handle future growth of an application if said growth was identified to occur in the SLA life span. Any difference in system resource limitations needs to be negotiated. Allow room for the unexpected as well, more often than not a "new" system pops up on our machines that we had not expected. If limits are a constraining factor than ways to resolve this issue need to be agreed upon immediately. Understand and document the ways in which the system limitation area will be answered. Sharing of resources, budget allocation transfers or out sourcing are alternatives to the problems of system hardware limitations.

e. How is the service to be measured and reported? Once expectations have been set, a reporting mechanism needs to be instituted to inform the parties involved how the "system" is operating in terms of the SLA. Monitoring of the system is essential. Proper monitoring will allow for any problems or variances in the SLA to be identified and resolved. Also, future SLA negotiations can utilize the data collected in the reporting mechanism in capacity planning scenarios. The tools available to administer SLAs fall into four categories: Diagnostic, System Management, Application Tuning and Capacity Planning. Diagnostic tools are the most readily available and used tools. These are the tools used in real time modes to answer the question "what is happening now on my machine ?". System Management tools provide the data repository that will allow for the Management reports required to validate the SLA during the contract's life span. Application Tuning tools can be utilized when a particular program or series of programs are not meeting the expectations set in the SLA with regard to "timeliness". An example here is a tele-marketing program that is taking 35 seconds to paint a screen, when the contract stipulates 10 seconds. The Application Tuning program can be used by the programming staff to tune the offending program such that the SLA criteria can be meet. The final category of tool is the product geared to providing Capacity Planning. Here the SLA volume estimates along with the provided future growth numbers can be used to project out future capacity scenarios. All four types of tools can be used to administer SLAs. The budget and technical ability of the organization will determine what tools are needed and purchased.

f. How long is the SLA? The duration of the SLA needs to be stated such that proper planning for re-negotiation can take place.

What are the benefits of a Service Level Agreement?

The major benefit is that expectations of the supplier and the consumer are set. Both sides are in agreement on the service and the associated criteria defined in using the service and providing it. This is the main tangible benefit of using SLAs.

However, the intangible benefits provide much to the parties as well. The transition from a reactionary firefighting methodology of performance management to one of a pro-active nature will be apparent if the SLA is followed and supported. Just think how you will feel if all those "system surprises" have been eliminated allowing you to actually THINK about the future. The SLA methodology provides a framework for organizational cooperation. The days of frantically running around juggling batch schedules and moving applications from machine to machine are eliminated if the SLA has been properly defined and adhered to.

Another intangible is that capacity planning becomes a normal, scheduled event. The SLA provides a vehicle in which volume of services are defined, and if done properly future volume requirements are communicated to the proper parties. This allows for the Performance Analyst to properly and regularly perform capacity planning reports. Regular capacity planning reports will save money in the long run since the output of the capacity plan will be factored into future SLAs over time. This allows for the planned increases in volume to to be used in the projection of future hardware purchases. Having advance knowledge in possible capital expenditures allows for proper and realistic budgeting.

How does a SLA fit into my HP Environment?

SLAs are common place in the mainframe mission critical Information System environments (i.e. IBM). Unfortunately, this has not been the case in the minicomputer arena. The reason for this is that the mainframe oriented shops have larger staffs and usually larger budgets. Included in these larger environments is typically a Performance Analysis/Capacity Planning section which has the charter for insuring that the mainframe doesn't run out of steam. These machines cost millions and millions of dollars and a staff of a half dozen to a dozen JUST for Performance Analysis/Capacity Planning is not out of the norm.

Compare that scenario with that of the HP environment. The HP machines do not have System Programmers, CICS Programmers, or JCL Specialists. We tend to be multi-tasking System Managers or Application Programmers. The machines are just plain easier to use. As such, a Performance Specialist is a rare site in most HP shops. This function is dovetailed in with the normal duties of the System Manager. Yes, some shops do have dedicated Performance Analysts but these are the exceptions rather than the rule.

The challenge here is to recognize the differences between a mainframe site and the more traditional mini-computer site. While the distinctions here are getting fuzzier by the moment the benefits that the mainframe shop receives from SLAs warrant our attention. HP shops can benefit from SLAs as well. The tools, resources, and services are available. There no longer is an excuse TO NOT do performance management. The choice each individual shop needs to make is to decide upon the level of detail that is required. You best understand the needs of your users. In using SLAs, keep these site attributes in mind. Define, negotiate, and administer your SLAs in a fashion that best fits your environment with consideration given to the potential benefits.

What have we learned ? Where should we go from here ?

The Service Level Agreement is a contract between the suppliers and consumers of system services. It is a methodology that provides the parties involved with a vehicle to define, manage, and administer performance management of computer systems. Benefits accrue to all involved in the SLA due to the openness and the structured nature of the SLA. Surprises are eliminated, future growth trends are identified, capital expenditures can be planned pro-actively. By definition, the SLA makes the parties involved in the

SLA think and act on performance management issues before performance becomes a problem.

Use Service Level Agreements. They will change performance management from a reactive activity to a pro-active activity allowing the parties involved to focus on planning and preparing for the future as opposed to continually fighting today's performance fires.

Object Oriented COBOL
A New Initiative From HP

Megan Adams and Dmitry Lenkov

Hewlett-Packard Company
California Language Laboratory
19447 Pruneridge Avenue, MS: 47LE
Cupertino, CA 95014

Abstract

Object Oriented COBOL is the next generation of COBOL. Originating at HP, Object Oriented COBOL addresses the classic software engineering problems of the growing complexity of large systems, the need to reuse and maintain existing code, the need to improve software productivity, and the desirability of incremental development of complex software. In addition, Object Oriented COBOL is tailored to the particular requirements of the COBOL user community. This includes being able to handle complex systems consisting of old COBOL code. Object Oriented COBOL provides support for embedding old COBOL programs within the Object Oriented paradigm. Object Oriented COBOL also provides COBOL programmers with an easy transition to a new technology (while maintaining complete compatibility), and includes features which improve maintainability of COBOL systems in the face of changing requirements.

In this paper, we outline the design goals of Object Oriented COBOL, provide an overview of the proposed feature set, and describe how Object Oriented COBOL will help the COBOL programmer to solve problems. Finally, we will report on the progress of HP's efforts to rapidly standardize this feature set, in the ANSI and CODASYL standards committees.

1. Introduction

COBOL is still by far the most heavily used programming language. It has been recently estimated that approximately seventy billion lines of COBOL code are currently in use. Over two million COBOL programmers continue to maintain this existing COBOL code, as well as create new and more complex systems in COBOL. This large community faces a number of problems which are common to those faced by the software development community as a whole.

1.1 Problems Faced by the COBOL community

The growing complexity of software systems is the single most important problem faced by the software community. While many parts of the software industry have to cope with this problem, the COBOL community might be the one most seriously affected by it. In addition to the usual problems, such as modeling complex systems, portability, maintenance, and software reuse, COBOL programmers have to deal with much larger and much older programs that continue to work and cannot be thrown away and rewritten. What are the solutions to these problems?

1.2 Possible Solutions

Four main alternatives have been widely proposed as solutions to problems faced by the COBOL community.

1.2.1 Translation to a Better Language

Translation of existing COBOL to a better language and then doing new development in this language has been proposed many times. This is clearly not realistic. It would require an immense investment in retraining over two million COBOL programmers. Also, a translation process would have to be developed that would produce readable code in the new language--this alone is an exceedingly difficult problem. Finally, translation of existing COBOL code to the new language would be a huge task.

1.2.2 Interface with a Better Language

This approach is more realistic. An interface to an object oriented language (for example, C++) would allow projects to use their existing base of COBOL code while developing new code in an object oriented way. However, it would either require programmers to use two very different languages, or managers would have to coordinate the work of two groups of programmers using different languages, as well as coordinate the process of code integration. There are no well-known precedents in the industry that would provide a basis for confidence in this approach.

1.2.3 Fourth Generation Languages (4GLs)

It has often been claimed that 4GLs greatly improve productivity. They do allow a single user to build small systems in a matter of hours or days instead of weeks, months, or years. However, some attempts to build high-volume, on-line systems in 4GLs have resulted in disasters. In addition, these languages are proprietary and do not provide the portability across platforms that standard COBOL provides. Furthermore, one of the major drawbacks of 4GLs is that they change the way we do programming, but not the way we program. The major software methodology design problems are not addressed.

1.2.4 Object Oriented Extensions to COBOL

Object oriented programming is widely considered to be the most promising technique for successfully handling the growing complexity of software systems, as well as problems associated with this complexity, such as increased need for ease of maintenance and software reuse. Object oriented programming does not require designing a new language from scratch. Extension of an existing language with a set of features addressing the new needs of the programming community has proved to be successful in many cases. Good examples include C++, which was based on the C language, and CLOS (Common Lisp Object System), which was built on top of Common Lisp. There is significant expertise in the software industry in adopting such extended languages. One of the main attractions of this approach is that a relatively small amount of work is required in order to reuse existing code within applications using the extensions, or to translate existing code into the extended language. Object Oriented COBOL can include object oriented features that have been proven in practice to help in solving software problems, while remaining 100% compatible with existing COBOL. This is definitely the most attractive alternative for addressing the needs of the COBOL community.

1.3 HP Initiative

HP delivered a proposal and working paper on Object Oriented COBOL to the CODASYL COBOL Committee in June of 1989. In response to the HP initiative and to growing interest among COBOL vendors and in the COBOL user community, CODASYL sponsored a symposium on Object Oriented COBOL in November of 1989. At this meeting the Object Oriented COBOL taskforce was formed, with the charter of drafting a specification for Object Oriented COBOL.

1.4 Paper Outline

This paper will describe basic concepts introduced in Object Oriented COBOL, outline its major features, and conclude with a progress report on the Object Oriented COBOL Taskgroup.

2. Design Goals

Object oriented languages improve software modeling capability and reusability. They represent an advance in design technology which offers significant support to all phases of the software lifecycle.

But the feature set of a new object oriented language must be carefully designed if it is to realize the potential of the technology. Here, we define design goals which are intended to insure that Object Oriented COBOL will fulfill its potential. These goals fall into two sets. The first set consists of goals common to all object oriented languages. This set is motivated by practice and theory in the object oriented design community. The second set consists of goals which are specific to COBOL and reflects the priorities of the COBOL user community.

2.1 Goals common to all object oriented programming languages

1. Ease of modeling of complex real world systems.

   Object Oriented COBOL should have the capability to define object classes which describe the operation of a part of the real world, providing the capability of tying real life problems to structured collections of user defined data types.

2. Modularity and ease of maintenance.

   The object class definitions of Object Oriented COBOL should allow a change in specification to have localized effect. That is, changes in specification need not impact system architecture.

3. Software reuse and productivity.

   Reusability of software components is potentially the most important single factor in increasing software productivity. Thus the the object class definitions of Object Oriented COBOL should permit and encourage reuse of software components.

4. Incremental development.

   The object class definitions should be easily extended and evolved.

2.2 Goals specific to Object Oriented COBOL

1. Object Oriented COBOL should be easy to learn. Ease of learning requires that we keep a correspondence between the new features and features in the existing language.

   In general, our goal is have the new feature set consistent in syntax and notation with the existing feature set.

2. Object Oriented COBOL should be entirely compatible with existing COBOL. That is, existing COBOL programs should be able to be compiled and executed in an Object Oriented COBOL environment.

   The general goal is to provide extensions rather than to modify existing COBOL syntax or semantics.

3. Object Oriented COBOL should use new syntax when new functionality is introduced. The new syntax should be consistent in style and notation with existing syntax.

   The general goal is to maximize the power and flexibility of the new features, while avoiding problems which arise when existing syntax and semantics are redefined.

4. The new features of Object Oriented COBOL should be simple enough to allow an average COBOL programmer to understand and be able to use them in a reasonably short period of time.

3. Object Oriented COBOL Concepts

The notion of an object is fundamental to Object Oriented Programming. Objects represent abstractions that are meaningful to users and user programs. An object is more than just information (or data). It has operations (or methods) associated with it. An object responds to a request by executing one of these associated operations, the one which corresponds to the request. Objects can be grouped into classes; this facilitates the design of systems which derive their structure from data characteristics, rather than from procedural logic.

Here we provide definitions and examples of the main concepts used in Object Oriented COBOL. These definitions form the conceptual groundwork for the Object Oriented COBOL feature set presented later, and thus it is important to understand these first. We introduce the concepts of object and class, services and requests, views, inheritance, and polymorphism.

Object Instance

An object instance is a computational entity whose behavior (the ability to access or modify the information associated with it) is characterized by a set of services that it can perform in response to requests from clients (the people or programs making the requests). The services are described without dependencies on the particular data structure or data format used to represent the information or the algorithms used to implement the operations. In particular, there could be several possible implementations of the same behavior. In object oriented programming, services are often called operations or methods.

Three primary uses of object instances can be identified. Some object instances model things in the domain of a COBOL application, such as cities and roads in a program for making maps. Other object instances provide an interface to resources in a computer system, such as modems in a computer network. And others model mathematical abstractions, such as stacks and arrays, which are used in programming.

Example:

A simple example of an object instance is a stack whose behavior is characterized by the operations push, pop, and top. These operations are specified and used without reference to the underlying implementation, which might be an array or a linked list. The code of the operations push, pop, and top would access the data through private operations, such as array_index or list_head, that are not part of the abstract stack behavior.

Rationale:

The notion of object instance is fundamental to object oriented systems. Object instances provide a framework for organizing the information in a system in a way that presents a natural and familiar interface to clients.

The term object instance was chosen over object because the term object is already used (with a different meaning) in sections of the JOD and the ANSI Standard which deal with uniqueness of reference.

Object Instance Handle

An object instance handle provides a way of unambiguously identifying an object instance. An object instance handle is a unique identifier for the object. An object instance handle might take the form of a name or a location.

Example:

In C++, a pointer value serves as an object instance handle. In the HP NewWave environment, a link from a document object to a chart object is an object instance handle. In the Unix file system, a file name is not a handle, because it may refer to different files over time if a directory is renamed.

Rationale:

An object instance handle is necessary to issue requests to an object reliably. It insures uniqueness of the object referred to, which prevents ambiguity.

Encapsulation

Encapsulation is achieved when implementation (code and data) is hidden behind an interface, where the interface is the "boundary" of encapsulation. Changes of implementation that do not change the interface do not then affect users of that interface. Encapsulation can also be understood as enforced data abstraction.

Example:

An object instance is encapsulated if it can be accessed by clients only by issuing requests for the object instance to perform its provided services. In other object oriented languages this means that the primitive operations used to implement the abstract operations are lexically hidden from clients of the object. The linked list operations are hidden from clients of a stack that is implemented as a linked list.

Rationale:

Encapsulation is especially important for the maintainability of software and the extensibility of systems. If data abstractions are used for organizing a system, but those abstractions are broken by bypassing their associated operations, then details of the implementation are exposed to outside components. This allows errors to ripple through the system, as will as making the system more difficult to understand and change.

Embedded Programs

An existing COBOL program which functions as part of an Object Oriented COBOL application is called an embedded program. It is an object created by wrapping an existing COBOL program with an appropriate object class interface.

Example:

Taking a COBOL payroll system and making it conform to the standards of an object oriented application.

Rationale:

COBOL has a huge existing base of software. Object Oriented COBOL, in order to satisfy the COBOL community, must be able to easily embed these working programs within Object Oriented COBOL systems. The ability to embed old COBOL programs greatly increases the reusability of these old programs.

Request

A request is a statement that specifies a service to be carried out by object instances. A request specifies a service and an object instance which is to provide the service. (This object instance is the provider of the service; it is also called the target object instance.) A request may take arguments and produce results.

Example:

A print request can be made to any printable object, e.g., a document or spreadsheet.

Rationale:

Requests cause the services of an object oriented system to be performed.

Generic Request

A generic request is a request that may be issued to different objects that provide (similar) services with different implementations and possibly different behaviors. The generic request itself does not determine how the services will be performed. When a generic request is issued, a selection process determines the actual code to be executed to perform the service.

Example:

If every printable object instance has the same view (or interface) for its print functionality, then a program which uses this view can issue generic print requests for all printable objects. The request may also specify a device object where the document will be printed.

Rationale:

Generic requests are a major factor in the reusability of object oriented programs. Code written in terms of generic requests can be used for different purposes when the requests are sent to objects that interpret them differently. In object oriented user interfaces, generic requests allow multiple applications to share a common interaction style, improving ease of use.

Dynamic Binding

Definition:

Binding is the selection of a method to provide a service. Binding is dynamic when the selection of a method to provide a service is made at the time the service is actually requested–that is, at application runtime. Dynamic binding is often associated with generic requests.

Rationale:

By not requiring that the actual code implementing a service be known in advance of the issuing of the request, client code can be written with a much broader range of potential (re)use. Dynamic binding makes a system easier to change and grow incrementally, because new objects can use old objects and vice versa, without having to modify existing objects. Dynamic binding makes systems easier to enhance and evolve.

Static Binding

Definition:

Binding is the selection of a method to provide a service. Binding is static when the selection of a method to provide a service is made before the service is actually requested. Static binding is often associated with (non-generic) requests.

Rationale:

Static binding allows compiler optimizations, and also facilitates early error detection. Also, because the actual code to be executed is known before runtime, static binding can result in systems whose behavior is easier to predict and understand.

Object Class

An object class defines the implementation of a data abstraction, including the format of the data associated with an object instance and how the services provided by the object class manipulate that data. Different object instances, by being of the same object class, share a common implementation. Object instances of related object classes can also share implementation, if the object classes that define them have an inheritance relationship. (Inheritance is a mechanism for defining an object class in terms of other object classes; see Class Inheritance.)

In general, object classes use instance variables to define the representation of state information, and methods to implement requests for services. Although the executable code is shared among instances, each object instance typically has its own internal state. Some object oriented programming languages (for example, Smalltalk) also define class variables, where the class variables are shared state for all objects of the class. The creation of new object instances is called instantiation. Some object oriented languages also define special methods within object classes for creating new object instances.

Example:

The class of a circle object might define the state of a circle object as represented by two instance variables, CENTER and RADIUS, and define two methods (EXPAND and MOVE-OBJ) implementing the expand and move services:

```
* procedure division code in EXPAND:
 COMPUTE RADIUS = RADIUS * FACTOR.
* procedure division code in MOVE-OBJ:
 COMPUTE CENTER = CENTER + DISPLACEMENT.
```

Rationale:

An object class defines data along with operations upon that data. Thus object classes facilitate the design of systems which derive their structure from data characteristics, rather than from procedural logic.

Object classes are the code and data descriptions of an object oriented system. They benefit a developer by allowing related classes to share implementation; that is, instances can be created without having to redefine each object from scratch. Inheritance is defined on classes.

Class Inheritance

Class inheritance (called inheritance here for convenience) is a mechanism for creating classes incrementally: one class can be defined in terms of other classes. The new class extends the existing ones by adding data to the object representation (data format), adding new operations, and possibly changing the definition of existing operations. Single inheritance only permits a class to be defined in terms of a single existing class. Multiple inheritance allows more than one class to be used in defining a new one. Inheritance often takes the form of a class of objects inheriting from other classes.

Example:

A class TITLED-WINDOW could be defined to inherit from the class WINDOW. The TITLED-WINDOW class would add a definition for a TITLE instance variable and procedures to implement the services RETURN-TITLE (to return the title) and SET-TITLE (to change the title).

Rationale:

Inheritance is a major factor in the reusability of object oriented programs. It provides a systematic way to extend even large and complex classes to satisfy new requirements. New classes inheriting from older classes allows reuse (and modification) of existing code without impacting clients of older classes. Inheritance gives the effect of copying and editing the textual definition of an object or class of objects to produce a new such definition, except that changes to the old definition will propagate to the new definition. A set of related classes can be organized in an inheritance heirarchy.

View

A view is a set of requests that may be issued to an object. An object satisfies a view if it provides services for all of the requests defined in the view. A view may also include information about the arguments and results of each request. The set of all the requests that an object handles constitutes one possible and most complete view of the object. Any other view this object satisfies is contained in this view.

Example:

Object Oriented COBOL provides a mechanism for declaring objects to have a USAGE which is a view. A view of a stack object might consist of the operations push, pop, and top. The stack might be implemented differently in different object classes, as, for example, a list or a dynamic array.

Rationale:

Views are important both as a description of how an object is used and as a way of checking for certain erroneous uses of objects. These errors can often be detected at compile time, which leads to improved software quality and productivity. Only objects that satisfy a particular view are valid providers for its requests. Object oriented code is written by assuming that various objects handle corresponding sets of requests, i.e., in terms of views of objects. Views make it easier to reuse existing software, and to evolve existing software to fit changing specifications.

View Containment

View containment is a classification of views, and therefore of object instances, in terms of view containment relationships.

One view can contain another view--that is, one view can contain the sets of requests of another view. This is what is meant by view containment. Views form a view containment hierarchy based on containment.

Example:

The view of the stack object consists of the requests push, pop, and top. The view of the queue object consists of the requests push and pull. The queue-input view consists of just the push request. The queue-input view is contained in both the stack view and the queue view. Stack objects and queue objects both satisfy the queue-input interface. Code that operates on objects assuming they satisfy the queue-input view (i.e., issues only push requests on those objects) can be used on both stack objects and queue objects.

Rationale:

View containment allows objects to be classified in terms of the common requests they support. It organizes and structures a system, making evident the ability of a client to issue generic requests.

Polymorphism

Polymorphism is an active property when requests are generic. The generic request denotes a range of related services and indicates what kind of objects can be requested to provide these services. What services are available for a given request depends on the target object. Generic requests reference a view that the target object instance should satisfy. If a view is referenced by a generic request, then the methods of any class which satisfies that view will be available to satisfy the request. This is considered to be inclusion polymorphism, based on the view containment relationship.

There is another type of polymorphism called parametric polymorphism. Here, classes or views may themselves be parameterized. The parameters can be views or classes.

Example:

An example of inclusion polymorphism is given in the View definition (above). Here we give an example of parametric polymorphism. A STACK class may be parameterized to take a variety of classes which implement strings, complex numbers, and integers. This gives the STACK class the ability to define the operations PUSH & POP upon a variety of unrelated stack elements. A single stack can then consist of strings and complex numbers.

Rationale:

Polymorphism is an essential property of object oriented systems. Inclusion polymorphism is the simplest and most essential. Parametric polymorphism offers sophisticated capabilities, and is particularly useful for developing object class libraries.

4. Review of Object Oriented COBOL features

The Object Oriented COBOL features described here are intended to implement, in a general and powerful way, the notion of an object in COBOL.

4.1 Examples

All of the provided examples for the features described in this paper will use one example as their base. This is an office mail system which handles memos, bulletin boards, business letters, and correspondence logs. The base class is OFFICE-FORM; all other classes are derived from this class. This system provides services for creating, formatting, printing, posting and deleting forms.

4.2 Object Class Definition

Objects can be grouped in various ways. Object classes represent one of the most commonly used mechanisms for grouping objects. An object class (or simply "class") is a group of objects that share an implementation. That is, they share the format of their data (representing information associated with objects of this class), and they share the code which implements their methods (or operations associated with objects of this class). Thus all objects of a class have the same data representation and the same set of methods associated with them.

Object classes and programs are different constructs. Object classes are defined in separate files, one to a file. Programs declare which object classes they will use, but do not define classes. The goal is to allow class definitions to be separately compiled and stored in a library, so that classes need not be recompiled each time the user program is recompiled.

Syntactically, object class definitions have a structure parallel to that of COBOL program structure. (Table 1. provides a schematic layout of an object class definition.) An object class is initiated like a program, except that after the IDENTIFICATION DIVISION, OBJECT-CLASS replaces PROGRAM-ID. They contain ENVIRONMENT, DATA, and PROCEDURE divisions. An OBJECT CLASS contains some new features, however.

One difference is that the PROCEDURE DIVISION of an object class definition can contain methods, which are similiar to nested programs, as well as normal procedure division code. By default, these methods can be invoked by any user of the class; that is, by default they are public. Methods can be specified to be private, however. In this case they are not visible to users of the class; that is, they can only be invoked by another method defined within this class. Another difference between programs and object classes is that the DATA DIVISION has an additional section for data, the INSTANCE-STORAGE section.

In general, data defined in the object class is not visible to users of the class. Data defined in an object class is accessed only through methods. For efficiency, however, these methods can be inlined by the compiler, providing data access without the overhead of a procedure call. The privacy of data provides for the encapsulation of the object.

Encapsulation means keeping implementation details of a class private. The clients of the class will only know what they need to know--that is, the interfaces to class methods. For example, if a class method is retrieving information from a table based on a key, the program using the class need not know how the retrieval is done. It could be based on hashing, sequential array indexes, etc. Note that the effect of this partition is to separate the interface from the implementation. Users are not impacted by changes in implementation that don't change the interface.

It is important to stress that separating the interface from the implementation minimizes the ripple errors that occur when implementation changes. In the retrieval example, it means that the retrieval algorithm can change with no impact on the program which uses the object class. If a performance improvement is necessary (for example, changing from sequential array indexes to hashing), or data specifications change (for example, table size increases so much that the table is written to disc between accesses), then these changes can be made to the implementation, without impacting the interface--or users.

```
IDENTIFICATION DIVISION.
OBJECT-CLASS.  class-name.

ENVIRONMENT DIVISION.
CLASS-ACCESS SECTION.
* Declare what other classes this class accesses.
* Declare inheritance, if any.
* Declare views, if any.

DATA DIVISION.
* All class data is private.
FILE    SECTION.
* Declare class files.
WORKING-STORAGE SECTION.
* Declare class data items.
INSTANCE-STORAGE  SECTION.
* Declare instance data items.

PROCEDURE DIVISION.
* The normal procedure division allowed here
* which can be followed by nested programs.

IDENTIFICATION DIVISION.
METHOD-ID. method-name.
* Methods are public by default.
* Declare a class method using 'METHOD-ID'.
* Method declaration similiar to program declaration.
END METHOD method-name.

IDENTIFICATION DIVISION.
METHOD-ID. method-name IS PRIVATE.
* Methods are private if specified.
END METHOD method-name.

* More method declarations...

END OBJECT-CLASS.  class-name.
```

TABLE 1. Schematic layout of an object class definition

We provide an example layout skeleton for a bulletin board. This board may contain text and memos; only one method is shown for simplicity. Note that an object of this class will contain a table of references to objects of another class. The following example is a preliminary illustration only; object class data is discussed more later, while the class-access section is described in our discussion of inheritance.

In the example below, the CLASS-CONTROL paragraph in the CLASS-ACCESS section declares that this class will be accessing the class MEMO. Clients use this syntax to declare what classes they will be accessing.

```
IDENTIFICATION DIVISION.
OBJECT-CLASS. BULLETIN-BOARD.

ENVIRONMENT DIVISION.
CLASS-ACCESS SECTION.
CLASS-CONTROL.
SELECT MEMO ASSIGN MEMO-F.
* A CLASS-VIEW section can follow here if necessary,
* giving the interface specification for classes used.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 NUM-BBOARDS-ACTIVE   PIC S99 COMP.
INSTANCE-STORAGE SECTION.
01 HEIGHT    PIC 9999.
01 WIDTH     PIC 9999.
01 TEXT-TABLE.
   05 TEXT-LINE  OCCURS 60 TIMES PIC X(80) VALUE SPACES.
01 MEMO-TABLE.
   05 MEMO-OBJ OCCURS 100 TIMES USAGE MEMO.
01 IDX PIC S9 VALUE 1.

PROCEDURE DIVISION.

IDENTIFICATION DIVISION.
METHOD-ID. POST-MEMO.
DATA DIVISION.
LINKAGE SECTION.
01 MEMO-PARM USAGE MEMO.
PROCEDURE DIVISION USING MEMO-PARM.
BEGIN.
IF IDX > 100 THEN DISPLAY "BULLETIN BOARD FULL"
ELSE SET MEMO-PARM TO MEMO-OBJ (IDX)
ADD 1 TO IDX
END-IF.
END METHOD POST-MEMO.

* more method definitions can follow

END OBJECT-CLASS BULLETIN-BOARD.
```

TABLE 2. Example BULLETIN-BOARD Class Definition

Here is the syntax for the CLASS-CONTROL paragraph, as it appears in both the object class definition and the user program:

```
CLASS-CONTROL.
SELECT MEMO ASSIGN TO MEMO-F.
```

Here MEMO is the name of an object class and MEMO-F associates that logical class name (what appears after the IDENTIFICATION DIVISION for the object class) with the physical location of information on the class. This parallels the SELECT ... ASSIGN clause as it is used for files, where a logical file name is associated with a physical file location. (Exactly how these are associated should be implementor defined.)

4.3 Object Class Data

Data in object class definitions can be one of four kinds.

1. Instance data, defined in the INSTANCE-STORAGE SECTION of the object class.
2. Class data, defined in the WORKING-STORAGE SECTION of the object class.
3. Method data, defined in the methods of the object class.
4. Data defined in nested programs which are contained in an object class.

The first two are discussed here. The third is covered in the discussion of methods, and the fourth is handled under the topic of embedding old COBOL programs within Object Oriented COBOL.

4.3.1 Instance Data

Instance data is defined in the INSTANCE-STORAGE SECTION. These data items have the characteristic that they are unique for every instance of the class. Instance data declarations are defined in the class and so are common across all objects of the class, *but* storage is unique to the object. That is why these are called 'instance data'--they constitute the state of a particular object instance.

Instance data is private to the object instance. This means that only methods of the object class may access these state variables, so that they are protectively encapsulated by a method interface. Read and write access methods for instance data can be inlined for efficiency.

4.3.2 Class Data

Class data is defined in the WORKING-STORAGE SECTION of the object class. These data items have the characteristic that their storage is shared by all objects of an object class. For example, in the BULLETIN-BOARD class defined earlier a class data item was defined:

```
01 NUM-BBOARDS-ACTIVE    PIC S99 COMP VALUE 0.
```

Note that a class data item can be initialized.

Initialization and lifetime of class data differs from that of the instance data. First of all, class data are not dynamically created and destroyed, even if the instances of the class are. In fact, class data exist before any objects of this class are created, and they are destroyed only when the program terminates, not when the object is deleted. Second, they cannot be referenced in the initializer code of a class. This would cause repeated initialization. They can only be initialized with a VALUE clause, which implies that a class data item cannot itself be an object instance.

4.3.3 Files as Class Data

Files defined in object class definitions are declared in the FILE SECTION, like file declarations in programs. However, note that files (and their associated file records) are class data, not instance data. This is because objects of a class which has FILE-A as an attribute will all be accessing the same FILE-A, not a unique FILE-A for each object.

Class data files, like class data and instance data, are private. This means that programs using objects of this class cannot do I/O on the file; they must invoke methods of the class to do the I/O. File class data will be protectively encapsulated as a rule. Reading and writing a file which is part of the state of an object from outside the class will violate the encapsulation provided by the class definition.

4.4 Object Class Methods

Method attributes are routines defined in a class definition. The method code (or implementation) is shared for all instances of a class. For example, the implementation of method POST-MEMO (declared in class BULLETIN-BOARD) is shared by all object instances of this class.

Note that this method has the normal divisions and sections of a COBOL program. This brings up the general point that, in general, methods should be assumed to be defined and to behave like programs, with the following differences:

1. Methods always have the INITIAL attribute.
2. Methods always have the COMMON attribute.
3. Methods are introduced with a METHOD-ID in the IDENTIFICATION DIVISION.
4. Methods are terminated with END METHOD.
5. Recursion is not prohibited for methods.
6. Methods can access any data in the INSTANCE-STORAGE section or the WORKING-STORAGE section of the object class in which they are defined.
7. Methods can call nested programs defined within the object class only if those programs have the INITIAL attribute.
8. They cannot have EXTERNAL data.

4.4.1 GET And SET Methods

Methods whose only purpose is to read or write data defined in the object class definition have a special definition syntax. These methods have no visible COBOL code associated with them; their function is indicated by the key words GET or SET, followed by a data item name (which must be at the 01 level). These methods are the only means a user of an object class has of accessing data declared within the object class definition. In some cases, invoking these methods will result in the code executing the operation being placed inline, resulting in the most efficient object class data access. Below are GET and SET methods for the instance data item HEIGHT (which was declared in object class BULLETIN-BOARD):

```
IDENTIFICATION DIVISION.
METHOD-ID. GET HEIGHT USING PARM.
END METHOD GET HEIGHT.

IDENTIFICATION DIVISION.
METHOD-ID. SET HEIGHT USING PARM.
END METHOD SET HEIGHT.
```

Here PARM is assumed to have the same record description as HEIGHT.

4.4.2 Initializer Code

Object instances can be initialized by initializer code. This initializer code occurs immediately after the PROCEDURE DIVISION USING clause in the object class definition. That is, it is not contained by any method. This code corresponds to routines, called 'constructors' in some object oriented languages, which initialize object class data.

Initializer code performs simple value initialization. That is, the initial values of object class data can be set by initializer code. Initializer code is not allowed to instantiate object instances.

4.5 Declaring and Allocating Object Instances

4.5.1 Declaring Object Instances

All objects in Object Oriented COBOL are accessed only through object instance handles.

Here is the declaration of an object instance handle:

```
01 BBOARD USAGE BULLETIN-BOARD.
```

An object instance handle is always initialized to a NIL pointer value.

Object instance handles are declared by giving them a USAGE of a class or a view. If the USAGE is of a class, the class must reference a class made accessible to this user in the CLASS-CONTROL paragraph. Object instance handles can also be given a USAGE which is a view. This will be discussed in the section on views.

Object instance handles cannot be redefined, and they cannot be defined in a table with a variable number of occurrences. They can be GLOBAL. Object instance handles must either be declared on the 01 level, or if they are elementary data items in a group item, all elementary items in the group must be object instance handles. In the latter case, only the individual handles can be referenced; the group cannot be referenced in any way.

Note that declaring an object instance handle does not in itself cause an object to be allocated and initialized.

4.5.2 Object Instance Allocation

Object instances are allocated (and deallocated) at runtime. From the point of view of memory allocation, all object instances are dynamic, and their memory is allocated in a dynamic memory area.

Object instances are created at runtime using the SET statement. The keyword NEW triggers allocation of memory. For example:

```
SET BBOARD TO NEW BULLETIN-BOARD.
```

4.5.3 Object Instance Deallocation

An object's lifetime begins when it is dynamically allocated via the SET. Its lifetime can terminate in one of two ways.

The first way it can terminate is through an explicit DELETE. For example:

```
DELETE BBOARD.
```

This has the effect of returning allocated memory--for BBOARD only. If the object BBOARD contains handles which reference other object instances (for example, MEMO-OBJ), the DELETE will not return memory allocated for these other objects. That must be done explicitly. After the DELETE is executed, BBOARD can then be initialized with another SET to reference a new object instance.

The second way the lifetime of an object can terminate is through automatic memory management (garbage collection).

4.6 Referencing Object Instance Data & Methods

4.6.1 Referencing Object Instance Data

To reference object data from outside the object class, a GET or SET method must have been defined in the object class definition. GET or SET method invocation looks similiar to regular method invocation; however, GET or SET methods may be inlined, providing efficient access. (The compiler determines when to inline.) What is required is one of the keywords GET or SET, the name of the object data, an object instance handle name, and a parameter. For example, we defined a GET method for HEIGHT (which is an object instance data item in BULLETIN-BOARD), so we are allowed to look at this item:

```
GET HEIGHT OF BBOARD USING CURR-HEIGHT.
DISPLAY CURR-HEIGHT.
```

4.6.2 Referencing Object Methods

Methods are invoked, but the invocation looks much like a GET or SET, but without those keywords. That is, there is some uniformity between object data reference and object method reference. If the method has parameters, they are supplied with the USING clause.

```
POST-MEMO OF BBOARD USING CURR-MEMO.
```

4.6.3 Referencing Class Data

A reference to object instance data must be qualified by an object instance handle. As there is only one copy of class data for all object instances of the class, qualification by object instance handle is not necessary for class data. Otherwise a reference to class data is the same as a reference to instance data. A class data reference can be qualified with the class name, as below:

```
GET NUM-BBOARDS-ACTIVE OF BULLETIN-BOARD USING CURR-BBOARDS.
DISPLAY CURR-BBOARDS.
```

4.7 Inheritance

Inheritance is the mechanism whereby one class inherits implementation from another. In this case the descendant is called the derived class or subclass, and the ancestor is called the base class or parent class. In COBOL, a derived class inherits all the attributes from a base class by listing the base class in the INHERIT clause of the CLASS-CONTROL paragraph.

Inheritance provides the capability to easily re-use and extend classes. Inheritance is a mechanism for incremental definition, which means that derived classes can add new services to those that the base class provides, as software requirements evolve over time. Inheritance is inclusive; that is, a derived class can define additional services but cannot delete any inherited services.

In the derived class, inherited public methods are still public, and inherited private methods are still private. GET and SET methods, if defined in the parent class, are also defined for the derived class. The derived class can access the private data and methods of its base class, just as though these were its own. In particular, the derived class inherits its own copies of the parent class private object instance data, and it inherits access to its parents private methods.

Inherited methods may be redefined in a derived class, with some restrictions. The method interface must remain the same as the interface in the parent class, although the method body (or implementation) can be different in the derived class.

Inheritance is defined in the CLASS-ACCESS SECTION of the class definition.

Here is the syntax for the general case:

```
IDENTIFICATION DIVISION.
OBJECT-CLASS. derived-class-name.

ENVIRONMENT DIVISION.
CLASS-ACCESS SECTION.

CLASS-CONTROL.
INHERIT base-class ASSIGN base-class-info.
SELECT  unrelated-class ASSIGN  unrelated-class-info.
```

For example, say we want to add a new object class to our office mail system. This class will be like a BULLETIN-BOARD, but will have a log of its contents. We will call the object class CORRESPONDENCE-LOG. It will inherit from class BULLETIN-BOARD, and add one instance data item: TABLE-OF-CONTENTS. Here is the inheritance specification:

```
IDENTIFICATION DIVISION.
OBJECT-CLASS. CORRESPONDENCE-LOG.

ENVIRONMENT DIVISION.
CLASS-ACCESS SECTION.

CLASS-CONTROL.
INHERIT BULLETIN-BOARD ASSIGN BBOARD-F.
```

The CLASS-ACCESS section above is the same as the one we saw in the section on the object class definition. What is added is the ability to specify inheritance in the CLASS-CONTROL paragraph. The INHERIT ... ASSIGN clause identifies the base class name, the file in which it is contained, and the attributes of the base class that will be redefined.

4.7.1 Declaring Object Instances with a USAGE of a Derived Class

It was noted earlier that object instances can be declared to be of USAGE of a class or view. In the case of the above example, this means we could declare a handle to an object instance in a client of CORRESPONDENCE-LOG as follows:

```
01 NLS-CORRESPONDENCE  USAGE CORRESPONDENCE-LOG.
```

This object instance will have its own copies of all instance data of BULLETIN-BOARD and CORRESPONDENCE-LOG, it will share access to the class data of these classes, and it will provide the services of these classes (including any GET & SET methods defined for the data). Object instance handles defined with a USAGE of a class are not allowed to contain references to any derived classes of the class specified in the USAGE. (This is allowed in many object oriented languages; we offer a more general version of this functionality with views.) When object class methods are invoked with an instance handle of a USAGE which is a class, the method and the invocation are always statically bound.

4.8 Views

An object class is defined to be the implementation specification for object instances of that class. The object class interface is all the services the object class provides to the outside world--in particular, the public methods of the object class, and the data for which GET and SET methods are defined.

The object class interface is the default view of the class. That is, by default the outside world views object instances of a particular class according to what is visible in the object class interface. We are presenting a specification which includes the ability to have multiple views of a class.

First we need to define a view more exactly. As mentioned above, the object class interface is the default view of the object class. The services provided by the object class define this interface. Note that the code of the methods which implement the services is not a part of the interface--rather, that is part of the object class implementation. Another way to understand what constitutes an object class interface is that it includes what a user needs to know to use the object class, and no more.

Thus, to define an alternate view of an object class, we need simply to specify an alternate interface. These views, or alternate interfaces, will be a subset of the default view of the object class. Before specifying the syntax for this, we will look at how this can be useful.

Just as relational databases provide differing views of their data, it can be appropriate for different applications to have different views of the attributes of an object class. For example, let us consider a payroll program which references objects of a personnel object class. Much of the data in the personnel object class is strictly confidential, and there is no reason to allow the payroll program access to this data. Thus, the payroll program's view of the personnel object class excludes the data which is not appropriate for the payroll program to access.

The utility of views is, however, much broader than indicated in the example. An object class definition provides both the interface and the implementation, while the view provides only the interface. Recall that an object instance can be declared to have a USAGE of a class or view. As a view is only an interface, more than one object class can satisfy the view, if the class contains the services listed in the view. As we will see, views provide for generic requests and polymorphism in Object Oriented COBOL.

Here is the syntax for views:

```
CLASS-ACCESS SECTION.
CLASS-CONTROL.
SELECT class-name ASSIGN class-name-f.

CLASS-VIEW.
VIEW. view-name.
method-list.
```

Method-list includes GET and SET methods, as well as regular methods. For regular methods, the interface description is as follows:

```
METHOD-ID. method-name USING parm-1 ... parm-n.
01 parm-1      any-parm-description-like-picx-etc.
 ...
01 parm-n      any-parm-description-like-picx-etc.
```

GET and SET methods in views can only describe data at the 01 level. It was noted earlier that GET and SET methods are inlined when accessed with an object instance handle that has a USAGE which is a class, to provide for efficient data access. However, when these methods are accessed via an object instance handle that has a USAGE which is a view, these methods are never inlined. Views provide the generality of generic requests and polymorphism (which classes do not provide), but they are somewhat less efficient. For GET and SET methods, the interface description is as follows:

```
METHOD-ID. GET data-name USING parm.
01 parm    any-data-description-like-picx-etc.

METHOD-ID. SET data-name USING parm.
01 parm    any-data-description-like-picx-etc.
```

We will explore views by means of an example. Two object classes, OFFICE-FORM and LIST, provide services for handling the printing of forms. OFFICE-FORM and LIST are not related; that is, neither is a derived class of the other. They both provide services other than printing of forms, but the user does not require these other services. The user can define a view which consists of the interface for printing, and nothing else. This requires that the interface for printing of forms is the same in both classes, although the implementation of the printing service can be different. Object instances then can be declared with a USAGE of this view. The object instances can be of either object class, but they must share the same view.

Object classes and programs can declare object instances of view types.

Here is the VIEW definition for the above example:

```
CLASS-ACCESS SECTION.
CLASS-CONTROL.
  SELECT LIST ASSIGN LISTF.
  SELECT OFFICE-FORM ASSIGN OFFICEF.
CLASS-VIEW.
  VIEW. PRINTABLE-FORM.
  METHOD-ID. PRINT.
```

PRINT takes no parameters, so the USING clause and data descriptions of the parameters are not required here.

How are object instances of view types allocated and referenced? If the client wishes to print lists and office forms, an object instance handle of USAGE PRINTABLE-FORM is declared:

```
01 PRINT-FORM USAGE PRINTABLE-FORM.
```

In the client, these views are associated with a particular object class on the SET statement. For example, the following two SET statements can be executed (in any order):

```
SET PRINT-FORM TO NEW LIST.
SET PRINT-FORM TO NEW OFFICE-FORM.
```

At these SET statements the compiler will verify that the services listed in the view are indeed provided by the object class listed in the SET, and that the method interface descriptions on the view and on the class match. Note that the object instance referenced by PRINT-FORM can thus provide either the printing service of LIST or OFFICE-FORM. Which service it will provide is not known until runtime.

Assuming that these SET statements are valid, the user can then print either a list or an office form with a single statement:

```
PRINT OF PRINT-FORM.
```

Which print method is bound at this invocation is a decision postponed until runtime. This is known as dynamic binding, and this PRINT request is a generic request. The decision is made based on the object class to which PRINT-FORM was set at the most recently executed SET statement.

Object instance handles with a USAGE can reference object instances associated with any object class that satisfies the view. In the above example, we described how this worked when the classes, LIST and OFFICE-FORM, were unrelated. This principle also works with related classes. Derived classes, as discussed earlier, inherit the interface and implementation of the parent class. Because they inherit the interface, a derived class will always satisfy the interface of the parent class. That is, the derived class provides at least the same services.

For example, OFFICE-FORM has two derived classes, OFFICE-MEMO and BUSINESS-LETTER. These classes inherit the PRINT method of OFFICE-FORM. That means that the following three SET statements are all valid, as all these classes provide the printing service:

```
SET FORM-OBJ TO NEW OFFICE-FORM.
SET FORM-OBJ TO NEW OFFICE-MEMO.
SET FORM-OBJ TO NEW BUSINESS-LETTER.
```

As above, at these SET statements the compiler will verify that the services listed in the view are indeed provided by the object class listed in the SET, and that the method interface descriptions on the view and on the class match. Now the statement

```
PRINT OF PRINT-FORM.
```

can print a list, an office form, an office memo, or a business letter. This is a generic request. Views can be understood as the implementation of polymorphism in Object Oriented COBOL.

4.9 Embedding Old COBOL Programs in Object Oriented COBOL Systems

Object Oriented COBOL has been carefully designed to allow old programs to fit easily into Object Oriented COBOL systems. Old COBOL programs with the INITIAL attribute are embedded as methods in an object class. Old COBOL programs which are not INITIAL are embedded as object classes. The old programs do NOT need to be changed. All that needs to be done is to declare an interface to the old program. Here is the interface for a non-INITIAL old program.

```
IDENTIFICATION DIVISION.
OBJECT-CLASS. EMBED FORMAT-LETTER
  USING FORMAT-PARM1, FORMAT-PARM2.
01 FORMAT-PARM1    PIC 999.
01 FORMAT-PARM2    PIC XXX.
```

The interface includes the parameters to the old program, and the name of the program.

Recall that the WORKING-STORAGE section in Object Oriented COBOL is defined to have class data. An embedded program thus has only class data defined, since it can have no INSTANCE-STORAGE section. The procedure division code of the old program corresponds to initializer code in Object Oriented COBOL, while any nested programs contained within the embedded program are not visible to clients.

Using an embedded program as an object class requires that a client be able to declare an object instance handle to have a USAGE of the embedded program, and to initialize the handle via a SET ... NEW statement, as follows:

```
01 FORMAT-L     USAGE FORMAT-LETTER.
SET FORMAT-L TO NEW FORMAT-LETTER USING PARM-1 PARM-2.
```

When an embedded program is non-INITIAL, invoking the initializer code the first time will have the effect that the the embedded program is called and its data items are set to their initial values. Subsequent references to the embedded program, such as the following:

```
FORMAT-L OF FORMAT-LETTER USING PARM-1 PARM-2.
```

will execute the embedded program but not re-initialize data. The compiler must generate some code associated with the interface to the embedded program, but this is not visible to the client.

Old programs which have the INITIAL attribute are embedded as methods. If FORMAT-LETTER were INITIAL, it could be included in the BUSINESS-LETTER class as just another method of BUSINESS-LETTER:

```
IDENTIFICATION DIVISION.
OBJECT-CLASS. BUSINESS-LETTER.

...
IDENTIFICATION DIVISION.
METHOD-ID. EMBED FORMAT-LETTER
  USING FORMAT-PARM1, FORMAT-PARM2.
01 FORMAT-PARM1    PIC 999.
01 FORMAT-PARM2    PIC XXX.
```

When an embedded program is INITIAL, invoking the initializer code will always re-initialize data items to their initial values.

4.10 Other features

Garbage Collection

Garbage collection is a technique of storage management whereby objects which are determined to be no longer needed are automatically destroyed. The user does not need to manually deallocate object memory. Memory compaction can also occur, to prevent excessive fragmentation. There are a variety of ways to specify garbage collection--for individual object instances, for classes of objects, or for the compilation unit as a whole.

Garbage collection for Object Oriented COBOL will be specified on the declaration of the object instance handle:

```
01  MEMO  USAGE OFFICE-MEMO IS COLLECTABLE.
```

Such a declaration means that when the garbage collector determines that this object is no longer referenced, then the memory can be automatically returned to the system. Manual memory management (with DELETE) is not needed.

Garbage collection results in safer data structures, and therefore results in faster development of new code. Modern algorithms have reduced the overhead to a few percent.

Exception Handling

Exception handling involves runtime recovery from failures. An exception handling mechanism provides a means of detecting abnormal conditions during software execution, and of recovery upon detection. In object oriented languages, there are a variety of ways exception handling can be specified. In particular, exception handling can be specified in the class definition, or in the view (interface) specification. For example, a view of a stack interface could include a specification for how to handle an attempt to pop an empty stack.

Exception handling is a great advance in enforcing reliability specifications for software.

5. Object Oriented COBOL Task Group progress report

The Object Oriented COBOL Task Group has had two meetings since its inception in January 1990. Current members and observers include COBOL compiler vendors (Realia, Micro Focus, Allinson-Ross), large computer companies (IBM, HP, Wang, Digital, Hitachi, Sun), and users (Hewitt Associates and Southwestern Bell). The Task Group has completed a Program and Scope of Work (approved by the parent committee), and has had extensive discussions of concepts and terminology. The design process is underway. The schedule for our deliverables is aggressive. Agreement on a draft feature set is expected in January 1991, a draft specification is to be available in July 1991 (detailed enough to form a basis for prototyping efforts), and an ANSI Addendum is to be ready for submittal in January 1992.

References

1. A. Snyder, W. Hill, W. Olthoff. *A Glossary of Common Object-Oriented Terminology.* Report STL-89-26, Hewlett-Packard Laboratories, Palo Alto, California, 1989.
2. A. Snyder. *The Essence of Objects.* Report STL-89-25, Hewlett-Packard Laboratories, Palo Alto, California, 1989.
3. B. Stroustrup. *The C++ Programming Language.* Addison-Wesley, Reading, Massachusetts, 1986.
4. A. Snyder *An Abstract Object Model for Object-Oriented Systems.* Report HPL-90-22, Hewlett-Packard Laboratories, Palo Alto, California, 1989.
5. Bertrand Meyer. *Object-oriented Software Construction.* Prentice Hall, London, 1988.
6. American National Standard Programming Language COBOL, ANSI X3.23-1985 ISO 1989-1985.
7. CODASYL Journal of Development, 1988. Published by Donald Nelson on behalf of CODASYL.
8. Personal communications with Mike Cannon, William Cook, Walt Hill, and Jussi Ketonen, of Hewlett-Packard Laboratories.

TITLE: Improving the Efficiency and Accuracy of Your Information Capture Through Automated Data Collection

AUTHOR: Raymond Agrusti

Eagle Consulting + Development Corp.

170 Kinnelon Road, Suite 3

Kinnelon, NJ 07405

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 3190

Transaction Logfiles – Their Uses

David B. Wiseman
Proactive Systems
4 Main Street, Suite 101
Los Altos, CA 94022
(415) 949-9100

This paper is based on research done by Alan Brewer and David Styles of Proactive Systems during 1988/89 and my thanks are due to them for their permission to use their work. To be clear about the intention of this paper, we must differentiate between System Logfiles and Image Logfiles. MPE has the facility to log certain events which may (or may not) be of interest to the reader. This facility is enabled/ disabled with the Sysdump command and captures events such as Logon/Logoff, Console dialogue and File Closes. IMAGE logging (correctly known as MPE User Logging) is a facility to capture an image (no pun intended) of each transaction put into a database without the database accessor being aware of such logging. It is a very powerful facility that is rarely used in most installations largely because very little is understood of why one might want to use such a facility. This then is the purpose of this paper. The rest of the paper will cover:

1. Setting up IMAGE logging
2. Logging cycles
3. What can be logged
4. So the machine fell over!
5. On-line backup/mirroring
6. Database diagnosis
7. The auditors visit
8. Sherlock Holmes revisited
9. Database maintenance

1. Setting up IMAGE logging.

IMAGE logging is enabled at database level using DBUTIL. The use of IMAGE logging should not entail any performance concerns as used to be the case when it was originally introduced. The steps involved in setting up IMAGE logging are fairly simple for most users. They are:

a) Give all users who will access the database LG capability with the ALTUSER command. Don't forget to do an ALTACCT first. This is much easier if you have a tool such as MPEX.

b) Log on as the database creator and get a LOGID. This is an arbitrary name and you should use something meaningful. I find with these things it helps to have a book kept with the system (or a file on Disc – use the help file subsystem to index it on-line).

   `:GETLOG logid;LOG=logfile,DISC` (device can be TAPE)

   The logfile will be built manually during the logging cycle and a warning message will be issued by MPE.

c) Set the LOGID from b) into the database. Again this is done in DBUTIL with the command SET database LOGID=logid.

d) Enable the database for logging. This is done by running DBUTIL.PUB.SYS and entering the command ENABLE database FOR LOGGING.

e) Note that the database can now only be accessed when a LOG START has been issued.

2. Logging cycles

A logging cycle is generally the period between two known consistent points in the history of the database. For most users, it will start immediately after one backup and end immediately before the next. In between, all updates to the database will be logged into the logfile. The general format of a logging cycle is as follows:

a) Build Logfile. The name is arbitrary but must be that used in the GETLOG command. It would be normal to have the last three characters of the name as numeric and then normally 001. This allows you to use the AUTO facility of IMAGE logging so that the logfile, when full, is automatically switched to a new logfile with the same name but serially numbered above the previous.

b) Issue a LOG Start command. This is associated with the logid from the GETLOG command. Where more than one database has the same logid, they will all be started together.

   ```
   :LOG logid,START
   ```

   Note that if for some reason an error occurs after this point and you need to restart, you must first :PURGE the logfile and then issue another :BUILD as the LOG START writes to the log file and so will not start again even though no records have been written to the log file by users.

c) Allow the users to access the database. From now on, all write access by your users will be logged into the log file.

d) Stop the user access. It is necessary to stop all access to the database in order for a LOG STOP to become effective. This means that all user programs must issue a DBCLOSE before logging can stop.

e) Issue a LOG stop.

```
:LOG logid,STOP
```

This will put the database into a "Stop pending state" waiting for all users to close off access to the database. As soon as this happens, logging is stopped and the log cycle is finished.

f) Back up the database and logfile. Use DBSTORE to ensure that the database is fully stored and that, in the event of a subsequent recovery, the database and logfiles are in a consistent state. Note that there is theoretically no need to store the log files if the database is correct but it does give an extra level of security as the database can be rebuilt from the previous night's STORE and the logfiles using DBRECOV.

g) Purge the logfile (or rename into another group if you wish to keep it for later analysis).

3. What can be logged

Well, by default, everything that is updated into the database is logged along with some general information such as a time stamp. The standard calls logged are as follows:

a) DBOPEN – including the LDEV, User ID, process or program name and much other useful information

b) DBCLOSE – as DBOPENs

c) DBPUT – a complete image of the record as put into the database

d) DBDELETE – a complete record of the database record deleted from the database

e) DBUPDATE – both the before image of the updated fields and the after image are stored. If the record is a master record, the key value is also logged.

f) If you use them then DBBEGIN and DBEND transactions are logged. These are very useful to ensure that complete transactions are entered into the database. Most application software suppliers (including HP) don't appear to use them as it would require much code change but there is little reason for users not to include them in their new code even if purely for the purposes of documentation.

g) Last but not least, the DBMEMO calls are logged. This is a little used intrinsic that can be used to document all changes to a database and any other useful information that may be required. Any user program can call DBMEMO and indeed, we have used it to log a copy of KSAM/MPE calls that were made to a database that we were logging.

4. So the machine fell over!

The most obvious use of these logfiles that we have collected over time is that if we have a machine failure then we can use a program called DBRECOV.PUB.SYS to replay all of the transactions from the current logfile into the restored database without needing to get help from the users. Why would we want to do this? Well, suppose that we implement a new telephone marketing department in our company – typically we would implement a Sales Order Processing system to handle all of these new orders. When a customer calls in, his order is taken by telephone and dispatched that evening from our warehouse. If the system goes down, the order could be lost and we might upset our best customers (those that order most often and are most likely to be ordering when the machine falls over). With logging, we can restore the database and replay all of the transactions automatically into the database and so ensure that all orders are processed once and once only. Some words of warning:

a) In a heavy OLTP environment, this can take some time – often 3 or 4 hours.

b) You MUST keep a copy of the latest backup on site or within reach.

c) You must keep the logfile on a recoverable device because if you lose the system, you need to be able to do a full system restore from the last dump and then restore the logfile from the minute before the crash. This means in practice, keep either tape or private volume disc.

d) If you don't use DBBEGIN/DBEND, you may have to use QUERY to get the database(s) into a consistent state. The application may only let you enter a transaction that, for example, would duplicate the 3 details already in the database from the logfile in order to get the other three that were a part of the same business transaction.

e) The logfile will not be any use as protection against an application failure as the incorrect transactions will have been posted to the logfile.

5. On-line backup/mirroring

Some of the obvious disadvantages of this recovery are the time it takes and that it does not offer any protection against a total machine failure. With the drop in hardware prices (a used 58 is only about $20,000!), we could connect our live machine to a smaller backup machine and keep a copy of the database on this machine. The logfiles can be transported onto the backup machine either using DSCOPY or a proprietary product such as BACKCHAT. This latter can update the database continuously in pseudo real time. This level of protection can be supported even on a single CPU system providing the advantages of mirrored disc without having to either dedicate many additional disc drives or suffering the performance of squeezing all of the database onto one disc drive.

Another advantage of this is that you can backup your copy database without interfering with our production workload giving a 24 hour facility. This is especially useful in applications such as stores where maintenance is carried out in the factory during non-production hours or in a 24 hour support operation such as ours.

Lastly, given a copy of the database, you can run reports on the copy database to reduce the workload on the main production machine. This can often be significantly less expensive than upgrading from, say, a Series 70 to a 950.

6. Database Diagnosis

One of the hot topics of the late 1980's has been the growth in the market for performance monitors both at system level and specifically for databases. Indeed, our company has been at the forefront of this trend and we have done much applied research into how to improve database performance. At this point in time, the problem with such tools however, has been that they give a static diagnosis of the database which will typically not take into account how the database got into its present state or how it changes with time. Two examples would be to look at capacity planning and detail chain maintenance. In the case of capacity planning, it has only been possible to change capacities when they have reached a predetermined fullness – for example, when the SAL-HIST dataset is 90% full, change the capacity such that it will only be 75% full. This totally fails to take into account whether the data content is growing at a rate of 1% per week or whether it grows when the SAL-UPDATE program is run every month. In the case of detail set messiness, again it would be useful to know whether a dataset was messy by degrees or when a particular program was run. This could lead to a change of REPACK strategy to run the dataset repack immediately after the daily update run and before the daily reports were run or alternatively to reduce the frequency of capacity changes to once a month, run after the SAL-UPDATE program. In both cases we can dramatically increase the effectiveness of our database maintenance time and increase the overall machine throughput.

How can this information be gathered? Well, of course, it all exists in the logfiles. We can measure the number of DBPUTs by program by day and then compare to the number of DBDELETEs by day. If DPUTs are fairly matched by DBDELETEs then we can look at the number relative to the number of entries in the dataset and when it becomes significant, recommend a repack for that set. Similarly, we can measure the excess of DBPUTs as compared to DBDELETEs and recommend capacity changes based on whether it is a gradual change or related to an individual process or program (or indeed user).

Another performance aid worth mentioning here is sorted chains. Sorted chains are generally held in some fear, but logfiles provide pointers to how frequently they are updated and whether the updates are made to the end of chains or in the middle with the obvious implications for database design.

7. The auditors visit

Many of you will have been brought up in the days when data processing was as closely monitored as accounting or perhaps you work for a large organization which does work for the government. In either case, or in many more, a visit from the internal (or external) auditors may be in the cards. They will want to check that what goes in comes out unaltered and that only authorized transactions can be input to the system. Unfortunately, once a transaction is put into most systems, there is no further trace of its origin. The IMAGE log file allows you to look at every transaction and see where it came from, who input it to the system, when it was input, which program was used, etc..

Obviously, this need can be prompted by an external group to Data Processing, but it can also be internally generated. For example, we once were asked who had been updating the CREDIT-LIMIT of a CUSTOMER record. We checked all of the programs which we knew could access this data-item (all of two) and couldn't see the possibility of it being done without the user's knowledge (the program was password protected and tied to a specific LDEV in the user's department, etc..) Well, we figured that the only way to be sure was to analyze the transaction logfiles every day to be sure that the data wasn't being updated by anyone else. After a year, we were able to show that all of the alterations had been done by the users themselves.

This illustrates how some questions can be answered which are vital if we are to be assured that our businesses are being well run. The key point is that if the information in the logfiles had not been available, then no audit would have been possible and we may have forever suspected our system.

8. Sherlock Holmes revisited

Another time, one of our users called up and said that every Tuesday, the stock levels of his products were coming up out of balance. It so happened that on Monday nights the end of week report was run on these stocks and it did a few updates to keep everything in line. So, of course, everybody blamed this program immediately and set about finding the bug. No luck! The program looked perfect. We ran it on a test copy of the database and it actually worked perfectly. Now we got suspicious and blamed the users! That made us feel better but didn't fix the bug, so we started to check the logfiles for Monday nights. Lo and behold, we found that another program run automatically after the end of week report, which did a tidy-up routine but failed in one essential detail – it didn't update the stock level correctly! The types of questions that can be asked of logfiles are many and various but I will list a few examples here for illustration:

How many write users have the database open concurrently?

Which dataset gets modified the most?

How many unnecessary DBUPDATEs (null updates which don't change anything) are done and which programs are worst?

What time of day do we get most modifications?

Which program changes the CREDIT-LIMIT field in the CUSTOMER set in the SALES database?

Has any CUSTOMER been given a CREDIT-LIMIT over $50,000?

Which program changes the STOCK-LEVEL to a negative quantity?

What changes has LDEV 21 made to the ACCOUNTS database?

Wouldn't it be nice if I could use the logfile as an audit trail of the work done each day?

Does the TRANSACTIONS dataset have more updates, deletes or puts done to it because I want to add a sorted path and I'm worried about the effect on performance?

If I recover from this logfile, will I lose data because of incomplete transactions?

Every time a user runs program XYZ and enters a transaction, the program hangs for 20 seconds. There's no locking conflict so what is it doing? How many DB calls is it doing?

These questions and many, many more can be answered from the information locked up in your logfiles WITHOUT YOU HAVING TO WRITE ANY PROGRAMS TO EXTRACT THE DATA.

Start logging today and get that information safely stored for when you need it!

9. Database maintenance

A short word of caution on what you can (and can't) do.

Image databases are networked and contain embedded pointers to records in chains. These pointers are held as Block number and a byte offset within the block. It follows that anything that changes the Block number (or blocking factor) is likely to be bad news for any program that relies on this information. The information in a logfile is either related to a detail set, a master set or is "other" information such as DBBEGIN or DBMEMO. This other information is for you to decode. The information relating to a master set carries the Key item value and so the relevant database record can always be located based on this information, but the information relating to details contains only the record number in the format above. The effect on you, the user, is that you must ensure that if you want to replay a logfile against a database then the database must be unchanged in terms of the record and block structure. Changes that will affect the record position include structural change (adding/changing/deleting items), repacking detail sets, changing capacities with some database tools, running DBUNLOAD/DBLOAD or DICTDBU/L and other similar changes. If you are going to use the database logfiles for updating a different database, then you must pay even greater attention to this.

Summary

In summary then, Image logging is easy to implement and provides powerful methods to assist in database recovery, information that can be used to improve performance, and a clear history of transaction information entered into the database for future audit.

EVALUATING THE BUSINESS IMPACT AND RISKS ASSOCIATED WITH A DISASTER

BY

Gary Fletcher

Hewlett Packard
24 Inverness Place East
Englewood, CO 80134

The flow of information in one form or another is a key element in most business's today. Information processing activities are not just a record keeping aspect of your business anymore. Resources gathered via the automated information processing systems are becoming the most valuable asset that a company has. A study done by the University of Texas indicates the following dependencies on computing facilities:

Degree of Dependence	Percent
Total	19.6
Heavy	65.8
Moderate	13.9
Slight	0.7

An interruption in that flow of information can be devastating to an organization. Several studies have been done to quantify the impact to an organization if such an interruption should occur. One such study published in the Mainframe Journal in Dallas Texas shows these startling statistics:

Business Considerations

- **Average firm loses 2 to 3% of total gross sales within the first 10 days after losing data processing**
- **50% of all companies that do not recover within ten business days never recover financially**
- **Critical business functions cannot continue more than 4.8 days without recovery in progress**
- **93% of firms were out of business within 5 years after a major disaster.**

MAINFRAME JOURNAL, Dallas, Texas

And yet it is hard to believe that some companies still do not have adequate business resumption plans. By this I mean a broad-based plan that addresses all aspects of an organization that is necessary to maintain an adequate level of business processing. That's partly because computers have crept gradually into companies, leaving many top executives unaware of their strategic importance. "They don't realize that the value of the information (in the computer) could be worth several times the value of their hardware, software and building," says Steven Christensen, a researcher at the University of Texas.

As a data processing professional, you probably understand the importance of business resumption planning. A project of this nature involves the efforts of many people and a considerable amount of time. In order for such a project to be successful it must have the full support of top management. But do you have the support of the upper management of your company?

You must have the commitment of upper management in order to get a project assigned that will give you the time necessary to develop the plan and a budget commitment in order to finance the planning and implementing of your recommendations. Without that commitment you may be wasting your time developing a plan that may never get implemented. This paper will help you to get that commitment from management using proven techniques.

MANAGEMENT COMMITMENT

In order to ensure the success of any business resumption planning project, upper level management must be willing to do the following:

-Understand the need
-Make a commitment
-Assign a budget
-Assign responsibility

Why is management hesitant to make that commitment?

Ostrich approach: "It can't happen here". Many managers feel that the probability of a real disaster occurring is very remote. Why accept an additional expense on the assumption that something will happen in the future.

Titanic Approach: "Unsinkable ship". Another common feeling of management, especially among the entrepreneurs, is that they have been through tough times before, and survived without extensive planning or expensive alternatives.

Why does management feel this way? Think about the type of people you are dealing with and the reasons for their success. Typically they have been trained, or have learned to:

-Think positive/success
-Emphasize accomplishment
-Take risks
-Allow no negative thinking
-Associate with positive people
-See the bottom line
-See the big picture

Here you are preaching doom and gloom. You must align yourself with their management style. Understand and be empathetic with their way of thinking, but yet you must still make them understand their vulnerability to disasters and the benefit to the company in preparing for them.

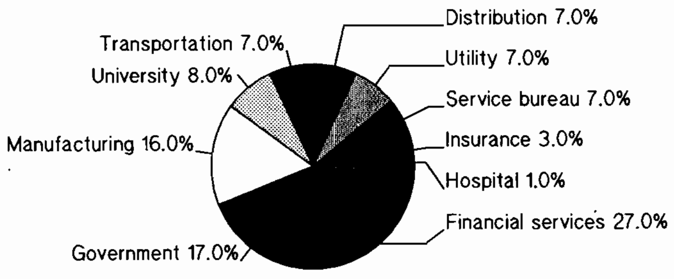
Consultant or counselor selling is the process of problem solving for your management group by ascertaining needs then presenting the value of your solution to match that need. In the disaster recovery arena, you can use risk analysis to ascertain the need and business impact analysis to show the value of disaster recovery planning.

RISK ANALYSIS

Risk analysis is the process for accessing the vulnerability that an organization has to interruptions in information flow. Frequency of the risk, the consequences and the controls that are or can be put into place to reduce or deter the risk can all be parts of the risk analysis. In selling to upper management it is important to get them to focus on the types of disasters that can and do occur in your industry or geographic area.

A study done by Contingency Planning Research, Inc. on 91 documented disasters, can help you to show the types of disasters that DO occur, and to what industries they have occurred. The following diagram illustrates the wide range of industries that experience disasters and that no particular industry can be considered to be "safe":

Who Has Disasters?

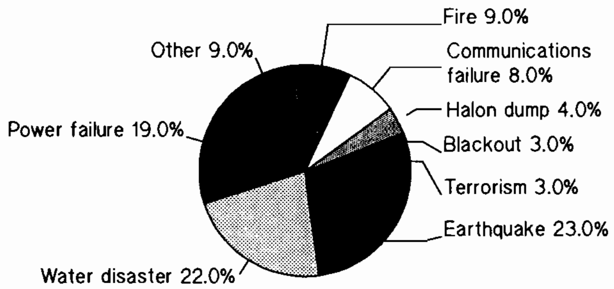


Based on 91 documented disasters.

Info. by Contingency Planning Research, Inc. (Computerworld, 7/11/88)

Of those 91 documented disasters, the following information was collected on the type of disasters that occurred. When talking to upper management concentrate on relevant, real life situations that can occur. For instance, a water main breaking above your computer center would certainly be classified as a "water disaster":

What Kinds of Disasters Happen?



Based on 91 documented disasters.

Info. by Contingency Planning Research, Inc. (Computerworld, 7/11/88)

The attitude of "it can't happen here" just isn't realistic in today's world. Perhaps you live in an area that is not subject to floods, earthquakes or hurricanes. But any business can be seriously damaged by fire, explosion, plane crash or sabotage by a disgruntle employee. Certain situations may deny you access to your facility. These may include a toxic spill or a near by chemical fire that may require evacuation of the entire area for days or weeks.

You can pick up almost any computer magazine and find an abundance of articles about companies that have recently experienced disasters and what they have (or have not) done to prepare themselves, and the ramifications of their efforts. It should not be too difficult to convince management that at some point in time a disaster of some sort will occur. But the business manager will want to know how such a disaster will impact his bottom line. In order to do this you must outline the financial and operational impacts that this event will inflict in order for management to determine the amount of resources to put into the prevention and recovery methods that you will be proposing. The tool that you can use to show the value of business resumption planning, is a business impact analysis.

BUSINESS IMPACT ANALYSIS

Your objective then is to determine the financial and operational impacts to your company that will result from a loss of information flow. This is the objective of a business impact analysis. The impact analysis will also help establish those areas of the company that are most vulnerable to disasters.

The business impact analysis will determine what business functions could be effected and if those business functions are essential to meeting the companies business objectives. This will help to classify business functions into essential and less essential, and help to determine your critical applications. Critical applications are those that must be processed in order for a company to continue its essential business functions.

There are many techniques used to gather the necessary data used in developing the business impact analysis, here are some common techniques:

-Facilitated group sessions
-Survey/Questionnaire
-Interviews
-Combination of the above

No matter what information gathering technique you choose to use, there are some key areas that the inquiry should focus on:

- System use
- Financial impacts
- Legal and Regulatory impacts
- Business relationship impacts
- Operational impacts
- Organizational interdependencies

Surveys/Questionnaires are the most common form used to gather the required information. Some tips for successful surveys are:

- Must have management support
- Can have pre-survey briefing session or at least a cover letter
- Assume worst time of year for each department
- Tailor to customers specific environment by using relevant examples or scenarios
- Keep it simple, use time frames and boxes
- Keep it relevant to business functions
- For financial information use specific dollar ranges

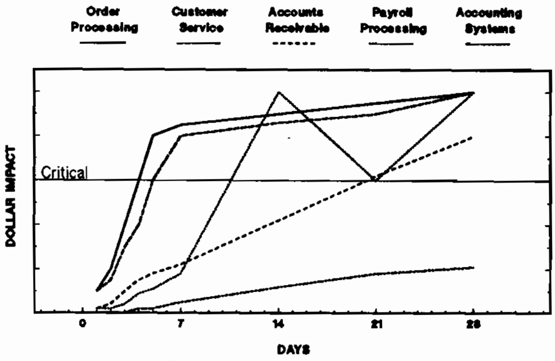
Focus your efforts on the groups within your organization that have the most to lose if a disaster should occur. Some key groups to focus on within the company may be:

- Marketing
- Finance
- Legal/Public relations
- Information Systems

The resulting financial impact will vary with each functional group within the company and if plotted will result in a loss curve over a period of time such as this:

Disaster Recovery Planning

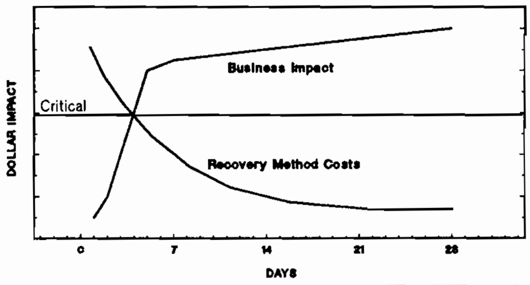
Business Function Loss Impact



This loss curve will then help to establish the recovery method that you choose to implement. Obviously, the faster the recovery time, the more expensive the recovery method. Management must decide the financial impact that they are willing to accept based on the investment they are willing to make in the recovery method. The following graph illustrates this point:

Disaster Recovery Planning

Business Impact VS Recovery Method



PRESENTING RESULTS

When presenting the results of your business impact analysis to upper management remember to keep it focused on the business impact relevant to the organization. A 20-40 million dollar loss may not be relevant to a billion dollar organization. Other suggestions for presenting the data are:

- -Don't focus on application's, talk in terms of business functions and issues
- -Organize the results by major business function or organization
- -use relevant examples and graphs

Remember to quantify the impact in business terms that relate to the companies overall business objectives. The financial impact may not be the key in managements eyes, after all, insurance can be used to cover some of the financial losses. Other key factors may be:

- -Cash flow interruption
- -Loss of customers
- -Loss of competitive edge
- -Erosion of industry image
- -Reduced market share
- -Loss of confidence
- -Legal/Regulatory violation
- -Loss of investor confidence

The business impact analysis must define what is critical, when its critical and why its critical and be in the context of the organization as a whole. Keep in mind it's relevance to the overall organization. Remember that upper management "sees the whole picture".

Your job is to present the potential impacts and the recovery alternatives and let management decide the options to take. Draw a conclusion on what the overall impact means to the company as a whole and make recommendations on what you want them to do. Once you have their commitment, you will be ready to develop a successful business resumption plan.

Comparing MPE V and MPE XL Customer Workloads

An Overview of System Performance Engineering and Workload Characterization

Julie McClung

Hewlett-Packard Company
Commercial Systems Division
19447 Pruneridge Avenue
Cupertino, CA 95014-9974

I. Abstract

Understanding the way customer applications use the HP3000 computer is beneficial not only to HP, but also to HP's customers, for it allows HP to design computers that will meet customer performance requirements. The quantitative description of how customer applications execute on a computer is referred to as workload characterization. This paper will explain what workload characterization is, why it is done, and the statistical methods used for analysis. I will discuss the system-level workload characteristics of both MPE V and MPE XL customer systems, and highlight several similarities and differences. The presentation will provide supplemental information.

II. Introduction

Customer demands for computer processing power grow over time. As new applications are developed and existing applications are extended, it is important for a computer vendor to understand how their customer workloads affect computer system performance. Simultaneously, the computer hardware and software technologies available to the vendor are advancing. A vendor has a key differentiation with respect to system performance and price-performance if its use of advancing technologies meshes well with its customer requirements.

Hewlett-Packard has made many advances in software and hardware technologies in the last few years and applied them to the HP3000. One of Hewlett-Packard's major advances is the incorporation of the PA-RISC architecture into both its commercial and technical computer product lines. The HP3000 900 Series runs on the PA-RISC architecture under the MPE XL operating system. PA-RISC provides the HP3000 with an execution engine which has unprecedented performance and price-performance advantages over its CISC predecessors. But commercial system performance requires synergy in hardware and software, not just fast hardware engines. The MPE XL operating system provides technology advances that utilize large address spaces, memory mapped I/O, integrated database and operating system services, mapped files and other software technologies to maximize the performance available from the PA-RISC engines [BUSCH87] [ZARA89].

With these changes in software and hardware coupled with advances in application resource demands it is important to understand the affect on system behaviors. This paper will examine key workload parameters for the current high end MPE XL customer base as compared to the previous MPE V customer base. Specifically, we will compare and contrast the global and workload characteristics of the Series 68 and the Series 70 customer base with the Series 95x (S950 and S955) customer base. We will examine the configurations, CPU utilization, disc utilization and TurboIMAGE characteristics so that you can compare your systems with a "typical" HP3000 high end system. It will become necessary for customers to understand these differences because many "rules of thumb" for MPE V systems are no longer applicable to MPE XL systems.

In addition, this paper will cover results from the workload characterization done on the Classic HP3000 customer base. On the MPE XL based systems, the role of Transaction Manager will be discussed, as well as the issue of compatibility mode versus native mode. Space limits the number of items to cover, so other important workload components such as networking will not be covered here.

III. Performance Evaluation Overview

This section provides an introduction to system performance evaluation of computer systems. Appendix A provides a detailed overview of workload characterization, the statistical methods used for analysis, and some results from characterization done on Series 68 and 70 customer workloads.

Performance is a measure of productivity. The most popular measures, or indicators, of performance are *utilization*, *efficiency*, *throughput*, *and response time* (response time measures will not be covered in this paper). All of these performance measures are directly related to the workload and the way in which a computer processes the workload. The term *workload* refers to the requests of the user community, that is, the programs, the data, the commands, etc.

When evaluating system performance, the primary *resources* considered are the CPU (the processor), disc, and memory. In a multi-user system such as the HP3000, each resource is shared, but each can only service one request at a time. Resources can work in parallel, for example, the CPU and disc can be processing requests simultaneously. This is sometimes referred to as overlapping operations. *Resource utilization* is the percentage of operating time during which the resource is busy. *Efficiency* is a measure of how well a resource gets its work done, defined as the ratio of work to the sum of overhead and work; as overhead goes up, efficiency goes down. Differences in the loads on various resources of a system caused by the characteristics of the workload directly affect its global (system-level) performance.

The *throughput* of a system (or resource) is defined as the amount of work performed by the system (or resource) in a given unit of time, for example, the number of jobs completed in one hour, the number of I/Os per second, etc. Throughput is usually less than the theoretical capacity of a system, which is the maximum value the system is capable of providing. The theoretical capacity can never be reached because of *system overhead*. The system starts thrashing, having to do more scheduling work than user work. For example, the Memory Manager brings a job's programs and data into memory, but by the time the job reaches the top of the dispatch queue, its programs and data are no longer in memory. There are many factors that influence throughput: workload characteristics, hardware, software, overlapping operations, and so on.

What happens when a system reaches capacity, as utilization approaches 100%? At some point a system reaches its saturation point (See Figure 1). The *saturation point* is the point at which adding one more unit of load does not yield one more unit of throughput. The best performance is achieved below the saturation point. The *peak throughput* is shown in Figure 1 as the practical capacity. Beyond the saturation point, throughput begins to decrease, overhead increases, and the system becomes less efficient. Once one or more of the resources is saturated, the entire system becomes saturated. The resource that saturates first is the bottleneck. A *bottleneck* is something that limits system performance. The goal of many performance evaluation studies is to "widen" or eliminate the bottlenecks that limit system performance. Of course, before "widening" or eliminating system bottlenecks, one must identify the bottleneck(s).

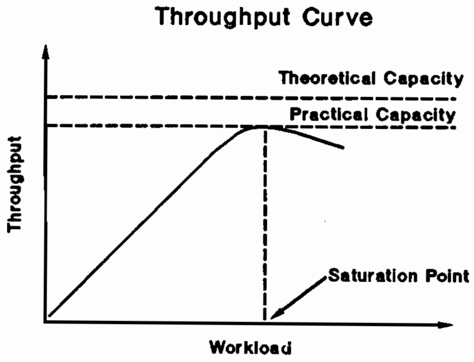


Figure 1

Workload characteristics, the quantitative description of the workload, are commonly expressed in terms of rates, such as CPU seconds per transaction and I/Os per second. Rates can be used to derive resource utilization. For example, if the maximum number of I/Os that a disc drive can do in one second is known, one could calculate the disc utilization. CPU seconds per transaction can be used to calculate the theoretical number of transactions that can be completed in an hour, or knowing how many transactions were completed in the hour, one could approximate CPU utilization.

There are many reasons for doing performance evaluation studies, the ultimate reason being to improve the cost-performance ratio. When purchasing a new computer a customer may want to do some comparisons to find the computer that best fits their workload (and budget) needs. Hewlett-Packard's Performance Briefs, for example, are used to position new computers relative to existing computers based on the performance measured while executing benchmarks, each benchmark representing a different type of workload (See [ROW90] for more information on benchmarks). System design engineers will model or test a new system in a customer-like operating environment in order to make important design decisions based on the ways in which the computer is to be used. Lastly, performance and workload characterization studies are done for system improvement and tuning of existing machines, that is, tuning the operating system, subsystems, and/or programs to better service the requests of the user community. Building models or benchmarks from customer workload characterizations make it possible to test machines in environments similar to the operating environments in which the machines will be used by the customers. Understanding workload characteristics allow analysts to make meaningful performance statements.

The process of system performance engineering at HP involves many steps. It begins by measuring customer systems. This is done by collecting system and workload data using the Snapshot package. The collection data is then reduced, analyzed, and summarized using various reduction and statistical tools. Statistical techniques, such as cluster analysis, are used to group workloads into classes based on similar resource requirements, where each class represents a different sector of the customer population (See Appendix A). The CSY Systems Performance Lab uses the customer workload data to build benchmarks which exhibit characteristics that are representative of the workload characteristics observed at our customer installations. Benchmarks are used to tune systems to best meet customer needs. This enables us to test new operating systems, subsystems, etc. in environments similar to those at customer installations.

All of the customer system and workload data is stored in a database. This database contains over 200 Series 68 collections, over 100 Series 70, and over 100 Series 900 collections.

IV. Compare and Contrast: Classic HP3000 vs. PA-RISC HP3000

This section will examine key system workload parameters for the current high end MPE XL customer base as compared to the previous MPE V customer base. Specifically, we will compare and contrast the global workload characteristics of the Series 68 and the Series 70 customer base with the Series 95x (S950 and S955) customer base. We will examine the configurations, CPU utilization, disc utilization, disc read to write ratios, and TurboIMAGE characteristics so that you can compare your systems with a "typical" HP3000 high end system. We will also examine some MPE XL specific features which affect system performance. The data listed in the following pages are observed values, not necessarily what the systems are capable of providing.

Configurations

As computers become larger and faster they must allow increased connectivity so that customers can configure additional disc drives and terminals. Comparing the Series 68 configurations to the Series 70 configurations, there is a small increase in the number of disc drives configured and very little increase in the number of terminals configured (See Table 1). Then comparing the Series 70 and Series 95x configurations, there is a 70% increase in disc drives configured and a 45% increase in the number of terminals configured. Also, there is a large increase in the minimum number of terminals configured on the Series 95x, almost three times the minimum terminal configuration seen on a Series 70. The Series 95x terminal configurations are likely to be low since PCs connected over VT (virtual terminal) have not been fully analyzed at this time.

Physical Hardware Configurations

	Disc Drives:			Terminals:		
	Average	Minimum	Maximum	Average	Minimum	Maximum
Series 68	5.6	2	15	119	24	243
Series 70	6.1	2	15	120	36	301
Series 95x	10.3	4	30	175	92	379

Table 1

CPU Utilization

In commercial computer systems, the goal is to maximize utilization of the CPU by minimizing time paused waiting for disc I/O. CPU utilization is the ratio between total CPU busy time and total system operating time. When comparing Series 95x CPU utilization to Series 68 and Series 70 utilizations, there is a trend toward higher utilizations for customers on the Series 95x systems (See Table 2). The average CPU utilization on the Series 68 and Series 70 is 65%, whereas, on the Series 95x, the average CPU utilization is 76%, a 17% increase.

The distribution of customer CPU utilization is shown on the right side of Table 2 and in Figure 2. The median is the point at which half of the observations fall below and half are above; quartile 3 (Q3) represents the 75th percentile, where the top 25% of the observations are greater than this value. On the Series 68 and Series 70 systems, the median is at 69% and 66%, respectively. On the Series 95x, the median of 78% is pulled up substantially by the ability of MPE XL to allow higher utilizations.

CPU Utilization

	Average	Minimum	Maximum	Q1	Median	Q3
Series 68	**65%**	**8%**	**100%**	**45%**	**69%**	**88%**
Series 70	**65%**	**12%**	**100%**	**47%**	**66%**	**88%**
Series 95x	**76%**	**33%**	**100%**	**58%**	**78%**	**91%**

Table 2

The MPE XL systems were designed to allow the CPU to be fully utilized by keeping it from becoming I/O bound (and consequently the CPU is not paused awaiting disc I/O). In general, MPE XL systems can be driven into the mid 90% busy range without becoming paused for disc I/O. This is very different than the Series 68 and 70 systems that "run out of gas" in the mid 80% busy range. Note also that most competitive systems display characteristics similar to the Series 68 and 70 systems.

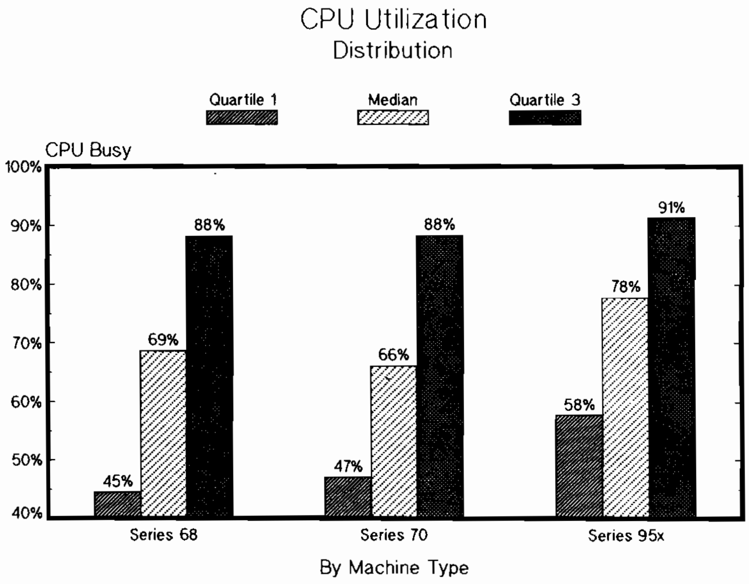


Figure 2

CPU utilization can be categorized into CPU consumed by the user community and CPU consumed by the system, which is defined as *system overhead*. *User CPU* consumption can be further divided into interactive work and batch work. Table 3 shows the breakdown of the user workload into interactive and batch as a percentage of the the amount of CPU consumed by the user community. While the comparative differences are small, there is a slight increase in the proportion of the workload that is batch when going from Series 68 to Series 70 to Series 95x systems. It is important to keep in mind that the batch component of a system's workload is growing along with the processor and continues to be a significant portion of the customer workload.

<u>Workload Mix By CPU Consumption</u>

	Interactive			Batch		
	Average	Minimum	Maximum	Average	Minimum	Maximum
Series 68	66%	0%	100%	34%	0%	100%
Series 70	64%	0%	98%	36%	2%	100%
Series 95x	63%	28%	93%	37%	7%	72%

Table 3

To ease the migration from MPE V to MPE XL, the MPE XL operating system supports the execution of object code from MPE V systems. MPE V machine emulation is made possible (supported) by the HP3000 Emulator and the HP3000 *Object Code Translator*. The HP3000 Emulator converts HP3000 machine instructions to PA-RISC instructions at execution time. The Object Code Translator (OCT) translates MPE V based HP3000 machine instructions to PA-RISC instructions prior to execution.

Translating CPU-intensive programs can cut the execution time of emulated programs. The advantage of OCT over emulation of MPE V programs is in the execution speed of the compiled code, where the largest savings is in the elimination of instruction fetch and decode cycles required by the emulator at run time [BUSCH87]. Both of these modes of operation, emulated and translated, are known as *compatibility mode* (CM). The optimal mode of operation is *native mode* (NM). For the best performance, program source code must be compiled by MPE XL native mode compilers into native mode object code.

Series 950 CPU utilization can be broken down by execution mode: emulated, translated and native mode (See Figure 3). Emulation mode programs consume 15% of the CPU and translated mode (OCT) programs consume 19% of the CPU. Thus, on the average, 34% of the Series 950 customer's system CPU busy time is spent executing compatibility mode code, the remaining 66% of the CPU time is consumed executing native mode code. This CM execution time represents a significant loss of processing capability of the system. If all of this CM code were recompiled into NM, overall system performance would improve by 10-20%. Individual application performance will vary based on the amount of CPU time spent in user code and libraries. To achieve maximum system performance it is imperative to migrate applications to NM whenever possible.

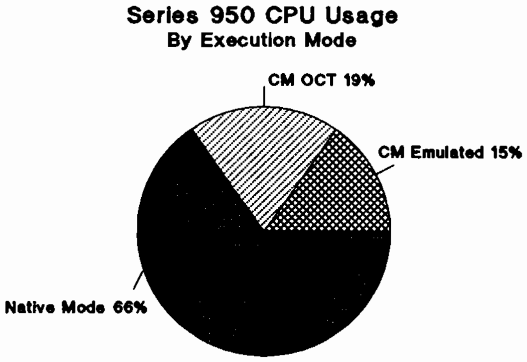


Figure 3

Compatibility mode execution of programs on MPE XL requires a switching mechanism. Any program running in compatibility mode requires a *switch* to native mode in order to use features of the operating system that are in native mode. Likewise, programs running in native mode require a switch to compatibility mode to use features of the operating system that are still in compatibility mode. Unfortunately, switching is a costly operation: about 1000 instructions for a switch to NM and 1500-2000 instructions for a switch to CM. The switch to CM is more costly because the parameters on the NM stack must be copied to the CM stack, whereas for a switch to NM, no copy is needed because native mode can access the CM stack directly. Generally, if a customer has a NM switch rate of greater than 200-300 switches per second on a Series 95x, it is recommended that the customer migrate application code to native mode.

Figure 4 show the distribution of NM switch rates for Series 950 and Series 955 customers. For the Series 950, the largest percentage of customers have switch rates between 150 and 200 per second, 58% of the customers have switch rates above 200 per second, and 29% have switch rates above 300 per second. On the Series 955, the majority of customers have NM switch rates greater than 300 per second, with several customers above 700 switches per second. Considerable performance gains could be realized if these switch rates were reduced.

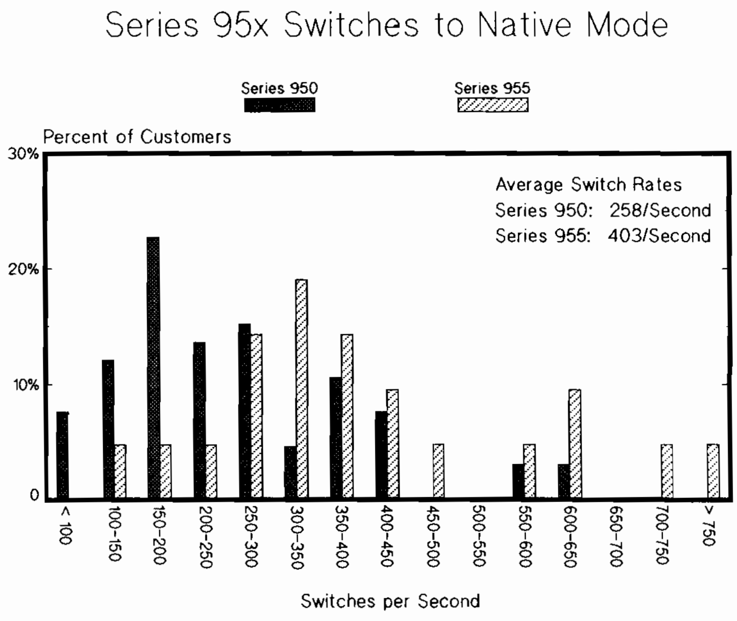


Figure 4

Physical Disc I/O

Performance on MPE XL systems is intended to scale directly with respect to processor speed, main memory availability, and workload. This implies that disc I/O, as well as waits for disc I/O, be eliminated whenever possible. Comparing Series 70 physical disc I/O to Series 95x physical disc I/O, the average read rate on the Series 95x is 40% less than the read rate on the Series 70 (See Table 4). The write rate on the Series 95x is 70% less than the write rate on the Series 70. Although there are decreases in the number of I/Os completed, the amount of data being transferred per I/O has increased by 75% for read transfers and more than ten times for write transfers on the Series 95x compared to the Series 70.

Figure 5 shows the distribution of I/O rates for customers on each of the systems being compared. The average I/O rate at each customer site is used. Percentiles are used to show the spread of I/O rates; like the quartiles in Figure 2, the 40th Percentile, for example, is the point at which 60% of the customers have an I/O rate higher and 40% of the customers have an I/O rate lower than the 40th percentile value. The dotted line represents the 90th percentile for Series 95x I/O rates. Ten percent of the Series 95x customers have rates higher than 25 I/Os per second. Over 30% of the Series 68 customers and more than 50% of the Series 70 customers have I/O rates greater than that.

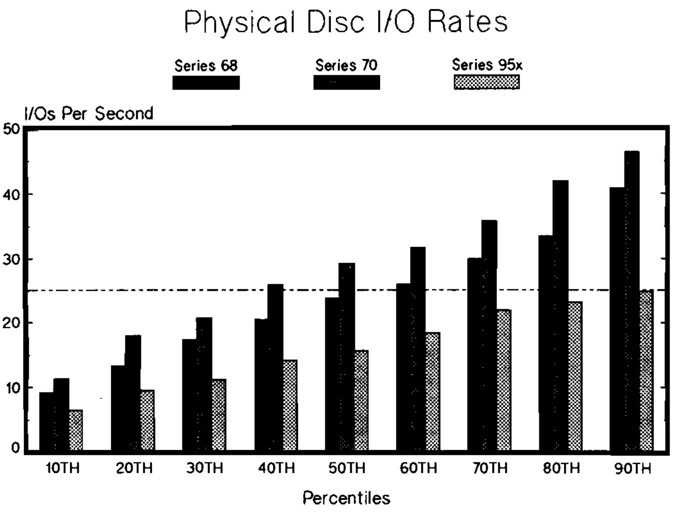


Figure 5

Disc transfer sizes are also different. In Table 4, bytes per read are calculated as total bytes transferred for reads divided by the total reads for each site. The net result of decreased I/Os and larger transfers per I/O on the Series 95x is that, overall, more data is being transferred (and processed), as shown in the bytes per second averages. This allows the system to amortize the cost of disc I/O over a much larger transfer size. Total (read + writes) data bytes transferred per second has more than doubled from Series 70 to the Series 95x, going from 81,158 bytes per second to 164,806 bytes per second. The net result is fewer physical disc I/Os per transaction for MPE XL systems and more efficient use of the disc subsystem.

The differences between Series 70 and Series 95x physical I/O characteristics are significant. Due to sophisticated, self-adjusting prefetching algorithms, larger memories, and consequently larger caches, the physical read rate is actually lower on the Series 95x than on the Series 70, even though the Series 95x machines are processing more transactions. The MPE XL Memory Management policies allow it to prefetch extra pages under certain conditions in an effort to reduce I/Os and chances for future page faults. It also has predictive mechanisms that allow it to prefetch data before a page fault occurs for sequential access. This results in improved responsiveness since a user's transaction would rarely wait for a disc read to complete. Also, MPE XL requests I/O in page increments, thus the minimum transfer is one page (4 KBytes). This also explains the larger transfer sizes observed on MPE XL systems. Due to the differences in the way the operating systems do disc I/O, one must be careful in comparing disc I/Os as a metric for MPE V and MPE XL.

Physical Disc I/O

	READS			WRITES		
	Average	Minimum	Maximum	Average	Minimum	Maximum
Series 68						
Rate/sec	12.2	1.2	43.9	12.2	2.3	32.8
Bytes/IO	4,189	924	16,807	1,533	590	10,034
Bytes/sec	51,073	2,830	383,936	18,810	3,341	165,318
Series 70						
Rate/sec	14.6	1.5	85.1	16.2	0.8	41.9
Bytes/IO	4,358	550	15,989	1,682	750	8,717
Bytes/sec	54,961	5,346	305,694	26,197	3,370	206,224
Series 95x						
Rate/sec	9.1	0.6	24.0	5.4	0.6	11.0
Bytes/IO	7,585	5,033	14,336	17,987	10,548	27,777
Bytes/sec	69,766	3,578	171,188	95,040	11,095	201,027

Table 4

The MPE XL *Transaction Manager* and its tight integration with the file system and TurboIMAGE database system provides data integrity and increased performance by eliminating and gathering disc writes. It is one of the main reasons for the decrease in physical writes on the Series 95x compared to the Series 68 or Series 70. (Transaction Manager will be covered in the next section.) Furthermore, the Memory Manager uses techniques to "gather" dirty pages in memory that are consecutive on disc and issue a single write whenever possible [ZARA89]. This explains the lower write rates as well as the larger transfer sizes per write.

CPU and memory speeds are advancing faster than disc drive speeds. Consequently, a design goal of MPE XL was to improve the efficiency of the disc I/O subsystem. As transfer sizes for disc I/Os increase, the amount of data that disc drive can transfer in a second also increases. Figure 6 helps to illustrate the benefits of doing larger transfers per I/O. In addition to the cost of actually transferring data, there are set-up costs involved with every I/O. Each I/O requests incurs the costs of overhead, which includes setting up the disc controller and positioning the disc head to the proper location on disc (seek and latency). The curves shown in Figure 6 illustrate the effective speed up for transferring data as transfer sizes are increased on MPE XL. Based on a fixed amount of data to be transferred, transfer sizes are varied and compared to the average read and write transfer sizes on the Series 70.

For example, the disc drive receives a read request for 70,000 bytes of data. On MPE V and the Series 70, the request would be processed in 16 I/Os, at 4358 bytes per I/O. On MPE XL and the Series 95x, the request would be processed in about 9 I/Os, at 7585 bytes per I/O. At Series 70 transfer rates, the transfer will take approximately 520 milliseconds. At the Series 95x transfer rates, the transfer will take approximately 315 milliseconds. Thus, the effective disc speed up on MPE XL for transferring data in larger chunks is 1.7. The savings comes from eliminating the overhead of 7 I/Os. In terms of disc throughput: at Series 70 transfer rates, the disc drive could process almost 2 of these transactions per second, whereas at the Series 95x transfer rates, 3 of these requests could be processed in a second. For write transfers on MPE XL , the effective disc speed up is even greater due to the larger differences in average transfer sizes used on the Series 70 and Series 95x.

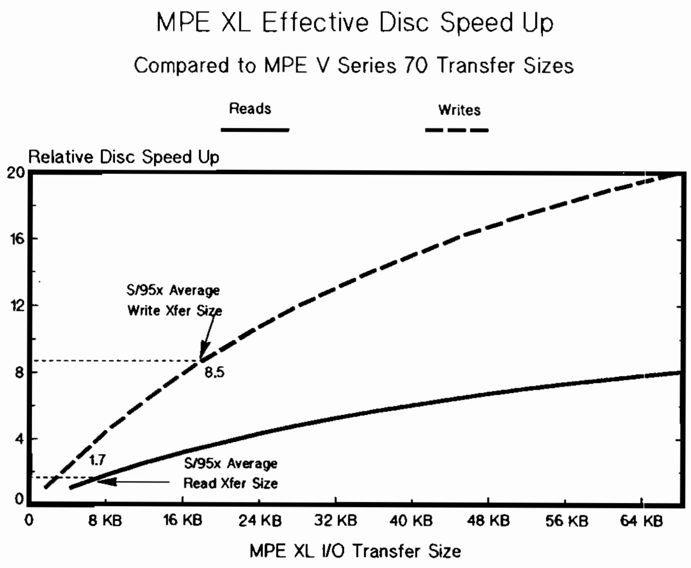


Figure 6

As a result of MPE XL's unique method of managing disc I/O, MPE XL disc I/O characteristics are very different from MPE V's. Not only does MPE XL require fewer disc I/Os to complete the same task but disc I/Os tend to be larger, more random, and more "bursty" in nature. This means disc solutions, such as disc controller cache, which were designed specifically to meet the disc I/O requirements of the MPE V systems, are not effective in the MPE XL environment [ARM90].

The ratio of physical reads to physical writes is also useful in characterizing workloads. The Series 68 customer workloads generate 1.1 reads for every write. The Series 70 customer workloads generate 1.3 reads for every write. And the Series 95x customer workloads generate 1.8 reads for every write (See Table 5). Comparing the Series 70 and the Series 68 read to write ratios, Series 70 workloads request 18% more reads to every write than the Series 68 workloads. The Series 95x workloads are generating 38% more reads to every write when compared to the Series 70 workload read to write ratios. The read to write ratios further illustrate the gathering and elimination of disc writes by the MPE XL Memory Manager and Transaction Manager on the Series 95x systems.

<u>Disc Read to Write Ratios</u>

	Average	Minimum	Maximum
Series 68	1.1	0.3	2.9
Series 70	1.3	0.1	28.9
Series 95x	1.8	0.6	2.8

Table 5

<u>TurboIMAGE Physical Disc I/O</u>

TurboIMAGE database applications are typically a significant part of most customer workloads. On many systems they are the vast majority of workload types. Thus, I/O to TurboIMAGE databases makes up a significant portion of the physical I/O on customer systems.

Although there are a few customers who do not use TurboIMAGE databases, on the average, for Series 68 customers, almost half (48%) of all physical disc reads and 26% of physical disc writes are done on TurboIMAGE databases (See Table 6). On the Series 70 customer systems, 59% of physical disc reads and 30% of physical disc writes are to TurboIMAGE databases. Based on the I/O rates per second, this represents a 40% increase for reads and a 59% increase for writes over the TurboIMAGE I/O rates on the Series 68 customer systems.

Although the Series 95x customer systems process much more database transactions, only 41% of all physical reads and 9% of physical writes are to TurboIMAGE databases. The read rate on the Series 95x is 50% less than the read rate on the Series 70 and the write rate is 90% less than the write rate on the Series 70 for database I/Os.

Another significant change is in the TurboIMAGE transfer sizes on the Series 95x when compared to the Series 70. In Table 6, *read transfer sizes* (bytes/IO) are calculated as the total bytes transferred for image reads divided by total image reads for each site. Transfer sizes on the Series 95x are roughly 2 pages (a page is equal to 4 KB) for database files. For writes, this is more than five times the average database write transfer size on the Series 70.

Physical TurboIMAGE Database I/O

	READS Average	Minimum	Maximum	WRITES Average	Minimum	Maximum
Series 68						
ImageIO/All IO	48.0%	0.0%	97.9%	26.3%	0.0%	96.9%
Bytes/Image IO	5,011	11	22,362	909	15	3,865
Rate/Sec.	5.3	0.0	23.9	2.9	0.0	31.6
Series 70						
ImageIO/All IO	59.1%	0.0%	99.7%	30.3%	0.0%	81.7%
Bytes/Image IO	4,765	522	22,494	1,531	266	23,018
Rate/Sec.	7.4	0.0	34.1	4.6	0.0	23.4
Series 95x						
ImageIO/All IO	40.9%	0.0%	82.3%	9.0%	0.0%	24.3%
Bytes/Image IO	8,280	5,249	13,784	7,948	5,446	15,040
Rate/Sec.	3.5	0.0	10.6	0.5	0.0	1.2

Table 6

The differences between Series 70 and Series 95x physical database I/O characteristics are due mostly to the reasons already mentioned for the overall physical I/O characteristics. Reads are down due to prefetching, larger memories, and larger caches. Writes are down mainly due to the Transaction Manager and the Memory Manager's ability to gather dirty pages and write them to disc in a single I/O. On MPE XL systems, all TurboIMAGE database modifications go through the Transaction Manager. Consequently, many of the overall physical disc writes eliminated by the Transaction Manager are database writes. Transaction Manager is an important part of any application which uses TurboIMAGE databases.

In order for the Transaction Manager (XM) to provide data integrity and eliminate disc writes, the Transaction Manager logs before and after images of modified data pages to a system logfile. (For more detail on Transaction Manager, refer to [ZARA89].) The logfile is posted (written) to disc frequently based on several system conditions. Then, at checkpoint time, all of the modified user data pages are written to disc en mass. This also explains the lower writes rates as well as the larger transfer sizes per write.

Thus, the elimination of TurboIMAGE disc writes must be considered in conjunction with writes to Transaction Management logfile. For the Series 95x customer data in Table 6, disc writes to the XM logfile were occurring at a rate of 1.6 per second. Assuming, in the worst case, that all of the XM activity was generated by TurboIMAGE database modifications, the combined write rate for TurboIMAGE databases and the XM logfile is 2.1 writes per second, still substantially lower than the TurboIMAGE write rates observed on the Series 68 and Series 70.

As mentioned, the XM logfile is posted for several reasons. The main reason observed at customer sites is due to Block on Commits. On average, over 70% of the logfile writes were due to Block on Commits. Block on Commits occur when a process (or group of processes) blocks itself until the next post occurs. Some transactions force the logfile to be posted, others just block the process until the next post. Twenty-eight percent of the logwrites were caused by the timer. The timer is triggered by Transaction Management activity, this ensures that the logfile will be posted within one second or less. Coinciding with the high Block on Commit post percentage, the size of the logfile buffer being posted is 1 page sixty-nine percent of the time. The logfile buffer in memory can hold up to sixteen pages of data. If the percentage of posts due to Block on Commits is reduced, even more I/Os could be eliminated by doing larger posts of the logfile.

Typically, Block on Commits occur when ILR (Intrinsic Level Recovery) or Roll-back Recovery is enabled for a TurboIMAGE database and when purging files. Enabling ILR for databases on MPE V was necessary to prevent broken chains, but on MPE XL Transaction Management prevents broken chains, ensuring physical database integrity by always keeping the database on disc in a consistent state (no partially complete transactions will be posted to disc). For increased performance and faster response time, it is recommended that ILR be disabled on all TurboIMAGE databases.

V. Summary

Hewlett-Packard has made many advances in software and hardware technologies in the past few years and applied them to the HP3000 Series 900 product line. The coupling of PA-RISC and MPE XL is designed to provide a base for evolutionary increases in performance, availability, and features. Performance is intended to scale directly with processor speed, main memory availability, and workload.

MPE XL was designed to allow the CPU to be fully utilized by minimizing disc I/O and eliminating process waits for disc I/O whenever possible. We have seen customer CPU utilization on the Series 95x to be much higher than what was observed on the Series 68 and Series 70.

Although there are many external similarities between MPE V and MPE XL, internal differences exist that have an impact on disc utilization and performance. One of the big differences is in the way disc I/O is managed. MPE XL advances in transaction management and memory management reduce the number of physical disc I/Os required as well as increasing the amount of data transferred per I/O. This allows the software cost of doing an I/O to be amortized over larger transfer sizes. Also, the effect of doing larger I/Os on MPE XL effectively increases the capacity and reduces the utilization of a disc drive by increasing its efficiency and cutting down I/O overhead.

There are some things that are limiting performance on MPE XL systems, including compatibility mode CPU consumption and Intrinsic Level Recovery (ILR) enabled on TurboIMAGE databases. First, compatibility mode CPU cycles still consume, on average, 34% of a customers CPU busy time. Consequently, switch rates from compatibility mode to native mode are quite high. If all compatibility mode code were recompiled into native mode, overall system performance would improve by 10-20%. To achieve maximum performance it is imperative to migrate applications to native mode and use native mode services whenever possible. Second, with the integrated transaction management services provided by MPE XL, it is no longer necessary to have Intrinsic Level Recovery (ILR) enabled on TurboIMAGE databases in order to provide data integrity by preventing broken chains. It is important that customers disable Intrinsic Level Recovery on all of their TurboIMAGE databases in order to achieve the best performance.

Finally, Hewlett-Packard uses system performance engineering to ensure that computers meet customer demands and future growth requirements. This involves understanding the workload characteristics exhibited by all types of customers and the affect that they have on system performance. Statistical techniques allow us to group customer workloads based on the similar demands they place on the system. With the customer workload characterizations, we can build benchmarks and models in order to understand the workloads' effects on system performance. With that understanding, HP can tune and design computers to process the requests of the user community in the most efficient ways.

VI. Acknowledgments

I would like to thank my management staff, Paul Petersen, Ron Kliesch, and Rich Friedrich, for the support and time given to me for writing this paper. I would like to thank Frank Rowand and Ron Kliesch for encouraging me to submit my abstract. And I would like to give special thanks to Rich Friedrich for his guidance, insight, and enthusiasm in helping me with this paper.

Appendix A: Overview of Workload Characterization And Results from MPE V Workload Characterization

Computer performance is the measure of resource utilization and efficiency in which a computer processes the requests of the user community. The requests are submitted by the user community by inputting sets of programs, data, and commands to the system. All of this input constitutes the workload. Performance cannot be expressed in quantities independent of a system's workload. No performance evaluation problem can be adequately solved if the workloads to be processed by the system are not specified. Thus, *workload characterization*, the quantitative description of the workload characteristics of a system, is a very important part of any performance evaluation study.

Workload characteristics are used for input to models or benchmarks. Benchmarks allow engineers to artificially load a system to perform controlled experiments. Hewlett-Packard's Performance Briefs, for example, are used to position new computers relative to existing computers based on the performance measured while executing benchmarks, each benchmark representing a different type of workload [ROW90]. System design engineers will model or test a new system in a customer-like operating environment in order to make important design decisions based on the ways in which the computer is to be used. Benchmarks and models from workload characterization studies are done for system improvement and tuning of existing machines, that is, tuning the operating system, subsystems, and/or programs to better service the requests of the user community. Understanding a system's workload characteristics allow one to make meaningful performance statements.

Workloads are quantitatively described using a set of parameters. The parameters are chosen to describe the workload characteristics of a unit of work, or workload basic component. Typically for interactive workloads, transactions are used as the unit of work. Transactions at the physical level are often defined by terminal reads. At the functional level, a transaction may be defined, for example, as the processing of an order, which, in many cases, involves several terminal reads. When selecting parameters, it is important to avoid parameters that are load sensitive, like response time, which is a result of the workload, not a part of the request. Parameters can be represented as amounts, proportions, rates, distributions, etc. Often, parameters are expressed as rates, such as CPU seconds per transaction and I/O operations per transaction.

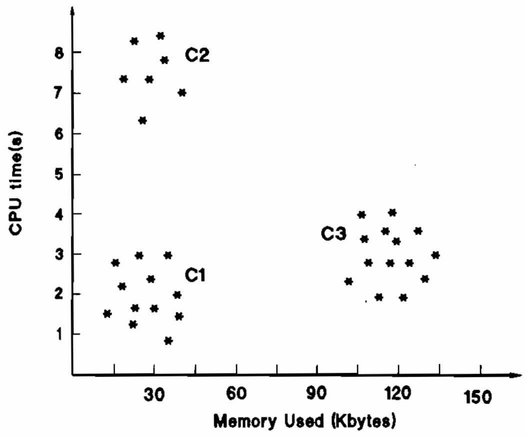


Figure 7

Prior to characterizing workloads, data is collected from the target system or systems. Once data has been collected and the parameters describing each component have been extracted, the resulting amount of data could be considerable. The need for statistical techniques for the purpose of identifying group of components with similar characteristics becomes apparent. As shown in the ideal example illustrated in Figure 7, components described with 2 parameters can be classified into similar group using classic data analysis, or even visual inspection, but in most cases, multidimensional data analysis techniques are needed. Among these techniques is *cluster analysis*; the goal of cluster analysis is to partition a set of points into groups, called clusters or classes. The benefit of this approach to workload characterization is that the large number of components in the actual workload can be categorized into a small number of classes in which the members of each class have similar resource requirements. The categorization yields a compact workload description, then when building a model or benchmark, a representative component can be extracted from each cluster and used in the model or benchmark to represent members of that cluster.

Cluster analysis has been performed on the high-end Classic HP3000 customer systems. (MPE XL workload characterization and cluster analysis is an on-going activity.) Each customer workload was divided into three major workloads: short interactive, long interactive, and batch. For the interactive workloads, the basic unit of work is the transaction, which is generally defined as the activity between two terminal reads. Short interactive and long interactive workloads are separated by the CPU required for each transaction. Long interactive transactions require more than two CPU seconds per transaction. Some examples of long interactive transactions are online compiles and online database queries. Short interactive (OLTP-type) transactions are the predominant transaction type observed on customer systems. The rest of this appendix briefly reviews the results from the clustering of short interactive workloads for high-end customers on Series 68 and Series 70 systems.

Cluster analysis works best with only a few metrics. It is important to retain as much information about the workloads in as few metrics as possible. Five metrics were selected to best describe the transactions' demands on the resources: CPU and I/O. The metrics used were: CPU seconds per transaction, IMAGE reads per transaction, IMAGE writes per transaction, non-IMAGE reads per transaction, and non-IMAGE writes per transaction. Non-IMAGE I/Os are the reads and writes to MPE files, KSAM files, and directory files, basically all I/Os that were not to database files.

Prior to the application of cluster analysis, the outliers were trimmed and the metrics were scaled to make them comparable. The procedure FASTCLUS, from the SAS Statistical Software Package, was used for clustering the workloads [SAS88] [ARTIS86]. The analysis resulted in six clusters, each differing from one another in various ways.

The resulting clusters are illustrated in the boxplots in Figure 8. Boxplots are used to show the spread of data. A horizontal line is drawn through the box at the median of the data, the upper and lower ends of the box are at the upper and lower quartiles, and the data points that are very extreme are shown by the asterisks. The boxplots help to illustrate that although two clusters may be similar on one metric, they vary on others. For example, boxplot 8d shows that cluster 2 and cluster 5 have about the same number of IMAGE reads per transaction values, but the number of non-IMAGE reads per transaction (boxplot 8b) for each is very different.

Cluster analysis is effective in grouping similar workloads. Each cluster workload will experience different system behaviors as a result of different resource requirements. It was discovered that customers using the same software application may be using it in different ways. Several applications appeared in more than one cluster. One can't assume all customers running the same application use it in a similar way.

In conclusion, workload characterization is an important part of any performance evaluation study. Cluster analysis may not be necessary in all cases, but it is a useful tool for grouping similar workloads into clusters. A representative member can be extracted from a cluster to represent all members of that cluster for use in benchmarking or modeling.

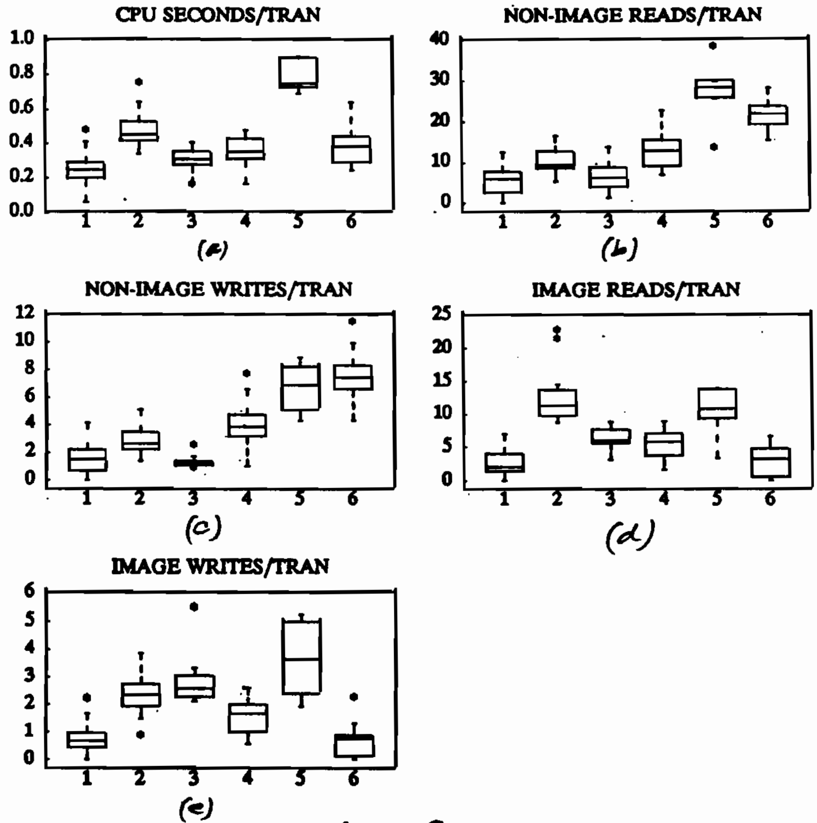


Figure 8

References and Further Reading

[ARM90] Armstrong, Melody, HP Fiber-Optic Link, *Interact*, March, 1990.

[ARTIS86] Artis, H. P., Workload Characterization Using SAS PROC FASTCLUS, *Workload Characterization of Computer Systems and Computer Networks*, G. Serazzi (editor), North-Holland, Amsterdam, 1986.

[BUSCH87] Busch, J. R., Kondoff, A. J., Ouye, D., MPE XL: The Operating System for HP's Next Generation of Commercial Computer Systems, *HP Journal*, December, 1987.

[FERR83] Ferrari, D., Serazzi, G., Zeigner, A., *Measurement and Tuning of Computer Systems*, Prentice-Hall, Englewood Cliffs, New Jersey 1983.

[ROW90] Rowand, Frank, What Does This Benchmark Mean To You?, *Proceedings of the 1990 INTEREX HP Users Conference*, August, 1990.

[SAS88] *SAS User's Guide: Statistics*, 1988 Edition. 1988. Cary, North Carolina: SAS Institute Inc.

[ZARA89] Zara, A., Friedrich, R., TurboIMAGE/XL and the Integration with MPE XL Transaction Management Services, *Proceedings of the 1989 INTEREX HP Users Conference*, September, 1989.

The HP 3000 Open Systems Environment

Janet Garcia

Hewlett-Packard
19111 Pruneridge Avenue
Cupertino, CA 95014

HP's NewWave Computing strategy is to provide a multivendor client-server environment for distributed applications. The underlying framework for NewWave Computing is a commitment to a standards-based architectural framework -- a commitment that allows interoperability and an integrated view into the computing environment. Open system components make possible the cooperative services for transparent database access, remote applications, shared office services, and integrated network and system management services. In the future, the HP 3000 intends to support Distributed NewWave by providing server-based object management services.

HP has based its corporate strategy on product innovation with the implementation of recognized standards and has a proven track record in providing industry standards leadership. HP was a founding member of the Open Software Foundation (OSF), currently chairs or co-chairs major standards committees such as IEEE POSIX and OSI, and holds positions on the boards of OSF and X/Open.

Both the HP 3000 and HP 9000 share a common foundation for standards in networking, relational databases, languages, and user interfaces. The HP 9000 is a leader in providing standards-based systems for the commercial and technical markets. The HP 3000 is a leader in providing commercial OLTP performance and functionality and leverages HP expertise and leadership for its standards offerings.

This paper introduces the reader to the HP 3000 Open Systems Environment. It highlights well-known products and services, but they are discussed within an open systems framework.

HP 3000 OPEN SYSTEMS BACKGROUND

Open Systems Definition

X/Open describes open systems as "Software environments consisting of products and technologies which are designed and implemented in accordance with 'standards' established and de facto; that are vendor independent; and that are commonly available."

The HP 3000 supports X/Open's open system definition. In the past, many users have equated open systems with Unix. Although Unix has played a key role in the definition of open system standards, an open computing environment is more than just an operating system. Interoperable networking and portable application programming interfaces in all key components of a computing environment are also required. The HP 3000 open systems framework includes standards in:

- languages and tools for ease of application development
- database management for common distributed data access
- networking for interoperability in multivendor system environments
- operating system interfaces for increased application portability
- user interfaces for a consistent look and feel in applications

Standard application programming interfaces in each of these areas make changes in the underlying technology transparent to the end user. Thus, the developer can concentrate on the programming interfaces, not the underlying mechanics of the particular operating system. The HP 3000 provides performance-added value to these portable programming interfaces.

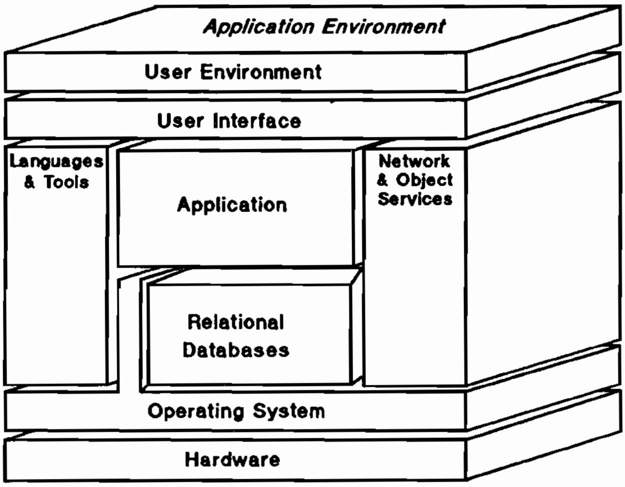


Figure 1. An open environment for NewWave Computing
This figure shows the application environment framework for the HP 3000 Open OLTP Environment. An application requires user interface, language, database, networking, and operating system standard interfaces and services to maximize portability and interoperability.

Key Benefits

This framework allows HP 3000 customers to take advantage of the benefits of open systems. Among these benefits are:

For End Users
- increased application availability
- transparent access to enterprise information
- productivity gains of a common user interface

For MIS Administrators
- hardware and software vendor independence
- increased scalability for effortless growth
- better integration of multivendor environments
- increased solutions flexibility

For Application Developers
- single source code for multiple platforms
- reduced porting costs
- increased user base
- reduced training costs

For Solution Vendors
- increased application base
- increased customer base
- faster availability of new technologies

What Standards?

Because no single standards source today provides a complete environment for portability and interoperability, the HP 3000 strategy is based primarily on three key components:
- X/Open interoperability and portability guidelines
- de facto market standards
- SAA interoperability guidelines

X/Open

X/Open is an international consortium of computer vendors working to create an internationally supported, vendor-neutral Common Applications Environment (CAE) based on de facto and industry standards. X/Open does not develop products. It endorses international standards where they exist, and adopts de facto standards where they do not exist. X/Open's efforts are embodied in a series of publicly available reference documents called the X/Open Portability Guide (XPG), which has become widely accepted as the dominant working definition of a full open systems environment. This standards group provides a set of practical unifying standards for users and vendors that bridges the gap between proprietary and industry standards, open systems, and IBM's SAA.

De facto standards

The HP 3000 strategy to support de facto market standards represents a long-term commitment to protect customer investments in the HP 3000 and in other vendor platforms as well. HP's support of de facto standards includes PC LAN connectivity with resource-sharing products such as Novell Netware and LAN Manager; and multivendor database and 4GL offerings from ISVs such as Oracle, Ingres, and Sybase.

IBM SAA

IBM's Systems Application Architecture (SAA) promotes development of consistent applications by providing standardized interfaces for IBM's system environments (MVS, VM, OS/2 EE, and OS/400). SAA defines Common Communications Services (CCS), Common User Access (CUA), and Common Programming Interfaces (CPI) for IBM's proprietary systems. The HP 3000 open systems strategy is to support components of SAA to facilitate interoperability and coexistence with IBM. In addition, the HP 3000 open systems strategy under the HP NewWave Computing framework provides more standards benefits than IBM's SAA.

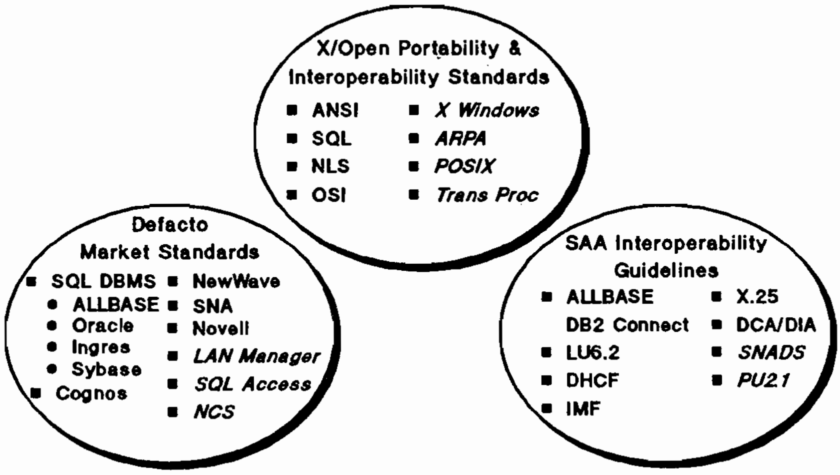


Figure 2. The HP 3000 Open OLTP driving forces
Examples of de facto and industry standards are shown. Note that these standards overlap. X/Open encompasses several de facto commercial standards. Several IBM SAA specifications are also viewed as de facto standards for enterprise computing.

The Goals

The goals for the HP 3000 Open OLTP Environment are:

- Provide commercial functionality today while standards evolve. Although standards organizations may lag behind technology advances in OLTP computing, the HP 3000 will continue to provide customers value-added functionality in the areas of system and network management, performance, and cooperative computing.
- Provide primary support for OLTP applications. The HP 3000 will support those standards relevant to OLTP commercial computing in both traditional and client-server configurations.
- Provide increased commercial application portability. The HP 3000 is committed to reducing the time it takes to port applications from commercial Unix by providing key programming interfaces adopted by standards groups.
- Provide coexistence with multivendor system solutions. Coexistence with IBM and Unix allows customers to protect their hardware and software investments. More integrated system solutions for HP 3000, HP 9000, MS-DOS®, and OS/2 environments will also be available.
- Provide HP 3000 investment protection. The HP 3000 will provide a long-term commitment to strategic, open interfaces while maintaining existing HP 3000 functionality. Customers can retain the lasting value of current applications and also adopt new solutions based on open systems.

The HP 3000 strategy is to provide a phased roll-out for additional standards products and application programming interfaces along with design guides, consulting, and training services. The products and services will evolve to provide an even more robust application environment including integrated CASE technologies for application development. HP's next generation data repository product will support open standards. It will support integration of third-party products and data sharing among databases, front-end CASE tools, and 4GL/3GL tools.

1. DATABASE MANAGEMENT, LANGUAGES, AND TOOLS

The main database strategy goals for HP 3000 open systems are:

- provide choice of key, multivendor ISV database solutions
- provide interoperability, standards, and best performance with ALLBASE/SQL
- provide application portability by supporting leading multivendor development environments and 4GL tools

Multiple Relational DBMSs

Some customers want a relational database solution that runs on multivendor platforms such as HP, DEC, IBM, and Unix systems. The HP 3000 offers these customers the choice of popular database and 4GL solutions from leading ISVs such as Oracle, Ingres, and Sybase. These products provide PC-based solutions for client-server HP 3000 configurations. They also provide maximum application portability across a wide variety of platforms.

The HP 3000 supports ALLBASE/SQL, with the industry-standard ANSI SQL interface. ALLBASE/SQL currently conforms to the ANSI SQL 1 standard, is X/Open XPG3-compliant, and will be extended to conform to the ANSI SQL 2 standard and X/Open's XPG4 SQL specifications.

Since ALLBASE/SQL is dedicated to the HP 3000 and HP 9000 only, it can more effectively take advantage of the underlying operating system and hardware technology to optimize performance, data integrity, and availability. The HP 3000 strategy is to continue to provide the best OLTP performance with ALLBASE/SQL and to position ALLBASE/SQL as the strategic database for new application development. The ALLBASE/TurboCONNECT product allows customers to develop new applications using ALLBASE as the database platform and still continue to have read-access to TurboIMAGE data.

ALLBASE/SQL also gives customers application interoperability and portability by supporting popular 4GL application development tools. Cognos' Powerhouse, Ingres' Application By Forms, FOCUS, and InfoCentre's Speedware tools can be used with ALLBASE/SQL. The Ingres and Cognos application development toolsets provide compatibility across a wide variety of platforms such as HP, DEC, IBM, and many Unix systems. FOCUS is a de facto standard 4GL application development and report writing tool dominant in the mainframe market. The HP 3000 will continue to have more 4GL solutions available with ALLBASE/SQL.

IBM database interoperability is key for commercial enterprise computing. The ALLBASE/DB2 Connect product provides application- level calls allowing remote HP 3000 applications direct read and update access to IBM DB2 databases on an S/370 mainframe.

SQL Access Interoperability for Diverse Databases

Even with the existing ANSI and X/Open SQL standards, users may still not have the interoperability they desire among different SQL databases. For that reason the SQL Access Group has been formed to develop a remote database access standard that will solve this problem. The SQL Access Group consists of key industry relational database suppliers such as Oracle and Informix, as well as major vendors such as HP, DEC, Tandem, Sun, and NCR. The SQL Access Group also has X/Open representation.

Based on the ISO Remote Database Access standard, the SQL Access standard will provide an interoperability solution for customers that wish to link different SQL databases running on different machines through standardized queries. In the future, users will be free to mix and match SQL-Access compliant products to meet their information management needs.

2. HP 3000 AND STANDARD OPERATING SYSTEM SERVICES

The HP 3000 will support the IEEE POSIX 1003.1 and plans to support the 1003.2 operating system interface standards. HP is also taking a leadership role in helping to define future OLTP standards, such as the X/Open Distributed Transaction Processing (DTP) standard. In brief, the X/Open DTP specifies services for database management systems such that transactions and administrative services are consistently managed among all DBMSs. This includes support of the resource, transaction, and communications management facilities. The remainder of this section focuses on POSIX support relative to the HP 3000. For more details on POSIX, read "POSIX And X Windows On MPE XL" (paper #3184).

Why POSIX on the HP 3000?

The main reason behind POSIX support on the HP 3000 is to increase the range of application solutions available to HP 3000 customers. The POSIX standard simplifies application portability because software suppliers can expect a well-recognized set of operating system interface services across different vendor platforms. Thus, it provides the ability for users and VABs to develop common applications for the HP 3000 and other POSIX-compliant systems, such as the HP 9000. POSIX also provides a high level of compatibility with applications written for Unix systems.

Since 4GL tools often shield the developer from the interfaces of the underlying database and operating system, POSIX on the HP 3000 may be essentially transparent to an application developer. POSIX, however, provides the services that will allow new 4GL tools and other tightly integrated subsystems to be easily ported to the HP 3000.

Providing POSIX allows customers to be secure in the HP 3000's commitment to open systems. POSIX support is becoming a measure of vendor openness. The U.S. government's National Institute of Standards and Technology (NIST) has adopted POSIX as the basis of its Federal Information Processing Standard (FIPS). OSF and X/Open have endorsed and also adopted POSIX specifications.

What is POSIX?

An operating system performs services for an application program, such as disk reads, process creations, and file management. Incompatibilities exist because each operating system performs these functions differently: the syntax of system calls to these services are different, the calls require different parameters and formats, and the order of the parameters matters. To support application portability across systems, the IEEE P1003 group, accredited by ANSI, was chartered to specify a Portable Operating System Interface (POSIX) standard for application development. The POSIX group was formed to specify the functions and services an operating system must support and to specify the application programming interfaces to these services. POSIX will be the set of interface services in common between the different flavors of Unix on the market today, DEC VMS systems, and the HP 3000.

The POSIX portable open systems environment definition is being developed by several committees. These standards include:

Working Group Status

Approved:
- 1003.1 Core POSIX System Interface

Draft:
- 1003.2 POSIX Shell and Tools
- 1003.3 POSIX Conformance Testing
- 1003.4 Real-Time Extensions

Evolving:
- 1003.5 ADA Language Bindings
- 1003.6 Security
- 1003.7 System Administration
- 1003.8 Networking
- 1003.9 FORTRAN Language Bindings
- 1003.10 Supercomputer Profile
- 1003.11 Transaction Processing
- P1201 POSIX User Interfaces (Based on X Windows)

HP 3000 and POSIX Plans

Specifically, HP has announced support of the POSIX 1003.1 standard on the HP 3000 by 1991. The HP 3000 also plans to support the POSIX 1003.2 Shell and Tools specification. The POSIX 1003.2 standard defines the interface to shell services, commands, and common utility programs as derived from the Unix operating system. This specification is currently under draft and is expected to be approved sometime in 1990.

In general, the HP 3000 support for POSIX will be focused on those interface standards that directly affect portability for OLTP commercial applications. Today, focus is on supporting the POSIX 1003.1 and POSIX 1003.2 standards because these specifications are the most solidified. As the POSIX committees define specifications for other key operating environment components, such as security, networking, system administration, and transaction processing, these will be adopted by the HP 3000 as appropriate. Both X/Open and POSIX are working to define transaction processing standards, and HP is helping to drive convergence of these standards.

HP 3000 and POSIX 1003.1

As a POSIX core-compliant system, the HP 3000 will support programming interface conventions for:
- process creation and identification
- normal and abnormal termination of processes
- application and process environment information
- process signal handling
- timer operations
- file/directory management
- backup facilities (TAR, CPIO)

The HP 3000 will also be extended to support a hierarchical directory structure as well as a new byte stream file type. The current POSIX standard specification only defines C language bindings. Support is planned in the future to provide other 3GL language bindings, such as COBOL. The HP 3000 will be certified under the P1003.3 certification standard, using the FIPS PCTS conformance tests from the U.S. government (NIST).

Application development guidelines for the POSIX 1003.1 standard will be made available by HP in the future. The POSIX 1003.1 manual can be obtained today through the IEEE Service Center.

POSIX and Unix

POSIX provides a high level of compatibility with Unix because it was derived from ATT's System V Interface Definition (SVID V.3) and Berkeley's Software Distribution (4.3BSD). However, some changes are necessary to 4.3BSD-based and SVID-based applications when they are ported to POSIX 1003.1 because corresponding facilities may not have identical interfaces. The POSIX 1003.1 core also does not include networking and system administration standards, so these may be the primary caveats Unix-based applications will encounter when porting to POSIX. Specifically for 4.3BSD applications, POSIX 1003.1 does not include sockets, NFS, high-precision timers for real-time applications, and highly technical math functions. SVID applications will find that POSIX 1003.1 does not include the STREAMS facility, TLI, and real-time services, such as shared memory and semaphores. (ATT SVID V.4 addresses incompatibilities between SVID V.3 and 4.3BSD.) Overall, application portability from Unix to POSIX should be equivalent to porting between the different varieties of Unix on the market today.

The HP 3000 Provides Performance

POSIX is not oriented toward any particular vendor's hardware, allowing vendors to implement efficient operating system code under the interface specifications. This allows the POSIX implementation on the HP 3000 to be tightly integrated with its underlying, high- performance OLTP operating system benefits, such as significantly reduced disk I/Os, efficient workload prioritization, and dynamic memory management. Additional performance benefits may require taking advantage of the HP 3000 block or field mode terminal processing capabilities.

The goal is to provide an OLTP performance advantage for industry-standard applications on the HP 3000 over other POSIX-compliant systems while maintaining source code compatibility.

The HP 3000 Vision for an Integrated POSIX Environment

A goal of the HP 3000 POSIX environment is 'integration' so that new features are as transparently accessible as possible to MPE users. MPE users will be allowed access to most of the new POSIX functionality, such as hierarchical directory management, via MPE intrinsics and commands. Likewise, POSIX extensions will allow access to MPE files and directories. POSIX applications will be able to access files created by MPE and vice versa, providing compatibility between existing HP 3000 data and new POSIX-compliant data. The HP 3000 will provide MPE system management capabilities for the new environment.

The HP 3000 Provides Additional Value for POSIX

As a result of compromises necessary to achieve consensus on standards, POSIX will typically provide only a subset of the functionality required to meet commercial needs of customers. Thus, it is expected that developers will take advantage of some of the added value features of the HP 3000 in lieu of sacrificing some portability. It is important to note, though, that many of the HP 3000 OLTP benefits can be provided transparently to the end user.

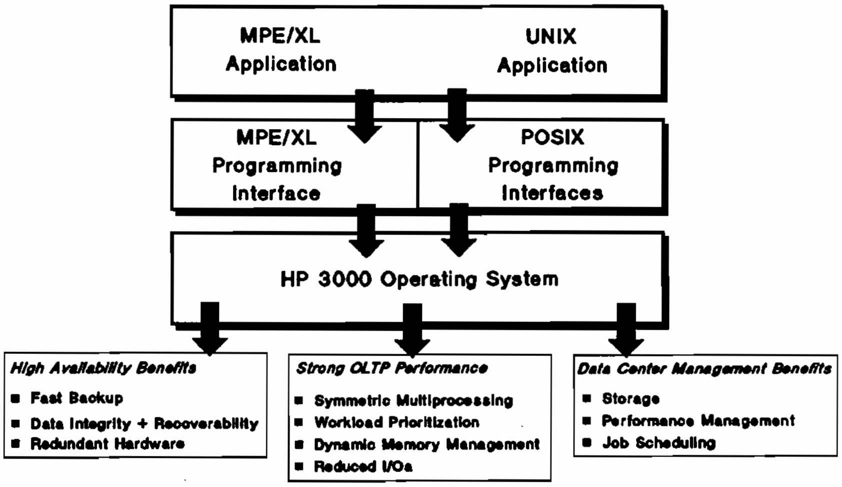

MPE/XL Application
UNIX Application
MPE/XL Programming Interface
POSIX Programming Interfaces
HP 3000 Operating System
High Availability Benefits
■ Fast Backup
■ Data Integrity + Recoverability
■ Redundant Hardware
Strong OLTP Performance
■ Symmetric Multiprocessing
■ Workload Prioritization
■ Dynamic Memory Management
■ Reduced I/Os
Data Center Management Benefits
■ Storage
■ Performance Management
■ Job Scheduling

Figure 3. The HP 3000 POSIX run-time environment
POSIX will sit side-by-side with MPE/XL programming interfaces and will be tightly integrated with operating system services. A Unix application can run on the HP 3000 using the POSIX interfaces and take advantage of HP 3000 features.

3. HP 3000 MULTIVENDOR NETWORKING

The HP 3000 supports a multivendor networking environment through a long-term commitment to OSI standards, as well as ARPA TCP/IP, IBM SNA, and PC LAN connectivity. The HP 3000 also provides gateways between networks to other vendors' systems, such as the DEC VAX. The HP AdvanceNet strategy encompasses meeting a customer's unique mix of networking needs for connectivity, interoperability, and integrated network management.

Developed by the International Standards Organization (ISO), the Open Systems Interconnection (OSI) represents a growing set of networking standards that will be adopted worldwide. TCP/IP and the ARPANet protocol suite is a DoD standard widely implemented by vendors today. The HP 3000 supports these standards today, and customers using ARPA, NS, and TCP/IP today may use OSI in the future for multivendor networking.

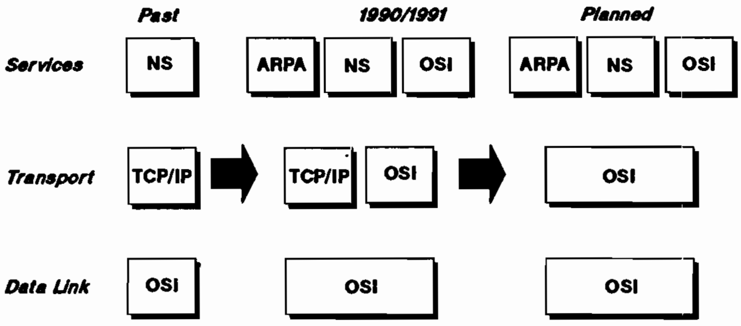


Figure 4. The HP 3000 AdvanceNet strategy
HP's AdvanceNet strategy is to provide solutions that will allow users to build on existing ARPA/NS networks that can evolve to OSI. The solutions will integrate new OSI capabilities with existing applications and network services.

The HP 3000 provides a variety of the basic networking services (electronic mail, file access and transfer, virtual terminal support, network management, and network directory services) under OSI and ARPA. Products based on these services can operate on all OSI physical connections, such as X.25 for wide area networking, IEEE 802.3/Ethernet; and in the future, IEEE 802.5/Token-Ring for local area networking.

Enterprise-wide Networking Capabilities

IBM connectivity and SNA architecture support will continue to be strategic for the HP 3000 and are part of HP's AdvanceNet strategy. The HP 3000 provides:

- a complete family of batch, interactive, and programmatic HP-to-IBM communication products
- advanced interoperability through high-level APIs and office connect products
- support for SAA and selective non-SAA capabilities

A variety of HP 3000 products and services provide interoperability with IBM: LU 6.2 for peer-to-peer networking, Distributed Host Command Facility (DHCF) for IBM 3270 access to HP 3000 applications, and SNA Distributed Systems (SNADS) for HP 3000 communication with IBM's electronic mail protocols. The HP 3000 also offers significant backbone flexibility providing HP-to-HP or HP-to-IBM communications capability over either X.25 or SNA links. SAA Common Communication Services (CCS) interoperability is also provided with HP AdvanceNet products.

PC Client-Server Integration

NewWave Office solutions provide resource sharing, database access, electronic mail, and networked PC management and have placed the HP 3000 in a leadership position for PC integration. HP will maintain this leadership position by providing open server support and application programming interfaces for the key PC LAN networking products that have emerged as de facto standards, such as Novell Netware and LAN Manager. This allows the HP 3000 to tie into existing PC networks for curs who require additional processing capability and want to protect their investment in PC LANs. The HP 3000 also supports Novell clients, virtual terminal HP 3000 access, through NS LAN gateways that allow all NewWave Office capabilities. Similar support is provided for HP LAN Manager clients. In the future, Novell's Portable Netware and LAN Manager application programming interfaces will be supported to enable program-to-program communications between the HP 3000 and PCs over these LANs.

Integrated Network Management

In keeping with HP's AdvanceNet strategy, HP OpenView is fully committed to international and industry standards for network management. Derived from the OSI Management Framework and optimized to include TCP/IP networks, the HP OpenView Architecture was designed as the foundation to manage multivendor network environments. HP's OpenView Windows Developer's Kit allows any vendor to develop network management applications under a common user interface based on standards. The HP 3000 uses OpenView Windows on DOS as the management station for its DTC and system management products and is planning to provide support for the ARPA SNMP and OSI CMIP network management services.

Although much work has been done in defining the communications infrastructure for network management, standards activities are ongoing in the area of networked systems management. The HP 3000 will over time adopt the X/Open definitions for system management.

The HP 3000 also provides the ability to coexist and be managed by IBM network management services. SNA/Netview Alerts are provided to allow HP 3000s to signal IBM Netview operators in the event of a link failure.

A Network Foundation for Distributed Applications

HP encompasses a combination of the OSI, ARPA, LAN Manager, Novell, and IBM SNA standards to provide services needed to build distributed applications. These include:

- Process-to-process communications. Currently provided with support of the SAA- and X/Open-adopted LU6.2 APPC protocol. The HP 3000 will also make available other process communication standards, such as remote procedure call capabilities through support of HP/Apollo's Network Computing System (NCS), LAN Manager Named Pipes, Novell's SPX/IPX, and ARPA Berkeley Sockets.
- Store-and-forward services for electronic mail. Provided through support of the OSI X.400 standard and support of IBM's SNA Distribution Services (SNADS).

- Built-in communication services. This includes products available to allow networked user interfaces, remote database access, transparent file sharing, and shared office services.
- Standard data streams. The HP 3000 will provide products built on ANSI, SNA, and ISO standards for document and graphics formats.

The Open Software Foundation is planning to announce their Distributed Computing Environment (DCE) technology in 1990. OSF's DCE includes remote procedure calls, naming authentication and directory services, and a distributed file system technology. The HP 3000 will support components of OSF's DCE in the future.

4. THE HP 3000 AND USER INTERFACES

Support of open window-based user interfaces is a key interoperability feature for open systems of the future. An application's graphiaA interface (GUI) makes it possible for applications running on different platforms to present a common "look and feel" to the end user, allowing users to move between systems and applications with ease. Thus, GUIs offer reduced training time for new applications and improved user productivity.

Different user interface standards have emerged for each of the major desktop systems. Therefore, no one standard satisfies all of the system requirements for a standard user interface. For example, the most widely known graphical user interfaces are Microsoft® Windows on MS-DOS, Presentation Manager (PM) on OS/2 systems, and the Apple Macintosh -- each with significantly different application programming interfaces.

In the absence of a single, standard user interface API, the application developer must consider the desktop devices to b pported. The HP 3000 provides application developers a wide variety of highly portable user interface alternatives for PCs, X-displays, and terminals, each providing a different level of interoperability, portability, and performance. Similar to functionality provided by several 4GLs on the marketcaay, HP plans to provide services in the future that will mask the technology differences between user interface APIs such as X Windows, PM, MS Windows, and VPLUS.

PC-based GUIs: NewWave, MS Windows, Presentation Manager

HP's NewWave provides an object-oriented application programming environment. NewWave is not a graphical user interface but derives its "look and feel" from the underlying GUI. It is currently supported under MS Windows with plans for support under Presentation Manager and OSF/Motif™. There are currently no formally adopted standards for object management, but HP is leading these efforts with NewWave's Object Management Facility. Several vendors such as Data General, NCR, and AT&T have licensed NewWave.

PC user interface packages based on NewWave, MS Windows or Presentation Manager are consistent with HP's cooperative computing strategy and provide the user portability benefits associated with the common "look and feel" of these GUIs. Oracle, Ingres, Sybase, and HP ALLBASE will also provide PC-based application development forms packages that provide a high degree of portability across platforms.

VPLUS/Windows is an alternative for existing VPLUS customers who desire a smooth migration path to a graphical windows-based user interface on PC platforms under a more common GUI.

X-based GUIs: X Windows, HP OSF/Motif

X Windows has been adopted by X/Open as the industry- standard protocol for implementing a distributed, graphical user interface. OSF/Motif is the graphical user interface built on top of X Windows that is provided by HP. The X Window System allows integration of applications that run on different vendor systems in a network to be displayed simultaneously on an X-display device. The HP 3000 will support X Windows and OSF/Motif by 1990.

The OSF/Motif environment provides behavior consistent with PC-based de facto standards, along with an enhanced 3D appearance. In manufacturing arenas, HP 3000 planning and control applications can also now share an integrated "look and feel" with HP 9000 or workstation-based X Windows applications managing the factory floor.

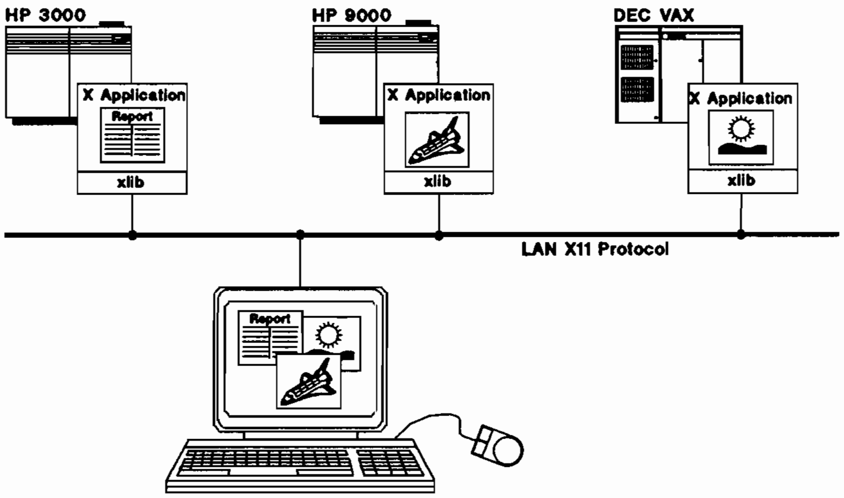


Figure 5. The HP 3000 and OSF/Motif
The HP 3000 provides a common user interface for multivendor network environments through support of X Windows and OSF/Motif.

Terminal-based interfaces: 4GLs, JAM, and VPLUS

HP offers primarily two portability alternatives for terminal-based user interfaces: the use of popular 4GL application development environments and a user interface management system JAM (J tplication Manager). Ingres, Sybase, Oracle, and Cognos all offer 4GL forms packages that isolate the application developer from terminal interface processing. The key benefit of using these packages is a consistency of behavior for applications across different platforms and their windows functionality.

JAM also provides windows-like functionality and is a powerful user interface prototyper available on the HP 3000 today. JAM applications offer interoperability between the HP 3000, HP-UX, and MS-DOS and runs on over 70 Unix platforms. JAM is recommended for customers who have decided to develop applications in C or COBOL II and intend to create stand-alone applications that are not tied to a particular DBMS.

Although not an industry standard, VPLUS will continue to be the best user interface alternative for OLTP performance.

HOW TO GET STARTED TODAY

Programming for portability guidelines on the HP 3000 follow closely those standards embraced by X/Open's CAE. Outlined below is a specific approach developers should consider when creating open systems applications for the HP 3000 environment. These programming for portability "tips" simply highlight a few guidelines application developers should consider because thoroughly explained guidelines would require a separate paper.

What about MPE/V?

Customers need to upgrade to the PA-RISC architecture to take advantage of open systems products and programming interfaces.

Modularized code

The framework for the HP 3000 Open OLTP Environment describes the key application programming interfaces required. Applications must be modularized so that programming interfaces for database management, user interface screen handling, networking, and the operating system are separated and can evolve easily to use new standard interfaces as they become available on the HP 3000.

Traditional OLTP versus client-server

New application development must consider whether the application will solely support traditional terminal-host configurations or client-server configurations as well. Either a 3GL or 4GL development environment is recommended for traditional terminal-host applications. A 4GL development environment is recommended for client-server applications because most popular

4GL tools on the HP 3000 today accommodate a variety of display devices, supporting both terminal-host processing and interfaces for PC-based distributed applications.

SQL database options

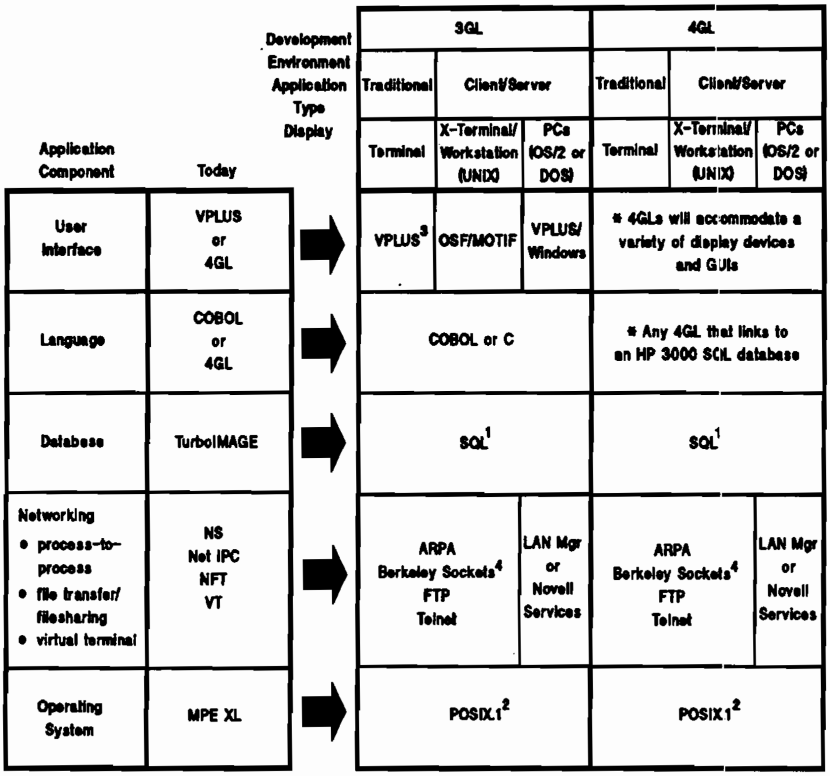
Application development on a standard SQL relational database management system is necessary for maximum application portability. HP 3000 customers can choose ALLBASE/SQL or any of the multivendor ISV database solutions available on the platform, such as Oracle, Ingres, or Sybase.

MPE programming interfaces and POSIX 1003.1

The transition to POSIX from MPE should be carefully planned. Application operating system calls should be modularized, and a developer should evaluate functionality and performance trade-offs. Note that many applications today are independent of direct operating system calls.

The future of VPLUS

Many HP 3000 customers have written their applications in VPLUS to capitalize on its block mode high-performance features. Although not a standard, VPLUS will continue to be the premium alternative for OLTP performance on the HP 3000. VPLUS/Windows provides a migration path to a more state-of-the-art user interface on MS-DOS PCs and supports the NewWave environment. VPLUS is recommended when the primary design consideration is performance. OSF/Motif is recommended for 3GL applications when the primary design consideration is standardization. It is important to note that 4GL forms packages will provide the most flexibility for portability across systems and various desktops.



Application Component	Today	Development Environment	3GL			4GL		
		Application Type	Traditional	Client/Server		Traditional	Client/Server	
		Display	Terminal	X-Terminal/ Workstation (UNIX)	PCs (OS/2 or DOS)	Terminal	X-Terminal/ Workstation (UNIX)	PCs (OS/2 or DOS)
User Interface	VPLUS or 4GL	➡	VPLUS[3]	OSF/MOTIF	VPLUS/ Windows	* 4GLs will accommodate a variety of display devices and GUIs		
Language	COBOL or 4GL	➡	COBOL or C			* Any 4GL that links to an HP 3000 SQL database		
Database	TurboIMAGE	➡	SQL[1]			SQL[1]		
Networking • process-to-process • file transfer/ filesharing • virtual terminal	NS Net IPC NFT VT	➡	ARPA Berkeley Sockets[4] FTP Telnet		LAN Mgr or Novell Services	ARPA Berkeley Sockets[4] FTP Telnet		LAN Mgr or Novell Services
Operating System	MPE XL	➡	POSIX.1[2]			POSIX.1[2]		

Figure 7. HP 3000 basic portability options
1. Any HP 3000 SQL database can be used. ALLBASE/TurboCONNECT for TurboIMAGE to SQL transition; 2. Initial support for C language bindings only. Note many applications independent of operating system calls; 3. VPLUS for optimal OLTP performance; 4. NetIPC provides Berkeley Sockets interoperability.

SUMMARY

The HP 3000 strategy is to support relevant commercial open system standards to provide application portability and multivendor interoperability while providing industry-leading OLTP performance and functionality and protection of customers' investments. The strategy encompasses support for de facto market standards, X/Open, and IBM SAA interoperability and provides key application programming interface standards for all components of a software computing environment. In addition, the HP 3000 supports a long-term commitment to OSI (HP AdvanceNet), industry-wide database interface definitions (SQL Access Group), standard graphical user interfaces (for example, PM, OSF/Motif), industry-leading object management (NewWave Computing), and standard network management protocols (HP OpenView). Commitments to both standards and industry-leading OLTP will make the HP 3000 one of the premier open platforms of the 1990s.

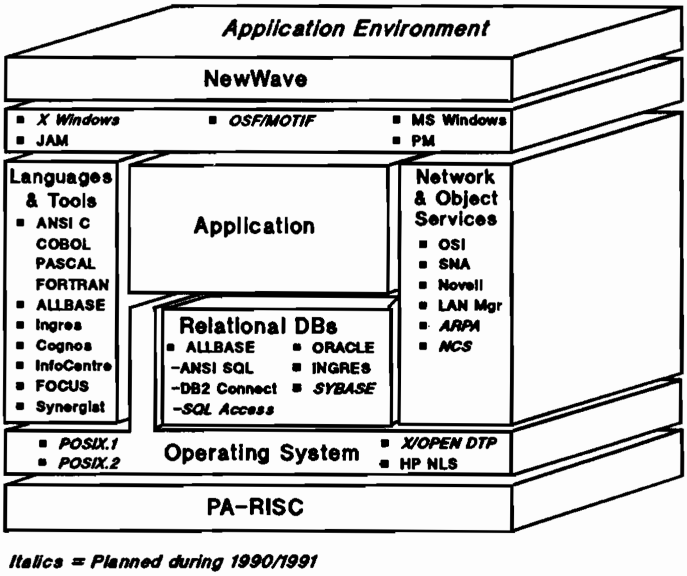


Figure 6. The HP 3000 Open OLTP Environment
The HP 3000 supports key products for an open application environment, providing the foundation for HP's NewWave Computing strategy.

Hewlett-Packard Fiber-Optic Link
A Performance Growth Path for MPE XL Systems

Melody Armstrong
HEWLETT-PACKARD COMPANY
P.O. BOX 39
BOISE, ID 83707

Abstract

This paper discusses the performance contribution HP-FL makes to MPE XL performance. This includes an in-depth look at the specific features implemented within HP-FL to optimize performance, and the impact they have on disk service time and disk throughput. The relationship between HP-FL and overall system performance is also discussed, including current and anticipated MPE XL disk I/O requirements.

Introduction

The Hewlett-Packard Fiber-Optic Link, referred to as HP-FL, is a disk interface specifically designed for Precision Architecture Systems. HP-FL is based on fiber-optic technology and transmits data between the SPU and the disk drives via light pulses. With the advent of HP-FL, HP 3000 MPE XL systems now have an attractive alternative to the traditional HP-IB disk interface (Hewlett-Packard Interface Bus).

The advantages of HP-FL relative to HP-IB are numerous:

o Up to eight HP-FL disks can be placed on a single HP-FL interface card, while HP-IB is limited to six HP-IB disks per HP-IB interface card. This means larger disk configurations can be achieved with HP-FL using fewer SPU I/O slots.

o HP-FL supports fiber-optic cable lengths up to 500 meters, while HP-IB supports a maximum cable length of 15 meters. This allows HP-FL a higher degree of configuration flexibility because disks can be placed further away from the SPU.

o The fiber-optic cable is immune to electromagnetic interference and does not emit radio frequency energy that might cause interference with other equipment.

o HP-FL offers an improved data transfer rate relative to HP-IB, 5 megabytes per second versus 1 megabyte per second, respectively. In conjunction with other performance enhancements, this increases HP-FL's performance potential.

o The flexible HP-FL design provides new configuration opportunities and additional functionality. This enables HP-FL to offer a growth path for the future and a platform for future mass storage solutions.

o HP-FL's unique capabilities are exploited through MPE XL's new high availability solutions including disk mirroring, on-line backup and SPU switchover, which are based solely on the HP-FL platform.

While many of the advantages of HP-FL are well known, one area that is not widely understood is the relationship between HP-FL and performance. HP-FL has been designed and tuned to meet the current and future performance needs of the MPE XL systems. As part of this tuning a number of enhancements have been incorporated within HP-FL including a distinct method of managing channel utilization, transaction pipelining, command queuing and seek reordering. These enhancements complement the increased data transfer rate capabilities to maximize performance.

Channel Utilization

The channel is defined as the communication path between the system and the disk. The host and disks interact frequently during the processing of a disk transaction to transfer commands, user data and status reports. The method in which this interaction is accomplished has a big impact on performance. This is especially true in multiple disk configurations where several disks may require use of the channel simultaneously.

Early HP-IB disk implementations did not always offer the greatest efficiency in relation to channel utilization. As a result, several enhancements have been implemented over the years to optimize disk performance in association with channel utilization. Among these enhancements were buffer prefill, rotational position sensing and data transfer during seek on a write. These improvements optimized the interaction between the disks and the channel and allowed for greater efficiency in multiple disk configurations.

In order to meet MPE XL needs, including efficient handling of large transfers, a new channel utilization method has been developed for HP-FL. This method incorporates resource management techniques that allow the disk and the channel to work independently of one another. More specifically, HP-FL ensures that data will be available to transfer by the time the channel is acquired, and negotiates with the destination device to ensure the necessary resources are committed prior to moving data. Transfers that exceed the size of available resources are broken into multiple data request/transmission blocks. This channel management method ensures that the transfer rate of the disk at approximately 1.4 to 1.8 megabytes per second does not limit the 5 megabyte per second data transfer rate of the channel.

Multiple Data Blocks

With the current implementation of HP-FL, transfers that exceed the size of the disk's 32 kbyte internal buffer are broken into multiple data request/transmission blocks. Thus, the disk is always capable of buffering a complete data block regardless of the total size of the transfer. If necessary, the disk's internal buffer is capable of managing two data request/transmission blocks simultaneously. That is, data associated with one block can be transferred from the buffer while data associated with another block is being accepted into the buffer. This prevents delays from occurring when multiple data blocks are needed.

Figure 1 illustrates the multiple data block concept employed by HP-FL.

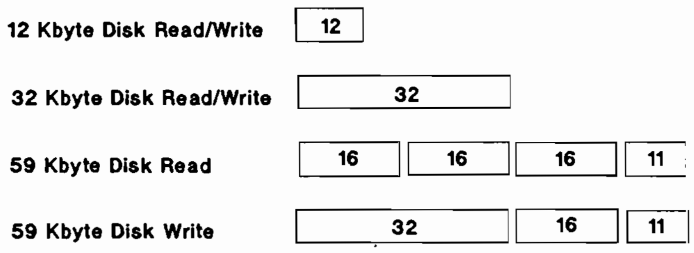


Figure 1. HP-FL Multiple Data Block Example

Disk reads in excess of 32 kbytes are separated into multiple 16 kbyte data blocks. Disk writes larger than 32 kbytes are broken into one 32 kbyte data block and subsequent 16 kbyte data blocks. Transfers that are equal to or less than 32 kbytes are transferred as a single data block.

The size of the data request/transmission blocks is hardware dependent and could change with future HP-FL implementations. The goal is to use block sizes that allow the disk to stay busy, while minimizing the overhead on the link. These goals are currently being met with the existing HP-FL implementation.

If it is necessary to break a data transfer into multiple data blocks, the channel is not held throughout the entire transfer. Instead, the channel is released between each data block transmission. This means that each data block is treated as an independent transfer. As a result, it is possible for other disks or the host to acquire use of the channel between data block transmissions.

Although it may seem inefficient to allow a single data transfer to be interrupted by another device, this scheme actually results in the fairest possible sharing of the channel by all devices. As a single entity each device may not always achieve the highest level of efficiency, but overall the efficiency of the disk network is improved.

The actual allocation of the channel is managed by hardware within the disk controller. For each group of disks attached to a system interface card, one disk is designated as the channel manager. This is determined by the device address and is typically device 0. A round robin priority scheme is used to allocate channel resources. This ensures that no host or device is starved.

Benefits

Figure 2 illustrates the benefit HP-FL's unique channel utilization method provides in the form of transaction time, which is comprised of controller overhead, seek, latency and data transfer time (data transfer time is the greater of channel time and disk transfer time). The graph compares the HP Series 6000 Models 670FL and 1.34FL to the HP Series 6000 Model 670H while performing a 1 kbyte transfer and a 64 kbyte transfer (assuming average random seek and latency).

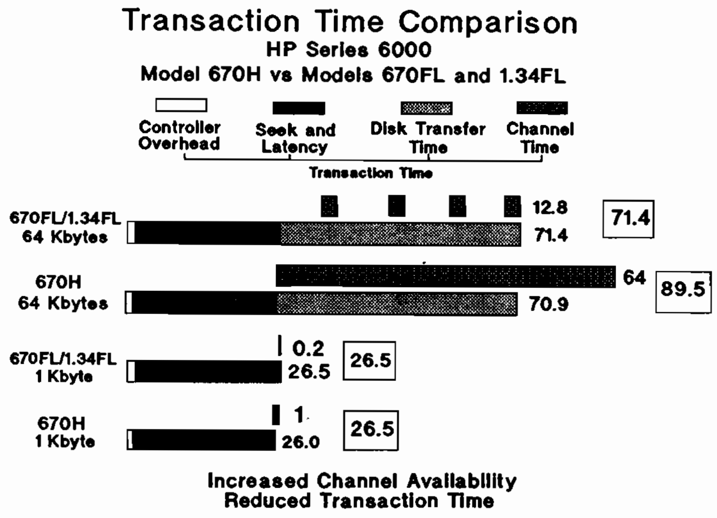


Figure 2. Transaction Time Comparison

One of the most noticeable differences between HP-IB and HP-FL is the usage of the channel during the data transfer. With HP-IB, the channel is requested .9 milliseconds before the target address is reached. Data is read from disk and transferred across the channel via the disk's internal buffer. The channel is held throughout the entire transfer regardless of total transfer size. If the transfer rate of the disk is slower than the channel, the channel may be forced to wait for the disk. Likewise, if the transfer rate of the channel is slower than the disk, the disk may be forced to wait for the channel, especially if the transfer exceeds the size of the disk's internal buffer. This means the maximum transfer rate is determined by the slowest entity, which in this example is HP-IB at 1 megabyte per second.

In comparison, the HP-FL disk ensures a full data block will be ready to transfer by the time the channel is acquired, so data can burst across the channel at the 5 megabyte per second data transfer rate. It accomplishes this by breaking large transfers that exceed the size of the drive's internal buffer (i.e. 64 kbytes) into multiple data request/transmission blocks, which are buffered separately. While a data block is being buffered, the controller calculates how much time is required to complete the necessary resource negotiation with the host and how long it will take until the last byte for that block of data is read from disk. At the optimal time, the disk negotiates with the host for required resources and bursts the buffered data block across the channel at 5 megabytes per second. As soon as the data block is transferred, the channel is released. The process is repeated for subsequent data blocks. Since the drive's internal buffer is capable of managing two separate data blocks simultaneously, the disk can begin buffering a succeeding data block while the initial data block is being transferred.

Although data is transferred across the HP-FL channel at 5 megabytes per second, the time required to read data from disk must be factored into the total transfer time. This means that total transfer time is dependent upon the speed of the disk, which is typically 1.4 to 1.8 megabytes per second. Based on a transfer rate of 1.4 megabytes per second, it takes approximately 45.5 milliseconds to transfer 64 kbytes with HP-FL and only 12.8 milliseconds of this is channel time. It takes approximately 64 milliseconds to complete the same transfer with HP-IB and a full 64 milliseconds of channel time is required. Not only does it take less time to complete the data transfer with HP-FL, but the availability of the channel is increased as well.

The exact transaction time savings associated with HP-FL varies from transaction to transaction and is dependent upon the length of the transfer as well as the transaction type. Typically, reads experience a higher savings than writes. This is because of the manner in which data is transferred during writes. Both HP-FL and HP-IB initiate the data transfer for a write as the seek begins. Depending upon the size of the transfer and the length of the seek and latency period, HP-IB may have enough time to buffer an adequate amount of data so the disk does not run out of data once the write begins. In such cases, total transaction time is more comparable to that of HP-FL. If there is not enough time to buffer an adequate amount of data, the HP-IB disk may be forced to wait for the channel. In order to compensate for these delays, the disk induces latencies which in turn increase total transaction time. In these situations, HP-FL will have some transfer time advantage. Either way, there is more channel time consumed with HP-IB than with HP-FL.

Transaction Pipelining

HP-FL disks also provide transaction pipelining , which is a performance enhancement feature that maximizes disk throughput in I/O intensive environments by overlapping transaction processing. In other words, one transaction can begin before the previous one has finished. This offers an advantage relative to HP-IB where only one transaction can be processed at a time per drive.

A single HP-FL disk is capable of simultaneously managing multiple transactions. The disk controller has the ability to buffer a maximum of 14 disk commands at a single time (command queuing). This means up to 14 commands can be progressing through the command decode process at one time. Furthermore, two transactions can be in the execution/report phase simultaneously. This allows the disk actuator to be logically separated from the I/O channel, which permits the actuator to be dispatched to the next target address regardless of the delays associated with the channel. This enables transactions to be continually fed to the disk, thus reducing disk idle time and maximizing disk throughput especially during peak I/O periods.

Transaction Overlap

A typical disk transaction is comprised of three stages: the command phase, the execution phase and the report phase. The command phase is that portion of the disk transaction in which the disk controller receives and decodes a command. As soon as this is completed, the disk transaction enters the execution phase. During this stage, the disk mechanism performs the requested operation by mechanically positioning the heads over the designated location (seek and latency) and transferring the data. As soon as the transaction has been executed, it enters the report phase. During this phase, the disk controller conducts necessary cleanup activities and issues a status report to the CPU indicating successful completion of the transaction.

The HP-FL disks overlap transactions between command and execution phase and between report and execution phase. In addition, two transactions can overlap during execution phase as long as the total transfer size of each transaction does not exceed 16 kbytes. Transaction overlap allows for the masking of controller overhead during command decode and report phase. It also minimizes the effect of a busy channel on transaction throughput. Following is a description of how transaction overlap is achieved during the various phases.

Execution/Command Overlap

While the disk mechanism is executing one command, the controller can accept and decode other commands. As commands are received, they are queued and decoded one at a time in order of receipt. The decoded commands remain in the command queue until the disk mechanism is finished with the previous transaction. As soon as the mechanism is available, the controller is ready to immediately launch the next request. This enables the controller overhead associated with command decode for a transaction to be masked by the execution activities of the previous transaction.

Execution/Execution Overlap

It is possible for two transactions to be in execution phase at the same time. Naturally, the disk actuator cannot perform the seek and latency for two transactions simultaneously. Similarly, it is not possible for data to be transferred across the channel for two transactions at the same time. However, one transaction can be using the disk mechanism while another is using the channel to transfer data. This is only possible if the transfer size of each transaction is 16 kbytes or less. This is necessary to ensure that disk buffer resources can be fully committed for each transfer.

Execution/Report Overlap

While the disk controller is preparing to send the completion report for a disk transaction, the disk mechanism can begin executing the next command. This allows the controller overhead associated with report status to be masked by the execution activities of the succeeding transaction.

MPE XL Implementation

Although the HP-FL disks have the capability of simultaneously overlapping multiple disk transactions, the MPE XL operating system limits the transaction pipeline to a depth of two. This means a maximum of two disk transactions overlap at a given time per drive. The system maintains this control based on the number of commands queued within the disk. If the disk has a queue depth of two, the MPE XL driver does not initiate another disk command until the completion report for the previous transaction is received and the queue depth falls to one.

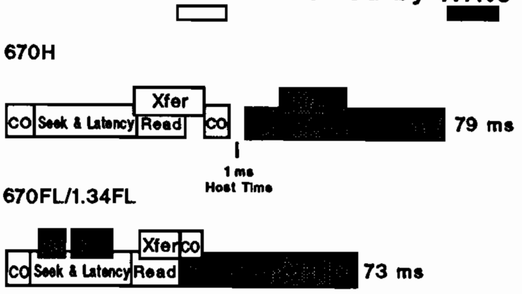
The MPE XL operating system limits the pipeline level in order to ensure that disk I/Os are conducted in the proper sequence. The nature of MPE XL applications is such that disk writes associated with a specific transaction must take place in the correct order to ensure file consistency. Although it may appear as though MPE XL is not getting maximum benefit from transaction pipelining, the largest incremental benefit is achieved when going from a pipeline of one transaction to two. Increasing the pipeline past two can offer some incremental benefit, but it is less than that realized by going from one to two.

Benefits

Figure 3 illustrates the benefit of transaction pipelining when a read of 16 kbytes is immediately followed by a write of 16 kbytes. This example provides a comparison between the HP Series 6000 Model 670H and the HP Series 6000 Models 670FL and 1.34FL.

HP Series 6000
Model 670H vs Models 670FL and 1.34FL

Read Followed by Write



16 Kbyte Transfer Size

NOT TO SCALE

Figure 3. Transaction Pipelining Benefit

Unlike HP-IB, the HP-FL disk can accept and decode the write command while the disk mechanism is performing the seek and latency for the read (execution/ command overlap). Although mechanical positioning cannot be initiated for the write while the disk mechanism is executing the read, the write data can be transferred to the disk's internal buffer (execution/execution overlap). Since the total transfer size of each transaction is 16 kbytes, there is adequate buffer space to accommodate both transactions. As soon as the disk mechanism is available, the write command is immediately launched. While the mechanism is busy performing the seek and latency for the write, the disk controller prepares and issues the completion report for the read (execution/report overlap).

It takes the Model 670H approximately 79 milliseconds to complete the two transactions, whereas it takes the Model 670FL/1.34FL approximately 73 milliseconds. Transaction pipelining contributes to this time savings by allowing controller overhead and host turnaround time to be masked. HP-FL's ability to transfer data at a faster rate than HP-IB also provides some time savings.

The exact amount of time saved with transaction pipelining is dependent upon a number of factors including the sequence of transactions, the timing of I/Os and especially the availability of the channel. The previous example is representative of an environment in which channel availability is ideal. With multiple drives sharing the same channel, there is a higher probability of experiencing channel contention. In such instances, the benefit of transaction pipelining is more noticeable.

Figure 4 illustrates the benefit of transaction pipelining when channel delays are experienced. As in the previous example, a comparison is shown between the Model 670H and the Model 670FL/1.34FL when a read of 16 kbytes is immediately followed by a write of 16 kbytes (assuming average random seek for both transactions).

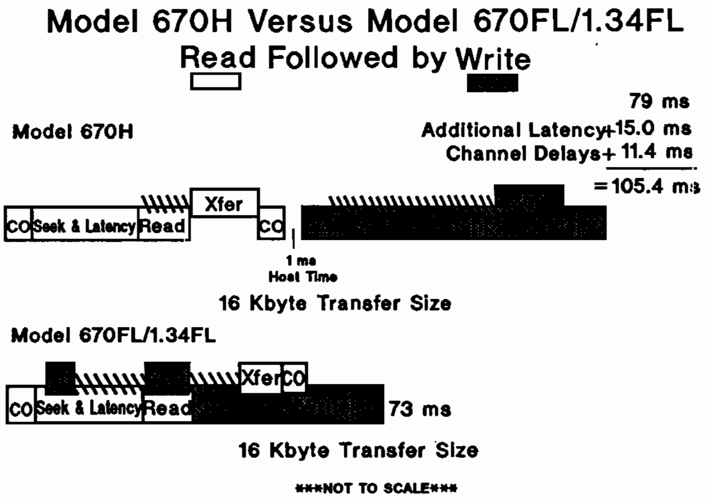


Figure 4. Transaction Pipelining Benefit with Channel Delays

If channel delays are encountered with HP-FL, there is typically enough time for those activities that require use of the channel (the command transfer, the data transfer and the completion report) to take place later in the transaction process without impacting total transaction time. HP-IB does not offer this flexibility, because disk resources are tied to the channel. Therefore, there is a higher likelihood that channel delays will increase transaction time with HP-IB.

When compared to the previous example, the channel delays experienced with HP-IB increase total transaction time by greater than 25 milliseconds, while total transaction time is unchanged for HP-FL. Although this example is very unfavorable toward HP-IB, it illustrates the benefit of HP-FL in multiple disk configurations.

Naturally, HP-FL is not completely immune to channel contention. If channel delays become excessive, the balance between channel and disk resources may be disrupted. When this occurs, the transaction pipeline becomes backed up and throughput is negatively impacted. However, the chances of this occurring with the present HP-FL implementation are negligible. The round robin channel allocation technique used by HP-FL ensures that channel resources are shared in the fairest possible manner. In addition, the transaction pipelining process has been specifically tuned for Precision Architecture systems to provide the highest level of efficiency, especially during I/O intensive periods.

Seek Reordering

HP-FL drives also have the ability to reorder "Locate and Read" and "Locate and Write" commands. If multiple commands are queued in the command buffer, the disk controller reorders these commands so they execute in the most efficient order. The reordering scheme is based on seek distance and the length of time a command has waited in the queue. This method minimizes seek overhead associated with high traffic environments while ensuring no command is neglected. As a result, seek reordering helps level disk response time and increase disk throughput for burst activity.

The current MPE XL systems do not take advantage of seek reordering. The MPE XL operating system maintains control of the I/O sequence by limiting the pipeline depth to two. Thus, there is no opportunity for seek reordering at this time. However, seek reordering can make an important contribution in multi-host configurations where multiple systems pipeline transactions to each disk. In this environment, seek reordering will minimize seek time and increase the potential throughput of each drive.

HP-FL and Physical Disk Performance

Hewlett-Packard uses the metric of I/Os per second to measure disk performance. I/Os per second is defined as the maximum number of disk transactions per second that a specific disk can perform at a given transfer size. Table 1 shows the disk transaction and I/Os per second figures for the Model 670H and the Model 670FL/1.34FL. A transfer size of 12 kbytes is shown because this is representative of the average MPE XL disk I/O. Since MPE XL also performs a number of very large transfers, a transfer size of 64 kbytes is also shown.

	12 kbytes		64 kbytes	
	Model 670H	Model 670FL/ 1.34FL	Model 670H	Model 670FL/ 1.34FL
Controller Overhead	1.1 ms	1.6 ms	1.1 ms	1.6 ms
Seek	17.0 ms	17.0 ms	17.0 ms	17.0 ms
Latency	7.5 ms	7.5 ms	7.5 ms	7.5 ms
Data Transfer Time (Based on 1.0 Mbytes/s Across HP-IB)	12.0 ms	-	64.0 ms	-
Data Transfer Time (Based on 1.4 Mbytes/s on HP 7937)	-	8.5 ms	-	45.5 ms
Total Access Time	37.6 ms	34.6 ms	89.6 ms	71.6 ms
I/Os Per Second (No Pipeline)	26.6	28.9	11.2	13.9
I/Os Per Second (Full Pipeline)	N/A	30.3	N/A	14.3

Table 1. Model 670H and Model 670FL/1.34FL Disk Read Throughput Figures

There is slightly more controller overhead associated with the HP-FL due to increased functionality. However, the ability to pipeline transactions effectively masks controller overhead on busy disks. The seek and latency time of the Model 670/1.34FL is identical to that of the Model 670H, because the same disk mechanism is used in both. The biggest contribution the fiber-optic link makes to physical disk performance is the reduction in data transfer time. This is because data transfer time is determined by the speed of the disk. As a result, a typical 12 kbyte MPE XL disk I/O executes in approximately 8 - 12% less time with HP-FL, depending upon whether the pipeline is full or not. This increases disk throughput by approximately 2 to 3 I/Os per second. A 64 kbyte disk I/O executes in approximately 20% - 22% less time with HP-FL and disk throughput is increased by nearly 3 I/Os per second.

It is important to keep in mind that disk transaction time and I/Os per second are used to compare the relative performance of one disk to another. These numbers represent fundamental disk performance without taking into consideration specific system attributes. Therefore, the benefit HP-FL has on system level performance cannot be extrapolated from these numbers.

HP-FL and MPE XL System Performance

The degree to which disk performance influences system level performance is determined in large part by the demands of the operating system and the user applications. Over the years, the disk I/O requirements of the classic MPE systems have evolved. Initially, these systems did not require a substantial amount of disk I/O. Therefore, CPU was often the performance bottleneck. As the hardware and software matured and the user and application base grew, disk I/O demands increased. Today, disk I/O is frequently a performance bottleneck for many classic MPE systems.

The knowledge gained from the classic MPE systems was incorporated into MPE XL operating system development. As a result, one of the MPE XL design goals was to improve system level performance by reducing disk I/O requirements. MPE XL accomplishes this through the use of mapped files, dynamic reads (prefetch algorithms), delayed and gathered writes, and a feature known as transaction management. These features take advantage of the large memory configurations available with HP-PA systems to effectively manage large amounts of data within main memory. As a result, MPE XL disk I/O characteristics differ considerably from those of MPE V. Not only does MPE XL require fewer disk I/Os to complete the same task, but disk I/Os tend to be larger, more random and more "bursty" in nature.

Because of the reduced disk I/O requirements of MPE XL, the impact HP-FL has on the system level performance of today's typical MPE XL systems is small. HP-FL has exhibited a 0-8% increase in system throughput over HP-IB in Series 950 benchmarks. However, the MPE XL systems are still in their infancy. As larger workloads are supported through the availability of faster processors and operating system tuning, disk I/O demands will increase. It is anticipated that increases will be seen in I/O size and well as I/O traffic.

To better illustrate the full performance potential of HP-FL, Figure 5 provides a comparison between the raw I/O capability of a single HP-IB drive and a single HP-FL drive. The data are representative of raw disk performance in an environment in which I/Os are generated at a rapid rate with very little system overhead.

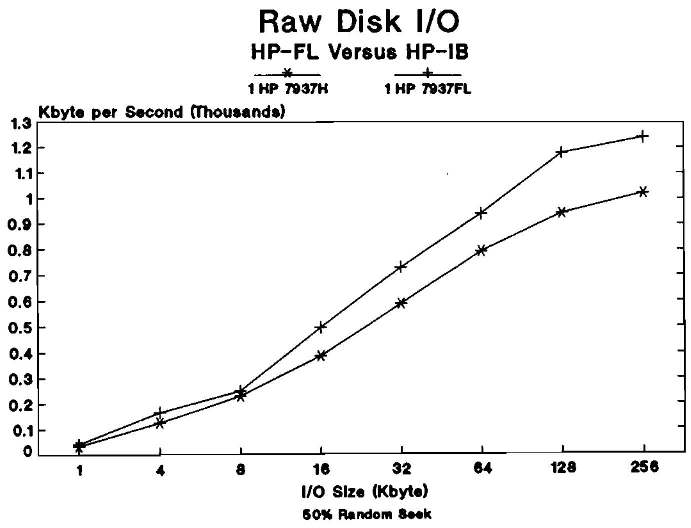


Figure 5. Raw Disk I/O - HP-FL vs HP-IB

It is evident that HP-FL offers a substantial performance advantage as the transfer size increases. At a transfer size of 8 kbytes and less there is only a small performance advantage over HP-IB. However, as the transfer size becomes larger, HP-FL outperforms HP-IB by a wider margin. This indicates that the biggest performance advantage with HP-FL is in environments where a substantial amount of disk I/O takes place and the majority of data transfers are large (greater than 8 kbytes).

Although Figure 5 gives a better indication of the performance potential of HP-FL, it is still a non-optimal view because the impact of better channel utilization is not fully represented. As more disks are placed on a single channel, the difference between HP-IB and HP-FL becomes even more dramatic. The HP-IB channel is already approaching its maximum burst rate while the HP-FL channel has 75% of its bandwidth remaining. This additional bandwidth will allow larger disk subsystems to increase their performance edge.

Summary

The Hewlett-Packard Fiber-Optic Link has been designed and tuned to meet the growing performance needs of the MPE XL systems. All of the enhancements incorporated into the interface including increased transfer bandwidth, efficient channel utilization techniques, transaction pipelining and seek reordering work together to optimize performance. The unique characteristics of HP-FL make it an excellent solution for I/O intensive applications that perform large transfers.

The impact HP-FL has on the system level performance of today's MPE XL systems is typically small. This is not because of HP-FL performance limitations, but rather the excellent job MPE XL does reducing disk I/O. As the MPE XL systems evolve and disk I/O demands increase, the need for HP-FL will increase and the benefits will become more apparent. In combination with the numerous other advantages including large disk configurations, long cable lengths, few environmental concerns, new configuration opportunities and high availability solutions, HP-FL is well positioned to meet the increasing storage needs of the MPE XL systems.

Design, Installation, and Maintenance of Cabling Infrastructures

Charles Breed
Hewlett-Packard
8000 Foothills Blvd.
Roseville, CA 95678

INTRODUCTION

Many businesses have specific networking requirements that may require different wiring solutions. Tailoring a scalable and industry-standard cabling strategy can be difficult. The media (or transmission media) is the physical wiring over which voice, data and video signals are transmitted. Various types of transmission media are available, each having its own information carrying capacity (bandwidth) and suitable application. Some basic guidelines for choosing the best cabling media for a particular network topology will be covered. The key evaluation criteria are price/performance tradeoffs, flexibility, expandability, administration, and environmental requirements.

In the past there has been much controversy surrounding the selection of a particular data networking technology. Today, the importance of networking is better understood and users are planning networks that will more effectively support their data communication needs. These networks, along with your physical environments, will have significant influence on the choice of appropriate media. The cabling systems are being pushed to their performance limits as digital voice, data, and video merge in today's communications environment. The resulting larger and more complex networks require extensive planning.

Understanding the different types of transmission media and knowing their restrictions help in the implementation of new networks. There are three predominant types of physical cable which voice, data, and video are transmitted. Coaxial, twisted-pair, and fiber-optic cable each have their own information carrying capacity and suitable application.

The increasing popularity of 10-Mbit/s unshielded twisted-pair LANs has driven the creation of the IEEE Type 10BASE-T standard. The benefits of UTP (unshielded twisted-pair) include reduced cost of ownership and lower cost to install. The fundamentals of Type 10BASE-T are derived from the telephone industry's standard wiring system. A.T.& T.'s Premises Distribution System (PDS) is compatible with the IEEE Type 10BASE-T standard. Most phone systems today use a 24 AWG, UTP type of cable, the same as Type 10BASE-T. The Type 10BASE-T standard also allows for 22 and 26 gauge cable, which are less common.

Coaxial cable has been a data communications media of choice for many years and is common in today's LAN environment. Coaxial cable can be used for either broadband or baseband networks. Coax consists of four basic parts, which can be composed of various combinations of materials. The inner conductor, which carries the signal, can be either solid copper or copper-coated aluminum. The inner conductor is protected by an insulating plastic coating. The outer conductor is composed of braided aluminum wire or laminated foil tapes. A PVC vinyl or teflon sheath then covers and protects the overall cable from physical damage.

Various forms and combinations of inner conductors, insulators, shields, and coatings are used for different connections. The heaviest coaxial cables are most often used to provide the miles-long backbone, or trunk, of a network. Branching off the main backbone are distribution cables that are usually smaller, more flexible, and provide a secondary backbone for floor to floor interconnection within a building. Tapping off the distribution cables are the drop cables that connect the network to stations, for example computers, printers, and controllers. These flexible drop cables are kept between 10 and 100 feet in length because of their lesser shielding capability.

Fiber-optic cable is a third alternative transmission media for local area networks. The cable consists of thin strands of transparent and translucent glass through which modulated light is transmitted. Fiber-optic cable can handle many more times the information load (greater bandwidth) than either coaxial or twisted-pair. It is also unaffected by electromagnetic noise when, for example, the cable is run next to EMI-producing components like electric motors and welders. Fiber optic cable is the most secure media, because it is extremely difficult to tap and therefore is the media of choice for most government and defense installations.

In local area network applications, fiber optic cable is generally used as a high-speed, high-capacity "backbone" to link buildings together into one network. Fiber-optics provide a secure, high-capacity link between two points.

Now that you have an idea of the three main types of cabling media, let's look at how this might affect the design of a network.

DESIGN:

The logical design builds the foundation for a structured cabling system and provides the high-level cabling direction for the detailed physical design specification. In a structured wiring system, each functional element, or subsystem, is designed to solve particular requirements for different wiring functions. A structured wiring system consists of the following elements:

* Work Area
* Horizontal
* Telecommunications / Equipment room
* Building and Campus Backbone

For the development of the logical design, each of the above areas is defined using the most cost-effective and best-suited technologies. As each subsystem is functionally different, each has its own specific requirements and design considerations that must be considered for the overall system design.

To build the actual structure of the cabling system, each of the functional subsystems must be identified and its boundaries must be determined.

The first step in developing the logical design is to establish the design objectives. A review of the business requirements is necessary; this information can be obtained from a site visit. Translate those requirements into the appropriate design objective for the cabling design. Consider the following as you evaluate your design objectives:

* Future Growth
* Applications
* Cost constraints

In an existing building, many subsystem locations will be predetermined. The existing subsystems should be integrated into the logical design whenever possible.

Based on the user locations, mark each planned or existing work area subsystem location on a floor plan. Based on the building's space requirements and building structure, at least one telecommunications closet should be identified for each floor. Try to identify where the equipment would best be suited. Although it may be too early to determine the exact locations for the telecommunications closet and equipment rooms, it is useful to show where there is sufficient space in the building to accommodate these structures.

Mark the backbone cabling on the floor plan by connecting the telecommunications closet and equipment rooms together. If your cable system is to link two or more buildings together, then mark the campus backbone subsystem on the plan. Also be sure to identify any existing subsystem, usable cabling, or conduit on your floor plan, as well as any known restrictions for cable access or plenum requirements.

The floor plan should now show a high-level picture of the proposed cabling system. This floor plan is intended as a planning tool to be revised and fine tuned as you continue through the development of the logical design.

Now that each subsystem has been identified, it is helpful to make a list of each subsystem's functional requirements. Segment the site and network requirements into the appropriate functional requirements for each area:

* Applications requirements
* Distance requirements
* Existing technology and equipment
* Environmental and cost constraints
* High-level traffic and performance requirements

After the functional requirements have been determined for the specific subsystem, each can then be defined. It is recommended that you follow the logical design process below:

1. Determine specific requirements
2. Review the design requirements
3. Choose the appropriate technology/transmission medium
4. Determine necessary transmission electronic components
5. Determine the physical arrangement running the cables

INSTALLATION:

After the design of the cabling system is complete, it is now time to consider how to implement the actual installation of the cable itself. The are three main sources for performing this function;

1. Internal site-communication specialists
   (this service may not be available)
2. Equipment / system vendors
3. Cabling-specific contractors

All of the above organizations are very different in terms of overall business purpose, area of specialty and limitations. Some corporate guidelines may dictate the use of the internal site-communication specialists, but this can be a problem. The internal organization might be unavailable or not knowledgeable about a certain media or, not have the proper tools for a media like fiber optics. In this case an outside contractor or vendor may be the only alternative.

Most system and equipment vendors today can provide for the installation of cabling by subcontracting that portion of the project. And some companies have developed departments to perform cable installation. The main source of revenue for these companies comes from the systems and equipment sales, but in order to maintain account control and provide the customer with a "turn-key" system, they have entered into this business. Some vendors may limit the cable installation to only their equipment or they may not be able to connect to certain network technologies.

The third avenue for your installation of cable can be a cabling contractor. These companies are usually small, local organizations with no affiliation to any one vendor or technology. Anyone wanting to employ the use of a contractor must have some basic skills of negotiating "change-orders" and project management, and must know the legal aspects of contracting. Most cabling-specific contractors have emerged from the electrical contracting business and can provide a good installation for the dollar invested. But you must check references and identify any support requirements you expect the contractor to provide.

MAINTENANCE:

Regardless of who installs the cable plant, there are a few other key points to remember. As a company grows, offices and workstations change. As users change and move within your organization, the wiring needs to change. To easily accommodate the typical moves, adds, and changes in a cost-effective manner, both the current and future requirements must be considered. These changes must be anticipated and planned for accordingly. The cabling system must allow for easy changes and these changes must not disturb the normal operation of business. This can be obtained by keeping good records and "pre-wiring" with additional cable.

One of the best ways to accommodate changes is to "pre-wire" the workstation and horizontal subsystem. Pre-wiring implies that the wiring is run to all areas throughout a building regardless of whether those areas are occupied. Every location is wired even if it won't be connected at the time of installation. As your building or office configuration changes and these areas become occupied, the wiring will already exist to meet your needs.

Pre-wiring helps to reduce long-term installation costs. It is considerably more costly in terms of labor and disruption to your business to install additional wiring after the initial installation.

The cabling system should not interfere with the physical appearance of a building. In office environments, the cabling should not affect the interior decorating of the area. Horizontal cabling should be contained within some type of conduit and should not be exposed. Depending on the aesthetic requirements for the building, different installation methods can be used to help improve appearances.

The effort of administering and maintaining your cabling system can be minimized by keeping current and accurate documentation. There are three important aspects of cable administration and maintenance:

1. An accurate and up-to-date network map
2. An Easy-to-understand and logical labeling scheme
3. Cable records that coincide with the above

Although the actual development or implementation of these documents may not occur until the cable system has been defined, it is important that administration and maintenance be considered throughout the planning process.

Your particular cabling requirements will vary depending on your business environment and applications. No one cabling system can adequately solve all business environments. Investigate the different cabling options and vendors. You may need to extract certain elements from more than one type of cabling system. If you can conduct detailed research and analysis of the cabling system before the installation, this will certainly save you and your company a lot of time and money.

BIBLIOGRAPHY:

HP SiteWire Planning Guide, 5959-2201
HP LAN Configuration Guide, 5090-2607

AT&T Intro to Premises Distribution, 55-400-020
AT&T A Guide to Premises Distribution, 555-400-021
AT&T Premises Distribution System, 555-400-022

Ungermann-Bass Interconnectivity, 27234-01

Siecor Universal Transport System, CC-110

The Truth about Developing Cooperative Applications.

Robert H. Dey
Hewlett-Packard Australia Limited
29 Ringwood Street
Ringwood VIC Australia 3134
International.+61 3 871 1684

Abstract

The age of cooperative computing is no longer just a pipe dream, it is well and truly within our reach. However, it is not enough to grasp at technology alone. The approach an organization takes relative to the development and integration of applications in that environment may have a substantial impact on overall productivity.

Today's MIS philosophies and development methodologies may no longer be appropriate for this enhanced open system information topology. What is needed is a new paradigm for developing applications and the supporting tools and products that overcome the resource sink the technology introduces.

This paper discusses the issues and challenges cooperative computing application developers face and provides the forum the opportunity to consider the benefits of implementing a new Information Management infrastructure which promises a true productivity breakthrough.

Introduction

Nearly five hundred years ago, the Italian statesman and writer, Machiavelli said

> *"There's nothing more difficult to take in hand, more perilous to conduct or more uncertain in its success, than to take the lead in the introduction of a new order of things..."*

In the decade to come cooperative computing based applications will introduce a new order. Even though technology has taken enormous strides since Machiavelli's day, his words continue to ring true.

Client/Server computing, cooperative computing, distributed processing et al is being cited by almost all industry market research organizations like Gartner, IDC and Forrester Research, as being the best answer to harnessing computing resources available to our optimum advantage and to effectively integrate existing resources.

Generally, the three terms are used interchangeably in the industry, however, there are differences. Here are their definitions:

Client/Server computing splits the processing of an application between a front-end portion on a PC or workstation, which processes local data manipulation and maintains a user-interface, and a back-end portion on a server, which handles database and other numerical or computation intensive processing.

Cooperative computing is similar, spreading data for a given application across several systems. Cooperative computing makes optimal use of all systems on a network and makes data available to any user connected to it. Additionally cooperative computing supports 'peer to peer' programming as well as client/server.

Distributed processing allows an application (code & data) to run on more than one system.

Every significant computer vendor has made cooperative computing commitments of some sort. Indeed last September in San Francisco you heard John Young, Hewlett-Packard's President and C.E.O., together with Joel Birnbaum commit HP to being a leader in the cooperative computing market early in this decade. You've already seen the release of server hardware, integrating networking solutions, object management facilities like NewWave and a variety of user interface products to meet the apparent needs of the market.

The MIS Challenge

With specific reference to the Client/Server software market, Forrester[1] said:

> *"The most important element in the advancement of client/server computing will be the availability of programming tools specifically designed to build applications based on the bifurcated model."*
>
> *"Currently the task facing developers is daunting... Forrester estimates that there are only 10-20 third party developers and another 15-25 MIS organizations that can forge bifurcated applications with old-world tools."*
>
> *"Despite the severity of the problem, tools that uniquely address client/server software are years away."*

Users are demanding a friendlier face on their corporate applications, the sort of consistency in behavior and presentation they experience on their PC's. They expect that their data is stored on a centralized server and maintained under the direction of MIS and Database administrators. That the details of data integrity, concurrence and security considerations should be transparent to them. They want to be able to treat the application as if it were single, undivided and dedicated to their exclusive use. A pretty tall order. Fortunately cooperative computing promises all this and more.

1 The Professional Systems Report, C/S Software Realities
Volume 6, Number 10. Forrester Research August 1989

In reality the challenge for MIS and developers of applications for cooperative, or client/server processing is that the application is not single - it is multi-user, multi-tasking and multi-processing. It is not undivided, even in its simplest form, it will execute partially on the server back-end and partially on the users front-end; whether it be PC or Workstation based. It will not necessarily be dedicated to any user's exclusive use - it will be more than likely a corporate wide solution, spanning a network of machines and applications. Like an order entry system or executive information system.

The trick is not to make the complexity of the application a burden to the user. What is needed are tools to help the developer achieve their objectives. These tools fall into two broad categories; those that help in developing applications and those that assist in managing the application once it is commissioned.

For the purposes of this discussion the "Developer" is not one who concerns himself with bits, bytes and milliseconds and knows all the ins and outs of handling the quirks of the operating systems, and its subsystems at its lowest level. Nor will we consider the end-user developer who produces small applications single handedly or "bandaids" a word processing, presentation graphics solution or spreadsheet with a database management system of some description to fit the client/server model.

We will primarily concentrate on the business application builder who develops strategic enterprise wide solutions. The sort of function traditionally provided by MIS.

The issues that confront the MIS departments of most organizations, such as

the visible and invisible application backlog

proliferation of PC's and how to control them

hardware growth, centralization versus decentralization

industry standards versus proprietary offering

changing needs of organization and users

have not gone away and still need to be addressed.

CASE And Its Role

Whatever design methodology - modular programming, top-down, composite, structured design or other popular techniques - your MIS department uses, one thing is clear, development of software by manual hand coding methods is too labour intensive to be really economical. What is needed are automated tools to assist with the effort. The most recent of a series of emerging technologies to addresss this fundamental concept is CASE... Computer Aided Software Engineering.

Ask a dozen people what CASE is and you'll get a dozen different answers. CASE tools exist so that they can assist in the planning, analysis, design, prototyping, development, maintenance and documentation of business information systems. Despite the confusion surrounding CASE, there are no shortages of products available to conform and operate in accordance with most of the popular software engineering methodologies. The challenge is to use the right one and have its output capable of integrating with your database, dictionary, 4GL, tools etc. Software development is a tricky business as many of you know first hand. CASE tools often suffer from unrealistic expectations. We should not pretend they are the panacea to our application development needs alone. CASE tools can help define, analyze (or possibly build) systems, but there is nothing magic in them, the CASE process requires skill and patience.

Other tools needed for the development of commercial cooperative computing applications range from simple to complex. These 'other tools' as a subset of CASE tools, generally called "lower-CASE", would ideally provide a range of facilities such as database definitions, screen generation, query and report generation, graphics, decision support, source and time stamping etc., all with a common command syntax, behavior and appearance.

Again there are a plethora of tools available that meet these needs. Perhaps not as a single integrated solution but as a family of tools. Oracle, Ingres, PowerHouse from Cognos, HP's Allbase Tools fit this general mould for host based application development - but their support of cooperative application development with multiple data managers like Image and KSAM from our legacy systems and SQL or object oriented databases of the future is not guaranteed and could leave cooperative application developers stranded.

Open systems are the order of the day. Some vendors believe that in providing tools solutions with maximum inter-connectivity is 'open', others advise that their product complies with standards and therefore is of an 'open' nature. The real intent of 'Open systems' though is to give users and developers the choice to 'mix and match' products from any source for their solutions. They can feel assured that their choice will cooperate and be compatible with other components of their system without restriction.

An absolutely critical component required for integrating all the data reduced by CASE is a common central dictionary. What is needed is a central repository for storing all the meta-data. A fact of life is that not all dictionaries are the same, standards for dictionaries, encyclopaedias, repositories have not been defined. Hewlett-Packard is no stranger to the problems of incompatible dictionaries.

By solving this problem, and providing a central repository which integrates data stored in a variety of local dictionaries in such a way as to present a single definition, we will take a major step toward making cooperative computing a reality.

Graphical User Interfaces

And then there are the Graphical User Interfaces (GUI's) first pioneered by Xerox almost a decade ago. However, now that everyone has woken up to their potential there are so many GUI's we can utilise for a variety of implementations in our applications, that it is difficult to choose which one is best for us?

It may be easy for the user to choose the preferred (right) GUI, but what about developers? Ironically, ease-of-use, consistency and simplicity for the user is often accompanied by difficulty, inconsistency and complexity for developers. In implementing GUI's the developers productivity is negatively impacted. It is decreased by the sheer complexity of the Application Program Interfaces (API's) each with anywhere between 200-700 intrinsics per GUI. Total incompatibility of various API's and their tight interweaving through typical applications maximizes difficulty of porting to other platforms.

And there's more... because windowing systems rely on message passing, your cooperative applications will need to handle hundreds of different messages.

We are confronted with a changing application model which is resulting in a significant and growing skills deficit for programmers. As we move from the logic-driven linear programming of tradition to user-driven multipath development typical of GUI's and cooperative applications, training will account for a major portion of funds allocated to bring the technology into an organization. That investment must be made in order to have any hope of success.

At this point we have not even considered the integration of our existing systems into this paradigm. Not all of us want to be forced to make a quantum leap to the cooperative computing technology. The time may not be right for such a move - the leap too great and risky. Most of us expect vendors to provide a bridge to help us move at our pace.

This is necessary because currently 60% of all computer connections are via the humble terminal, seemingly incapable of providing the GUI environments of PC's and workstations. To convert an entire enterprise's terminals to PC's or workstations would stifle even the most ambitious manager's enthusiasm.

In order to take another step towards making a major productivity breakthrough in creating effective cooperative applications, what is needed is a product that makes it as easy to develop GUI based applications as it has been to develop terminal oriented ones. It would also be nice if you could use one product that supports both terminals and your choice of GUI to enable you to evolve incrementally from your current environment to cooperative computing.

Summary

Hewlett-Packard is conscious of this need, we understand what it is that is required by developers at all levels to move to the new technology and to take advantage of the many attributes and benefits the environment has to offer today and in the future. Think about what it is that you need to help your transition to the world of cooperative computing.

Stay tuned!

Quality Software -- Walking It Through

Lisa Burns
Hewlett-Packard
Corporate Offices
P. O. Box 10301
Palo Alto, CA 94303

I've worked in software for seven years now, and my experiences with quality have been varied. I worked on one team where we never could quite get the process down -- our releases had consistent problems. Another project team was always so schedule-driven that we went to our first user site test knowing that much of the code had never been tested, and that parts that had been tested flat did not work. I've also been fortunate enough, however, to work on two different projects where the quality of our product was consistently high. Installation steps worked without a hitch, and only very minor problems occurred during user testing. Let me assure you, these last two projects were much more pleasant for everyone involved!

So what was the difference between the high and low quality projects? The programming staff was no better on the successful projects. The programming tasks were no easier either. One of the big differences, however, was that software walk-throughs were held at every phase in the successful projects.

Walk-throughs are sometimes called peer reviews, or as they are at HP, software inspections. A walk-through is a process by which material to be released to users is reviewed by project team members with the goal of identifying errors, needed clarification or omissions. Project members receive this material in advance of the walk-through meeting, and spend time reviewing it by themselves. Then, at the meeting, members point out logic errors, unclear documentation, or missed test procedures or data cases. The author of the material then has the opportunity to correct these problems before the material is released to the project's users, or before the next phase of the project is started.

What kinds of material can be reviewed in a walk-through? Project phase documentation, such as investigation write-ups, external specifications, internal specifications and design alternatives can be examined in a walk-through. The cost of catching a design error during the external specifications phase, before a single line of code has been written, is exponentially lower than correcting the same error after the code has been released to users. Thus, walk-throughs conducted on these design documents are perhaps the most important of all. In addition, of course, walk-throughs can be heold on code itself. Logic errors, omitted data cases and other program defects can be prevented in this way. Perhaps more importantly, problems with explanatory comments and maintainability of code can also be prevented by walking through source code. Finally, test plans including procedures and test cases can be reviewed in a walk-through before the test is executed. This can help identify missing or erroneous test cases before the investment of testing time has occurred.

In addition to project team internal documentation, user documentation can benefit from the walk-through process. User guides, training material and operational guides can be reviewed in a walk-through meeting for clarity, technical correctness and completeness. Catching errors in these documents before they are distributed can prevent hundreds of unnecessary calls from confused users, and can prevent the cost of re-printing incorrect manuals.

So now you know that you have some material which ought to be reviewed. How do you go about conducting the walk-through process? The steps in the process are as follows:

1) Complete the material to be reviewed.

2) Choose the participants and a time for the walk-through.

3) Distribute the material to the participants.

4) Prepare for the walk-through.

5) Hold the walk-through meeting.

6) Resolve any problems identified in the meeting.

The first point, complete the material, may seem obvious. However, scheduling a walk-through meeting before code is completed for a module, for example, may result in frustration for the participants when the meeting is postponed due to an unforeseen problem. Likewise, distributing one version of a manual to walk-through participants and then replacing it with another version at the walk-through meeting itself will also cause frustration. Conversely, waiting until two months after a test plan was written before holding a walk-through will produce different problems. The author may not recall why some test cases were included and others omitted by that time. For this reason, it is best to wait until the material is complete, then hold the walk-through within one or two weeks.

Choosing participants for a walk-through is one of the keys to making the process work well in your organization. Choosing at least three, but no more than six participants works best for us. Always include a user or user representative in every walk-through, even if the material is very technical. It is important to retain the user perspective throughout the life of a project. Similarly, if you are attempting some very difficult techniques in code being reviewed, you may wish to include a technical expert on the walk-through team. Finally, if you are working on interfacing specifications, you may wish to include a member from the project with which you are exchanging data to ensure that their requirements are met.

One final note regarding choosing participants. Remember that the whole point of the walk-through process is to identify potential problems with your software or documents. Therefore, it is important that people feel comfortable enough to point out errors and problems to the author of the material without fear of negative consequences of doing so. For this reason, it is usually best not to include supervisors of the author in the walk-through meeting. This way, the number of errors found in a person's work cannot influence their supervisor's estimation of their ability.

In terms of scheduling the meeting, be sure that all members can attend at the chosen time. Choose a location where participants will not be interrupted, and allow one hour for the meeting. At this time you may also wish to assign roles to the participants. You may choose a reader who will lead the group through the material during the meeting. You may also wish to assign a moderator who will ensure that the group stays on track and who also records any problems or suggestions identified during the meeting.

The next step in the process is to distribute the material being reviewed to the walk-through participants. The most important thing about this step is to ensure that people have adequate time to review the material prior to the meeting. This means that materials must be distributed at least 48 hours ahead of time. Four days ahead is better, and will help ensure that participants prepare adequately. The author will need to allow enough time in his or her schedule to duplicate the materials and distribute them this far in advance.

In addition to the material which is actually being reviewed, you may wish to provide supporting material to help the participants understand the scope of the project. For example, if a new module is being reviewed, you may also wish to distribute code for the module which will call the new module. In the case of an internal design document walk-through, you may wish to provide the external specifications of the same module so that participants can check that the internal design meets those specifications.

Include an agenda page with the material distributed. A sample is shown in Figure 1. The agenda includes the time, date and location of the walk-through meeting, the participants and their roles, a list of the material being reviewed and a list of all additional material being distributed to participants. Finally, the agenda includes a note regarding anything special you would like the participants to be especially careful to check for. In this example, the author wants us to check the setup procedures for completeness and clarity, to ensure that versions of interfaces specified are correct, and to ensure that the test steps are complete.

Armed with the agenda, the supporting material, and the material to be reviewed, the walk-through participants begin preparing for the meeting. Preparing will take between 45 minutes and 2 hours, and participants will need to plan accordingly. As they examine the material, they should consider the effect of a new module on existing code, or of changes to the system on all aspects of the software. The checklist shown in Figure 2 may help the reviewers to consider the effect of a change on different parts of an application.

Now (finally!) we are ready to hold the walk-through meeting itself. The reader takes the group through the material, asking for any problems or questions at each paragraph or section. Participants point out any areas in the material which they feel are in error, are in need of clarification, or could be improved. The moderator notes these points on a log sheet. A sample format for the log sheet is illustrated in Figure 3. At the end of the meeting, this log sheet is given to the author of the material so that corrections can be made. The entire meeting should take no more than an hour and a half.

The most important thing about conducting the walk-through meeting itself is to create an atmosphere where participants are comfortable identifying potential problems but at the same time the author of the material does not feel threatened. It is important for participants to employ tact when pointing out potential problems. One rule to follow here is to be impersonal with criticism, but personal with praise. For example, if a participant finds a potential problem in a documentation paragraph, he or she might say, "I think that a user might be confused by the explanation of the new field in section 2." If a participant has been especially impressed by something in the material, however, he or she might say, "I especially liked your explanation of the new report in section 3".

The next most important thing about the walk-through meeting is to ensure that the discussion sticks to problem identification, not problem solving. Participants should point out logic errors or omissions, and should not concern themselves with how to correct them. Also, participants should be careful to concern themselves with functionality, not style. This means that if a program conforms to shop standards and is functional, the fact that the author uses a different style of writing should not be an issue. The walk-through moderator can help keep the participants on-track in either of these cases.

Once the meeting has concluded and the list of potential problems has been compiled, it is the author's job to make corrections and clarifications to the material. At this point, he or she may do one of two things. If the list of problems is short, and the corrections can be made easily, the author simply makes the corrections and notes how each was resolved on the problem log sheet (see figure 4). The updated sheet is then re-distributed to the walk-through participants.

If correcting the problems will be a larger task, the author makes the corrections and then schedules a second walk-through meeting with the same participants. At this second meeting, the reviewers will walk-through the corrected material. Some shops use a guideline that if the corrections take more than one day to make, a second meeting must be scheduled. Either way, the walk-through participants are informed of the way each point was addressed, so that they can voice any concerns about the correction.

This process of walking through documents, code and test plans has been extremely beneficial for projects in our shop. Most of the projects within the department are implementing this process for at least some of their projects. Teams hold walk-throughs on project investigation documents, new screen designs, new code, and changes to existing code. In addition, walk-throughs on plans, user guides, and even on important memos are held. The quality of all of these products has been significantly improved by the process.

My team is quite insistent about walking through almost everything they produce! Every program change, every test plan, and every design document is reviewed. In addition, all user documentation is also reviewed. Each walk-through team includes a user representative, and members from interfacing teams are frequently included. Walk-throughs have helped us produce very high quality code. With a system under maintenance with over 100K lines of COBOL, we have an average of five very minor defects found in each initial user test. We have had very few problems reported after the code is released to production and installed in all sites. Only once in five years has a problem been large enough to warrant a "patch" release.

Of course, the main motivation for any project team to introduce walk-throughs is to improve software quality. However, our department as a whole has realized additional gains from the process. First of all, programmers and support personnel are cross-trained. Even if an administrator is primarily responsible for on-line programs, he or she will become familiar with the batch portions of a system by participating in those walk-throughs. This helps reduce training time and reduces the impact of turnover within a team.

Also, walk-throughs allow experience and expertise to be shared throughout the department. Programmers participating in walk-throughs for another project can lend their technical knowledge of a particular technique to that team. Communication within one team and between teams is also improved with walk-throughs. By actually examining specifications and code for modules with which they will have to communicate, programmers more fully understand the requirements for new code and for code changes.

Finally, code and documentation which has been reviewed in a walk-through is much easier to maintain in the future. As walk-through participants identify logic or explanations which are unclear, program and user documentation is clarified. This will mean less confusion the next time that same material must be enhanced or changed.

If you are going to implement walk-throughs within your own shop, you may benefit from these recommendations based on our experiences:

1) Make sure that all of your staff understands the walk-through process. Training is essential. You may wish to training your programmers and administrators informally by having them observe other teams conducting walk-throughs. Alternately, you may wish to train them formally through classes provided by your organization or by enrolling them in outside classes. Several organizations provide training in structured walk-through techniques.

2) Ensure that all phases of a project are walked-through. The earlier in a project that walk-throughs are conducted the better. Review design, specification, code and test phases.

3) Ensure that all project members work is walked-through. Do not limit the sessions to review of junior team members work. They will feel singled out for examination. No one is perfect, and walk-throughs will improve the quality of everyone's work.

4) Create a non-threatening atmosphere during the meeting. Ensure that people understand that error identification is positive, and that people will not be evaluated on the results of a walk-through.

And finally, most important:

5) Ensure that time for walk-throughs is included in your schedule. The time to distribute material, to prepare for a meeting and to conduct the meeting must be built into the project schedule. Our experience has shown that far from adding time to a schedule, walk-throughs actually reduce it. Errors are caught earlier and can therefore be fixed more quickly. Participants must have the time up front, however, to review material and attend meetings.

Our experience with walk-throughs has been extremely positive. Programmers and administrators feel that time spent in reviewing material is time very well spent. Hopefully the tips outlined above will help you implement walk-throughs in your shop successfully. The best measure of success of walk-throughs will be their positive effect on your programs, your documentation, and ultimately, on your users!

FIGURE 1

AGENDA

DATE DISTRIBUTED	March 28, 1988
DISTRIBUTED BY	Lisa Burns
MEETING DATE/TIME/LOCATION	Cape Cod room, 1 p.m. April 6th
MATERIAL TO BE REVIEWED	System Test Plan

MATERIALS

1. System Test Plan
2. List of service request included in release G.00.00
3. List of improvements identified in release F.00.00 debriefing

OBJECTIVES FOR REVIEW

1. Check set-up for completeness, clarity
2. Check versions of interfaces for accuracy
3. Check test step completeness

WALK-THROUGH TEAM

MODERATOR	Joe Smith
READER	Mary Jones
AUTHOR	Lisa Burns
REVIEWER 1	Bob Terry
REVIEWER 2	Don Jackson

FIGURE 2

CHECKLIST FOR WALK-THROUGHS

1. TECHNICAL ISSUES

Is the source code copylib affected? Are $INCLUDE files affected?

Will other programs need to be re-compiled? Tested? Which modules are affected?

Is the database affected? Which items will be added, renamed?

Does the data dictionary need to be updated for this change?

Will product installation steps need to be adjusted? Does the database need to be converted? Is new JCL being released?

Is a new forms file required for VPLUS screens?

How is stack space affected by this change?

Are group or account SL changes required for this change?

2. USER ISSUES

How will the change affect users? Do manuals need updating?

Will training need to be prepared? Will hands on training be needed? Will existing training modules need to be updated?

Will office procedures for system use need to be changed?

Will users need to prepare data for the installation of this software? What manual steps will be required in the installation?

3. INTERFACE ISSUES

Which interfacing systems are affected by this change? Will testing need to be conducted with these systems?

Is this feature dependent on changes made by another team?

Does this change require code supplied by another team? Which version of that code is required? Will the subroutine be prepped in or accessed via an SL?

Are interface file formats affected? Which teams must be notified of the format change?

4. TESTING ISSUES

Does this change require special test data? Specific test environment?

When will new code from other teams be ready for testing?

Will other teams need to be involved in testing?

Who will supply test data?

Is a special test environment required for this change?

Does this change require a specific version of third-party software?

Does this change require a specific version of MPE?

FIGURE 3

LOG SHEET

PAGE 1 OF 1 WALK-THROUGH DATE April 6th, 1988

MATERIAL REVIEWED System Test Plan
SERVICE REQUEST # G.00.00 release

PARAGRAPH/ LINE/PAGE	DESCRIPTION	TYPE	RESOLUTION
Section 1	Incorrect version #	Error	
Section 1	Add specification of MPE version	Omission	
Section 2	Add step to test various printer models	Omission	
Section 3	Include unit test for search criteria change	Omission	
Section 4	Step to initialize data-base should come before data extract step	Error	
Section 4	Expand description of file manipulation steps	Clarity	

FIGURE 4

LOG SHEET WITH PROBLEM RESOLUTION NOTES

LOG SHEET

PAGE 1 OF 1 WALK-THROUGH DATE April 6th, 1988

MATERIAL REVIEWED	System Test Plan
SERVICE REQUEST #	G.00.00 release

PARAGRAPH/ LINE/PAGE	DESCRIPTION	TYPE	RESOLUTION
Section 1	Incorrect version #	Error	Version number corrected
Section 1	Add specification of MPE version	Omission	MPE Version V-Delta-1 added
Section 2	Add step to test various printer models	Omission	Steps to test 2934, thinkjet, lazerjet, deskjet added
Section 3	Include unit test for search criteria change	Omission	Unit test plan added to section
Section 4	Step to initialize data-base should come before data extract step	Error	Steps re-arranged
Section 4	Expand description of file manipulation steps	Clarity	Narrative updated and expanded

What Does This Benchmark Mean To You?

Frank Rowand
Hewlett-Packard
19447 Pruneridge Avenue
Cupertino, CA 95014-9974

I. INTRODUCTION

Benchmark reports are readily available from sources such as trade journals, HP Performance Briefs and conference proceedings. The report is written to explain some performance metrics as they applied to the goals of the author (such as multi-vendor comparison, positioning models within a product line, sizing a system to be purchased, capacity planning, or forecasting performance changes due to factors such as an operating system upgrade or an application upgrade). However, you must decipher the significance of a given benchmark with respect to your needs.

This paper will discuss the purpose of some HP and industry standard commercial benchmarks and how to better understand the results of these benchmarks. Some of the questions that will be addressed are:

In what ways are benchmarks useful?
What are key benchmark metrics?
How should you interpret these metrics?
How are these metrics often misinterpreted?
For what purposes are benchmarks conducted?
How do you apply the results to your situation?

II. BENCHMARK

benchmark n 2b : something that serves as a standard by which others may be measured [WEBSTER].

Webster's definition tells us what you do with the something that is a standard but leaves us free to determine what that something is. My working definition will be:

> a set of metrics which describe the behavior of a set of software within a specific hardware and software configuration.

A single benchmark doesn't tell us a lot. A benchmark becomes much more useful when we have other benchmarks to compare it to. But how do you compare benchmarks? When we put the two definitions above together, we see that when we compare two benchmarks we are comparing two set of descriptive metrics. (Note that we are not comparing two sets of hardware.) So the first step to understanding how to use a benchmark is to understand the metrics used to describe the benchmark.

III. METRICS

There are only a few metrics that are reported in almost all benchmarks. There are also a lot of metrics that may only be of interest within a specific set of benchmarks. The common metrics are response time, throughput, percent cpu busy, elapsed time, and MIPS.

RESPONSE TIME is the elapsed time from the beginning of a transaction to the end of the transaction. The beginning of an interactive transaction is usually defined as the time that the terminal user presses the last key required to start the processing of the transaction (typically the <enter> key, <return> key or a function key). In some benchmarks the beginning is instead when the first key of the transaction is pressed. The end of the transaction is most often either when the last character of the response is seen by the user or when the user is able to start typing the beginning of the next transaction (when the user sees the "prompt"). A third definition is when the first character of the response is seen by the user.

THROUGHPUT is the amount of work completed per unit time. Some examples of throughput are transactions per second (TPS, used in On Line Transaction Processing benchmarks), data base updates per second, and megabytes per second (transfer rate to or from a peripheral).

PERCENT CPU BUSY is the percent of the time that the processor is processing instructions instead of being paused (or in a dispatcher idle loop).

ELAPSED TIME is length of time that something, usually a batch program or job, takes to complete.

MIPS is included in the list of metrics only with the greatest of reluctance. The only reason to mention MIPS is to explain that it is an extremely poor metric in the context of software benchmarking. There are many reasons for this, but I will explain just one.

The technique used to measure or calculate MIPS is not as obvious as many people assume. MIPS is defined as how many million instructions the cpu can process in one second (either peak or sustained). Theoretical peak MIPS can be calculated as one divided by the number of seconds to complete one instruction. Actual MIPS (which can be affected by factors such as pipeline interlocks, cache misses, and memory bus contention) is measured by running an instruction stream on the cpu and measuring elapsed time (MIPS = (number of instructions / elapsed seconds) / 1000000). This means actual MIPS can vary, depending on the instruction stream that is executed. The next innovation to cloud the meaning of MIPS was to define the actual MIPS of one machine to be "one MIP" and then rate all other machines relative to the standard machine. Thus we have IBM MIPS, and VAXMIPS. As an example of the vagueness of the definition of MIPS, we have seen the relative performance of two HP machines, based on some common VAXMIPS software suites, vary from a ratio of 0.46 to 0.9 [JACKSON].

IV. REPORTING METRICS

The way in which a metric is reported strongly affects it's usefulness. For example, if two benchmarks have equal average response times you might assume that both systems are equally useful. But what if you are buying a system to control a chemical process where response time of two seconds is required? Knowing that average response time of a system is one second is not sufficient; you must know that the maximum response time is less than two seconds. If you are instead purchasing a system for use in an on-line commercial environment you may be more concerned about the standard deviation of response time since a large variance in response times is often claimed to be detrimental to user productivity.

Some informative metric reporting formats are average, 90th percentile, minimum, maximum, standard deviation, graph of values of the metric vs. number of occurrences of the value, and graph of the metric vs. elapsed time within the benchmark. These formats are listed in order of increasing detail of information. The less detail, the easier it is to compare a metric between benchmarks ("does benchmark A have a smaller or larger average response time than benchmark B for 100 users?" is easily answered; "is the response time vs. elapsed time graph better for benchmark A or benchmark B?" may be the subject of some controversy). The more detail provided, the more likely that anomalies will be visible.

V. BENCHMARK "GOODNESS"

There are many factors that combine to make a benchmark "good" or "bad". However, since the attributes of a "good" benchmark are usually much smaller in number than the factors that make a benchmark "good" it is easier for the benchmark engineer to verify whether he has built a "good" benchmark by examining its attributes. *A BENCHMARK DOES NOT HAVE TO HAVE ALL THE ATTRIBUTES OF "GOODNESS" TO BE USEFUL.* Specific uses of benchmarks may require only one or two of the attributes. I will return to this subject after describing the attributes. A benchmark is generally considered "good" if it has the following attributes:

representative
scalable
repeatable

A benchmark is REPRESENTATIVE if the actions it performs are similar to the actions that the actual application being benchmarked would perform. The similarity includes factors such as arrival rate of transactions, mix of transactions executed, locality of data access, I/O rate, number of users, and think time. A benchmark can be ranked on a continuum ranging from totally representative to not at all representative. If the input of users to a real system are captured and then repeated exactly in an exact copy of the real system (data and configuration) then the benchmark is totally representative of that system in that time frame. (Real systems typically have several standard workloads, with each workload exhibiting different characteristics. A financial system will often have a peak workload at the end of a fiscal year, but may show a different type of heavy workload while the annual budget and forecasts are being prepared.) For each aspect of the benchmark that is different from the real activity, the benchmark is less representative.

A benchmark is SCALABLE if (1) its performance behavior changes between different test configurations in the same way as a real application and (2) if its performance on a given test configuration changes as load is added to a given configuration in the same way as a real application. Different applications will scale differently so this definition of scalable is only valid within the context of a specific workload. This may appear to be a subset of representative but it is often let out of the definition of representative.

A second useful definition of scalable is: A benchmark is SCALABLE if, as the benchmark is executed on more powerful configurations, the benchmark performance is not limited by bottlenecks in the benchmark software (though it may be limited by bottlenecks in the hardware or software configuration).

A third definition of scalable is: A benchmark is SCALABLE if the amount of work done by each transaction is independent of the hardware configuration and the load on the system.

A benchmark is REPEATABLE if its performance behavior is similar each time it is executed on a given configuration of a system. There will usually be some variance of the value of a metric between runs on a given system. If this variance is larger than the variance of the value of the same metric between two different systems then the two systems must be considered to have equal performance for that metric.

I mentioned earlier that a benchmark can be useful even if it is lacking one or more of the attributes of a "good" benchmark.

Sometimes one of the attributes is not required for the benchmark metrics to be accurate. In this case it is not worth the added expense of verifying the attribute.

Sometimes the "correctness" of the metrics *may not be strongly affected* by the absence of one of the attributes but the cost of providing the attribute is large. In this case it may not be worth the added expense to provide the attribute. One of the most difficult tasks in the performance field is to balance the complexity and expense of a study (for instance, a model or a benchmark) against the increased exactness or correctness of the result. Most benchmarks exist within this range, where tradeoffs between cost and exactness must be made.

Sometimes the "correctness" of the metrics *may be strongly affected* by the absence of one of the attributes but the cost of providing the attribute is so large that it cannot be provided. Once again, some difficult choices must be made. If the value of the benchmark results is small then the benchmark should not even be attempted. If the need for data is strong and there is no alternative technique to provide the data then the benchmark (lacking the expensive attribute) will be performed but the results should be treated with great caution.

If a benchmark is not repeatable then it is not useful for performance measurements. A benchmark that is not scalable has limited usefulness. A benchmark that is not representative may be useful for some purposes, such as comparing different vendors' products.

Now that I have said that it is easier for the benchmark engineer to measure the attributes of a benchmark to verify its "goodness", I have to confess that everyone else has to either believe that the engineer desired to match these attributes and that he did indeed verify them or try to understand the "goodness" through analysis of the benchmark. I will give some insight into this analysis in the next few sections. I will also provide some guidance regarding the "goodness" of some Hewlett-Packard Commercial Systems Division benchmarks, some industry standard benchmarks, and some other proprietary benchmarks.

VI. BENCHMARK CONSTRUCTION AND EXECUTION PURPOSE

Two very important factors in understanding the results of a benchmark are often not explicitly noted. Since the process of building a benchmark can be very time consuming and expensive it is normal to reuse the same software set for several years. Thus the first factor to understand is: what was the reason for running the benchmark? The second factor to understand is: what questions was the benchmark originally built to answer? Some reasons for building and running benchmarks are listed below.

POSITIONING OF CONFIGURATIONS WITHIN A PRODUCT LINE.

Each vendor needs to understand the relative performance of the platforms they provide. Pricing decisions are strongly affected by system power. The sales representative must be able to quote the proper size system to provide a good solution to the customers needs. The customer needs to know if he has an adequate growth path if his application or business is growing and he may need to know what smaller systems would be appropriate if an application was distributed to other units within his company. All of these needs require an accurate positioning of systems, relative to each other.

Benchmarks to provide product line positioning are most useful if they are representative. When specifying the relative performance of two systems, the vendor may provide a range of values (reflecting different workloads) instead of a single number.

POSITIONING OF CONFIGURATIONS AGAINST THE COMPETITION.

Each vendor is very aware of the relative performance of their platforms and/or solutions compared to the competition's platforms and/or solutions. Performance is a key element in the value provided by a computer system. In the computer market systems must be priced competitively based on the value provided to the customer. When marketing a product, the vendor can use superior performance as a competitive advantage or, if performance is not superior, may have to convince the customers that other elements in the value equation lead to a better overall solution than the competition's.

Benchmarks to provide inter product line positioning are more useful if they are representative. If you plan to use the system for one application and that application exists on both platforms then by benchmarking the application on both systems you can calculate accurate price / performance numbers for each system that is compared. If you later purchase or build another application with a different workload you may find the new application has different price / performance ratios on the same systems.

If you build your own applications you will probably find that there are not comparable benchmarks on two vendors' systems that match your application's workload. The best that you can hope for is to find an industry standard benchmark that exercises some of the features of the system that you expect to use. For instance, if you are developing a commercial OLTP application, TPC Benchmark A (which will be described later) will be a more useful benchmark than a technical benchmark such as Whetstone. (TPC Benchmark A has many of the characteristics of a commercial OLTP application since it reads and updates both large and small datasets in a database, performs terminal reads and writes, and spends more time in system code than user code.) Since TPC Benchmark A is a simple, synthetic benchmark it won't be representative of any specific application (for example, TPC Benchmark A does not include any batch work, while the average HP3000 system includes a significant batch component). This means that you will be able to use an industry standard benchmark to provide a rough first cut of which vendors provide comparable performance or price / performance.

OPERATING SYSTEM TUNING

Hardware and software engineers need to characterize and tune the behavior of the products they build and maintain. The needs of these engineers include tuning the system to perform well in typical customer environments, stress testing, reliability testing, and finding bottlenecks.

Engineers may choose to tune to a proprietary, representative benchmark so their systems will efficiently process their customers' typical workloads. They may instead choose to tune to most efficiently process a non-representative, industry standard benchmark (if they are going to tune to a non-representative benchmark then they ought to tune to something they can compare to their competition instead of to a proprietary benchmark). Or they may tune their system both for representative benchmarks and industry standard benchmarks.

A "bad" benchmark can be useful for stress testing and finding bottlenecks. A non-representative benchmark can be built to utilize a resource more intensively than a normal customer would, or it could focus on a small subsystem to remove much of the complexity of benchmark analysis. However, it is important to not rely too heavily on "bad" benchmarks, but to instead focus on representative benchmarks that test the entire system.

CAPACITY PLANNING

System managers need to properly size systems for new applications and for the growth of existing applications.

SYSTEM CONFIGURATION

Vendors and system managers both need to understand the most effective combination of components to create a system configuration. The goal is to have a balanced combination of processor speed, memory size, number of channels, number of discs, etc.

APPLICATION PROGRAMMING GUIDANCE

Application programmers need to know how to best use the operating system interfaces and utilities. For example, what block size should be specified to maximize throughput while reading a tape file? When should an index or key be added to a data set? Is it better to kill a son process that will be dormant for a long time (say, for twenty minutes) and recreate it when needed or to leave it suspended (holding resources while dormant) and awake it when it is again needed?

Questions with a narrow focus, such as the examples above, are often best answered by a simple, non-representative benchmark. The danger of this approach is that a component of an application may be optimized to the detriment of the rest of the application or system. Questions about algorithms with non-local impact may be better answered by prototyping, instrumenting or externally measuring the application in a production environment, or benchmarking the application with an actual workload.

VII. BENCHMARK CONSTRAINTS

I have listed some reasons for building and running benchmarks. A related issue is how the goals of the benchmark constrain the creation or execution of the benchmark.

A proprietary benchmark offers great flexibility to the owner of the benchmark.

It is excellent for comparisons within a product line.
It can be representative of a very specific class of customers.
It can be quickly changed to test new hardware and software implementations and features or to stress newly found bottlenecks.
You may not be able to compare the same benchmark in separate reports since the benchmark may have changed.

An industry standard benchmark provides a comparative benchmark.

It is excellent for comparisons between vendors.
It probably won't be representative of any particular application.
It will be slow to change to test new software and hardware technology.
You will be able to compare the same benchmark is separate reports but only if all testers have followed the same, precisely defined methodology.

The reason for running a specific benchmark will affect the resulting metrics. The purpose of the benchmark will be to optimize one or more of the metrics. If one metric is optimized, then another metric may be affected negatively. Some of the common benchmarking reasons are:

For a given hardware configuration, what is the performance?

This one is easy, run a benchmark on the system. There is no hidden agenda.

What is the maximum performance the system can possibly provide?

There are many techniques for this type of benchmark. First, keep adding hardware (memory, discs, channels, etc.) until adding more does not improve performance. Second, tune everything you possibly can (the benchmark, the data base management system, file placement on discs, operating system parameters). Third, decrease think time to reduce the number of users to minimize the amount of resources required to maintain user contexts. (Think time is the length of time the user takes to respond to a prompt from the computer.)

When you read a benchmark report, examine the cost of the hardware and software. The benchmarker will be willing to double the cost of the system to provide a 10% performance gain. You might not be so willing to pay that price. Finally, check that think time is realistic.

If a benchmarker fully exploits the techniques to maximize system performance then the benchmark will not be representative of most commercial systems. If some or all of the techniques are applied in moderation then the benchmark may be exactly representative of your environment. You have to apply your own judgment to decide how representative or not each benchmark is.

What is the performance of the system with an appropriate configuration?

The methodology is: (1) choose an "appropriate configuration", (2) look for bottlenecks and excess capacity, (3) apply moderate tuning and adjust the configuration to remove bottlenecks and excess capacity, and (4) repeat steps 2 and 3 as needed.

The key to this type of benchmark is: what is an appropriate hardware configuration (before and after tuning) and what is moderate tuning? If either of these attributes become very non-representative then the benchmark is similar to a benchmark run to measure maximum system performance. This is the type of methodology that would normally be used to a perform a representative benchmark.

Now that I have provided a basis to analyze benchmarks I will discuss three classes of benchmarks: industry standard benchmarks, benchmarks used by the Hewlett-Packard Commercial Systems Division, and other proprietary benchmarks.

VIII. INDUSTRY STANDARD BENCHMARKS

TPC BENCHMARK A (TPC-A)

The Transaction Processing Performance Council (TPC), whose members include HP, Tandem, IBM, and DEC, issued the standard specification in November 1989 for its first benchmark, TPC Benchmark A. TPC Benchmark A was based on Debit Credit but is much more tightly specified so that the results of TPC Benchmark A, run by different vendors, can be compared. This benchmark is currently the best industry standard commercial benchmark for comparing different vendors' systems.

TPC Benchmark A is a simple, non-representative, interactive, on-line transaction processing (OLTP) benchmark. However, the benchmark exercises many functions that are commonly used by a commercial, OLTP application; it accesses a single data base (centralized or distributed), data base access is update intensive, small and large data sets are used (affecting lock contention), and terminal I/O is required. Both the construction of the benchmark and the reporting of the metrics are strictly defined [TPCA].

A very important requirement of the TPC Benchmark A standard covers data integrity. The benchmark is required to meet the "ACIDity test" - the properties of Atomicity, Consistency, Isolation, and Durability. The definitions of ACID are:

Atomicity: "The system under test must guarantee that transactions are atomic; the system will either perform all individual operations on the data, or will assure that no partially completed operations leave any effects on the data."

Consistency: "Consistency is the property of the application that requires any execution of a transaction to take the database from one consistent state to another."

Isolation: "Operations of concurrent transactions must yield results which are indistinguishable from the results which would be obtained by forcing each transaction to be serially executed to completion in some order. This property is commonly call serializability."

Durability: "The tested system must guarantee the ability to preserve the effects of committed transactions and insure database consistency after recovery from any one of [a specified list of failures]."

The ACIDity test ensures an application environment that is representative for on-line transaction processing systems; system performance can not be increased through sacrificing data integrity.

The standard also defines the benchmark workload, including data set size per TPS, number of users per TPS, minimum average transaction cycle time (think time + transaction duration), and transaction arrival rate. All users in the benchmark access a common set of data files, which may be a distributed data base. The transaction includes reading from a terminal, updating a large dataset, a fairly small dataset, and a very small dataset, writing the transaction to a history file, committing the transaction, then writing to the terminal.

Reporting standards include what to report and how to report it. The required "full disclosure report" covers many areas. The information required to replicate the benchmark must be furnished. This includes items such as application source, settings for customer-tunable parameters, and system configuration. The method used to determine repeatability must be documented. The results of the ACIDity tests must describe how the requirements were met. The terminal connect method must be specified as either wide area network or local connect. The throughput of local connect is specified as "tpsA-Local" and the throughput of WAN connect is specified as "tpsA-Wide". The standard clearly states "These two metrics are NOT comparable with each other". An independent audit of the benchmark results is highly recommended though not required.

The primary metrics required in the full disclosure report are Transactions Per Second and system price. Supporting metrics include response time data. The maximum throughput (TPS) is constrained by a response time limit; 90% of all transactions started and completed during the measurement window must have a response time of less than two seconds. A frequency distribution graph of response times is required. The average and 90th percentile response times are indicated on the graph. The maximum and average response time must also be reported. A second graph plots 90th percentile response time versus TPS. Values for response time at 50%, 80%, and 100% of reported throughput are required.

The above paragraphs are meant to convey a flavor of the standard, not to be a complete description. If you are interested in more details, please read the standard.

TPC Benchmark A is not representative and it is scalable. Even though it is not representative, *it is currently the best commercial, OLTP benchmark available for comparing the systems of different vendors. It provides a framework in which the metrics of competing vendors can be reasonably compared, unlike its predecessor, Debit Credit. Another contribution of TPC Benchmark A is that there are two goals that it emphasizes: maximize TPS and minimize price per TPS.*

DEBIT CREDIT

Debit Credit is a simple, on-line transaction processing benchmark. The benchmark is a simplified version of a benchmark used by Bank of America to select a vendor for an on-line system. It received a large amount of publicity through an article in the April 1, 1985 issue of Datamation ("A Measure Of Transaction Processing Power"). Even though I have included it in a list of Industry standard benchmarks, Debit Credit is an incomplete, de facto standard. Many vendors have implemented the Debit Credit benchmark, but there has been great variety in the implementation and reporting methods chosen by each. Thus the metrics in each published report can not be compared to any of the other reports without a lot of estimating and guessing. I would recommend that you don't even try to compare metrics from different Debit Credit reports; only try to understand Debit Credit results within the context of a single report.

Debit Credit is not representative and it is scalable. Even though TPS and price per TPS are usually reported, it is difficult to compare numbers from different reports.

SPEC BENCHMARK SUITE RELEASE 1.0 (currently technical)
SPEC COMMERCIAL

The Systems Performance Evaluation Cooperative (SPEC), whose members include HP, IBM, and DEC, is an organization whose "primary goal is to create and distribute application-like benchmarks that fairly measure the performance of computer systems" [GRAY].

The SPEC benchmark suite is a collection of "10 benchmarks drawn from real-world applications, including CASE, EDA, MCAE, and other scientific and engineering areas" [SPEC WINTER 90]. These benchmarks are all cpu intensive, with either an integer or a floating point emphasis. All ten benchmarks do little I/O. The SPEC benchmark suite is an improvement over the previously available technical benchmarks, and should further improve as SPEC claims "Future releases will focus more intensively on other aspects of system performance, memory, I/O, graphics, networking, multiuser and commercial applications" [SPEC WINTER 90]. The SPEC Benchmark Suite, like the technical benchmarks previously discussed, is not representative of a commercial workload and thus is not a useful benchmark suite for commercial systems.

No commercial benchmark has yet been added to the SPEC Benchmark Suite, but a subcommittee has been formed to focus on commercial applications.

AIM III
NEAL NELSON BUSINESS BENCHMARK

AIM Technology (the AIM III benchmark) and NEAL NELSON & ASSOCIATES (the Neal Nelson Business Benchmark) are both private firms that will provide reports of their benchmarks on a variety of systems. The characteristics that I will discuss are similar for both benchmarks, so the following applies to both.

The benchmarks run on UNIX or UNIX-like operating systems. They are multi-user, synthetic benchmarks with no data base access. Neither benchmark performs any terminal I/O (this is an important issue if the benchmarks are ever run on MPE since terminal I/O is used by the dispatcher to adjust process priority). These two benchmarks are not representative of a normal commercial workload and are thus not very useful [FARNED, SAITO].

TECHNICAL BENCHMARKS:

LINPACK
WHETSTONE
DHRYSTONE
SPICE
etc.

These benchmarks were written to use resources that are of interest to technical applications. Each of these benchmarks is not representative of a typical commercial workload for one or more of the following reasons. The benchmark is cpu intensive, spends more time in user code than in system code, doesn't measure the impact of increasing the size of the cpu cache, is not multi-user, is not multi-tasking, is not multi-processing, does not heavily load the I/O system, and arithmetic operations are primarily floating point. Most of these benchmarks are designed to measure the PEAK performance, not the TYPICAL performance, of a portion of the hardware (e.g. the alu or the floating point processor). (SPICE is a notable exception, being representative of a specific technical workload.) Thus these benchmarks are extremely unlikely to be informative for a commercial environment.

IX. BENCHMARKS USED BY THE HEWLETT-PACKARD COMMERCIAL SYSTEMS DIVISION (proprietary and industry standard)

TPC BENCHMARK A

The TPC Benchmark A was described in the Industry Standard Benchmarks section.

TPC Benchmark A is a simple, non-representative, interactive, on-line transaction processing (OLTP) benchmark. Both the construction of the benchmark and the reporting of the metrics are strictly defined [TPCA].

A very important requirement of the TPC Benchmark A standard covers data integrity. The benchmark is required to meet the "ACIDity test" - the properties of Atomicity, Consistency, Isolation, and Durability. The definitions of ACID are:

Atomicity: "The system under test must guarantee that transactions are atomic; the system will either perform all individual operations on the data, or will assure that no partially completed operations leave any effects on the data."

Consistency: "Consistency is the property of the application that requires any execution of a transaction to take the database from one consistent state to another."

Isolation: "Operations of concurrent transactions must yield results which are indistinguishable from the results which would be obtained by forcing each transaction to be serially executed to completion in some order. This property is commonly call serializability."

Durability: "The tested system must guarantee the ability to preserve the effects of committed transactions and insure database consistency after recovery from any one of [a specified list of failures]."

The ACIDity test ensures an application environment that is representative for on-line transaction processing systems; system performance can not be increased through sacrificing data integrity.

The standard also defines the benchmark workload, including data set size per TPS, number of users per TPS, minimum average transaction cycle time (think time + transaction duration), and transaction arrival rate. All users in the benchmark access a common set of data files, which may be a distributed data base. The transaction includes reading from a terminal, updating a large dataset, a fairly small dataset, and a very small dataset, writing the transaction to a history file, committing the transaction, then writing to the terminal.

Reporting standards include what to report and how to report it. The required "full disclosure report" covers many areas. The information required to replicate the benchmark must be furnished. This includes items such as application source, settings for customer-tunable parameters, and system configuration. The method used to determine repeatability must be documented. The results of the ACIDity tests must describe how the requirements were met. The terminal connect method must be specified as either wide area network or local connect. The throughput of local connect is specified as "tpsA-Local" and the throughput of WAN connect is specified as "tpsA-Wide". The standard clearly states "These two metrics are NOT comparable with each other". An independent audit of the benchmark results is highly recommended though not required.

The primary metrics required in the full disclosure report are Transactions Per Second and system price. Supporting metrics include response time data. The maximum throughput (TPS) is constrained by a response time limit; 90% of all transactions started and completed during the measurement window must have a response time of less than two seconds. A frequency distribution graph of response times is required. The average and 90th percentile response times are indicated on the graph. The maximum and average response time must also be reported. A second graph plots 90th percentile response time versus TPS. Values for response time at 50%, 80%, and 100% of reported throughput are required.

The above paragraphs are meant to convey a flavor of the standard, not to be a complete description. If you are interested in more details, please read the standard.

TPC Benchmark A is not representative and it is scalable. Even though it is not representative, it is currently the best commercial, OLTP benchmark available for comparing the systems of different vendors. It provides a framework in which the metrics of competing vendors can be reasonably compared, unlike its predecessor, Debit Credit. Another contribution of TPC Benchmark A is that there are two goals that it emphasizes: maximize TPS and minimize price per TPS.

MANUFACTURING

The manufacturing benchmark is an interactive manufacturing application. The benchmark functions include accounts payable, order management, and manufacturing inventory. These three functions accounted for over 80% of the work performed on the application in an actual customer environment. Each user in the benchmark accesses one of the three functional areas, with a ratio of 2 accounts payable users : 1 order management user : 4 manufacturing inventory users. The workload, including think times, is captured from a production system. The application accesses twelve TurboIMAGE data bases, as well as KSAM and flat files. Terminal I/O is via a mixture of the VPLUS screen handler and character mode I/O.

The manufacturing benchmark is both representative and scalable, which makes it a useful benchmark to compare different HP3000 systems. Since there is no comparable benchmark on other platforms, is not useful for comparing the HP3000 to other computer families.

MRP

The MRP benchmark is a batch Material Requirements Planning application. The first program reads a transactions from a file, processes the transactions (adds, updates, and deletes entries in a TurboIMAGE data base), analyses the data (supply & demand analysis), creates transactions, and builds a file for later reporting purposes. A second program, run as a child process of the first, deletes the transactions processed. It calls a third party subroutine which is in compatibility mode to read and sort large amounts of data. The user programs are compiled in native mode.

The MRP benchmark is both representative and scalable, which makes it a useful benchmark to compare different HP3000 systems. Since there is no comparable benchmark on other platforms, it is not useful for comparing the HP3000 to other computer families.

SC33

SC33 is a multi-component, representative, commercial OLTP benchmark. It is different from most representative benchmarks in that it is based on a group of actual systems that have similar resource usage patterns instead of being based on a type of application, such as inventory management. It accesses three TurboIMAGE data bases and ten flat files, as well as doing terminal I/O. The workload consists of short interactive, long interactive, and batch components.

The short interactive component is composed of one or more users, with each user performing a fixed ratio of several commands in a random order. The average short interactive transaction consists of 6 logical TurboIMAGE reads, 2.8 logical TurboIMAGE writes, 6 flat files reads, 1.2 flat file writes, 1 terminal read, and 5 terminal writes. TurboIMAGE I/Os are achieved through a mixture of calls to the DBGET, DBFIND, DBUPDATE, and DBPUT intrinsics.

The long interactive component is composed of one user performing a fixed ratio of several commands in a random order. The average long interactive transaction consists of 700 logical TurboIMAGE reads, 240 logical TurboIMAGE write, 750 flat file reads, 600 flat file writes, one terminal read, and 28 terminal writes. Most of the TurboIMAGE reads are sequential reads through a TurboIMAGE detail dataset.

The batch component is composed of a single batch job which runs for the duration of the benchmark. The batch user executes a mix of three commands in a random order. The average batch command consists of about 25 flat file I/Os and 50 TurboIMAGE I/Os.

The SC33 benchmark is both representative and scalable, which makes it a useful benchmark to compare different HP3000 systems. Since there is no comparable benchmark on other platforms, it is not useful for comparing the HP3000 to other computer families.

DEBIT CREDIT

Debit Credit has already been discussed in the Industry Standard Benchmarks section. Remember, Debit Credit is not really standardized; it is useful for relative positioning of systems within a single benchmark report (as in the performance briefs). Debit Credit is not representative and it is scalable. Even though TPS and price per TPS are usually reported, it is difficult to compare numbers from different reports.

X. OTHER PROPRIETARY BENCHMARKS

RAMP-C

RAMP-C is a synthetic, interactive, OLTP benchmark that IBM has used to benchmark both its own systems and its competitors' systems. I will contrast some of the aspects of RAMP-C with TPC Benchmark A since IBM has used RAMP-C to compare its computers to its competitors'.

The official RAMP-C description is in an IBM Internal Use Only document so the description which follows may not be complete [IBM].

The integrity constraints of RAMP-C are much weaker than TPC Benchmark A, thus less representative of an OLTP environment. The only portion of the ACIDity requirements that RAMP-C specifies is Isolation, or concurrency control.

RAMP-C "is designed for a flat file system like IBM's VSAM or DEC's RMS" [IBM] but IBM has chosen to compare FLAT FILE benchmarks to DATA BASE benchmarks. The file system used for some benchmarks include VSAM for the IBM 9370 & 4381, AS400 relational, HP3000 IMAGE, and VAX RMS.

Each user running the benchmark application accesses five different files. Each copy of the four data files are accessed by up to 5 users and each copy of the the transaction log file is accessed by up to 25 users. Two of the files are accessed sequentially and three are accessed randomly. The file access model of RAMP-C is somewhat like many separate, small applications which don't share data, as opposed to the TPC Benchmark A model which is somewhat like a single application with common file data shared by all users.

There are four different application programs in RAMP-C, referred to as Class 1, Class 2, Class 3, and Class 4. They vary in complexity, with Class 1 being the least complex and Class 4 the most complex. For every five users in a benchmark, one will run a Class 1 program, one will run a Class 2 program, two will run a Class 3 program, and one will run a Class 4 program. Within a benchmark run, the think time for each class of program is fixed, such that the ratio of total transactions completed in the benchmark is 35% (+/- 2) Class 1, 25% (+/- 2) Class 2, 30% (+/- 2) Class 3, and 10% (+/- 1) Class 4.

The RAMP-C description specifies that the benchmark report will include a bar chart showing the throughput of all transaction types combined at 70% system capacity (or 70% CPU utilization). It also requires a line chart showing average response time for all transaction types combined against throughput. IBM has also published graphs depicting 90th percentile response time for all transaction types combined against throughput.

It is not clear from the published material that I have examined whether the RAMP-C benchmark is representative. The RAMP-C benchmark is scalable. It is useful for comparisons within a single product family. RAMP-C results should not be used to compare systems in different product lines if the implementation of the benchmark varies significantly on the different systems (e.g. flat file system vs. data base management system). The characteristics of RAMP-C are somewhat like a set of many small applications, not a large, shared data application.

XI. SUMMARY OF BENCHMARK CHARACTERISTICS

MANUFACTURING

Interactive OLTP
HP proprietary
Representative (based on third party application)
Comparative benchmark within HP3000 product line
Heterogeneous transactions (simple to extremely complex)
Heterogeneous scripts (three classes of script, each contains a mix of transactions)
Exercises terminal I/O (character mode and VPLUS), flat file I/O, and data base (IMAGE & KSAM)
Integrity requirements: Isolation, Durability

Evaluation Point: knee of the curve

MRP

Batch
HP proprietary
Representative
Comparative benchmark within the HP3000 product line
Exercises flat files, TurboIMAGE, native mode, compatibility mode
Integrity requirements: Isolation, Durability
Evaluation Point: knee of the curve

RAMP-C

Interactive OLTP
IBM proprietary
Synthetic (representativeness not clear)
Comparative benchmark within single product line, not across product lines
Heterogeneous transactions (simple to slightly complex)
Homogeneous scripts (four classes of script,each contains one transaction type)
Exercises terminal I/O and either flat file I/O or data base
Integrity requirements: Isolation
Evaluation Point: 70 percent cpu utilization

SC33

Primarily interactive OLTP, with a batch component
HP Proprietary
Representative of one class of MPE/V customer workload
Comparative within the HP3000 product line
Heterogeneous transactions
Heterogeneous scripts (three classes of script, each script within a class contains the same percentage mix of transactions in a random order)
Exercises terminal I/O, flat file I/O, and TurboIMAGE data base
Integrity requirements: Isolation, Durability
Evaluation Point: knee of the curve

TPC BENCHMARK A

Simple interactive OLTP
Industry standard (TPC consists of thirty four member companies)
Not representative
Comparative benchmark across vendors - tight specifications and reporting requirements
Homogeneous transactions
Homogeneous scripts
Exercises terminal I/O and data base (update intensive DB services emphasis)
Integrity requirements: Atomicity, Consistency, Isolation, Durability
Evaluation Point: response time tight specifications and reporting requirements

XII. WHAT A BENCHMARK REPORT IS GOOD FOR

With what I've said up to this point you should understand the meaning of benchmark metrics, the attributes of a "good" benchmark, why benchmarks are created and run, and how useful some specific benchmarks are. Now, if you have the results of one of these benchmarks, what are some of the purposes they can be used for?

SIZING OF A NEW SYSTEM

If you are planning to purchase a system you can use a comparative benchmark to help size the system appropriately for your needs. The benchmark is either simple and non-representative or you were able to get a benchmark of your actual application. If you are fortunate enough to have a benchmark of your actual workload then you can pick a system that should meet your needs fairly closely and you can move on to the next (nearly inevitable) step of forecasting the growth of your application. But it is more likely that the benchmark results you possess are from a simple, non-representative benchmark. The simplicity is a result of building a benchmark that will provide performance metrics that can be compared accurately across product lines. Of course, the down side to this accuracy is that meaningfulness is compromised. Don't let the apparent accuracy of the comparison tempt you to assume that the meaning of the metric also has an inherent accuracy. (For example, the TPC Benchmark A metric of TPS must be reported to three significant digits but if your intended use of the system is software development the TPC Benchmark A metric is not a valid predictor of the performance of your workload, no matter how accurately it is reported.) Most benchmark reports will carry a caveat to remind you of this. In Hewlett-Packard reports this ranges from the casual "your mileage may vary" to "actual customer system performance WILL vary based on workload and other factors". An example of this caveat from another vendor is: "The following apply to the interpretation of the RAMP-C Benchmark Results. Each customer's requirements are unique, and no single benchmark workload can adequately represent the broad diversity of commercial transaction processing applications. Response times, throughputs, and the relative positions of the systems may vary under a different workload. It would be inappropriate to use the results of the RAMP-C benchmark as a basis for system sizing" [IBM 6/1/89].

The message I wish to convey is that *a non-representative benchmark can be used as an aid to sizing competitive systems.* The results should be used to decide what systems are to be considered, not as the final factor to choose between two finalists. Be sure not to lose sight of the other factors in the selection process (e.g. growth path, reliability, support, application development environment and tools, and operation and system management).

SIZING FOR GROWTH

There are many ways of planning application growth and adding or upgrading systems to match the growth. If you are experiencing high growth rates, you may be able to purchase new, larger systems (or upgrades) on a regular basis, knowing that you will need the extra capacity soon (and even if you purchase a system more powerful than you need now you will need the extra power a month from now). The easiest method is to wait until your system is overloaded and then add capacity (but be sure to keep your resume up to date if you use this approach). The third method is to measure current performance on a regular basis, be aware of expected application growth, and use the performance grapevine (the experiences of members of your local users group, articles in trade magazines, etc.) to set your expectations of how much your system's performance will increase if it is upgraded. Your sizing decisions will be improved if you also consult published benchmark results to set your performance expectations. A fourth method is to use software modeling tools (also known as capacity planning tools) to predict the impact of application growth on system performance and the performance or capacity increase resulting from an upgrade or the purchase of a new system. A fifth method is to benchmark your workload, with current and increased levels of usage. These alternatives are listed in order of increasing cost and increasing accuracy (though this ordering does not always hold true). Most people find the third and/or the fourth method to be the most effective way to size their systems. Since this is a paper on benchmarking, I will ignore capacity planning tools and just comment on the use of benchmarks to set performance expectations.

Sizing for growth may be similar to sizing of a new system but there is an important difference. Currently, the benchmarks used to compare different product lines are both simple and non-representative. When sizing a new or upgraded system the variety of benchmarks to consult is larger; you can use both non-representative and representative benchmarks to aid in your sizing decision.

The use of non-representative benchmarks has already been discussed (they are a useful aid but their precision is relatively low). Representative benchmarks have the potential to increase the level of precision if they are representative of your workload. Remember all the caveats mentioned in the previous section ("your mileage may vary", etc.). The more like your application a benchmark is, the less strongly you have to apply the caveats, but don't ever forget or underestimate the importance of the caveats. The following table shows why the fact that a benchmark is representative of something does not necessarily make it more precise.

Series Z vs. Series Y
Relative Performance Summary

Application Type	Interactive Benchmarks	Rel. Perf. Ratio Series Z : Series Y
Accounting/Financial	1	1.6 - 2.35
Banking	2	1.7 - 1.9
Inventory Management	1	3.1
Manufacturing	2	1.6 - 1.9
Marketing Research	2	1.75 - 2.0
Materials Management	1	2.6
Library	1	2.1
Stock Broker	1	2.1
Overall	11	1.6 - 3.1 avg = 2.07

source: [BOSCH]

It is obvious from the table that certain workloads show a greater performance improvement when moved from a Series Y system to a Series Z system. If the Inventory Management benchmark is representative of your workload that you can expect a much larger performance improvement upon moving from Series Y to Series Z than if the Marketing Research benchmark is representative of your workload. The increased precision of a representative benchmark comes from the fact that it approximates your specific workload. (In other words, if you have a representative benchmark that is not representative of your workload then you must treat the results as those of a non-representative benchmark.)

The discussion above makes it clear that making sure a benchmark is representative of your workload is a key issue. There are at least two ways of comparing your workload to a benchmark's workload. First, what resources or system features does each use? Second, what limits the performance (throughput and/or response time) of each workload?

There are two factors of resource usage. First, what resources are being used? Cpu? Disc I/O? (If so, what kind? Flat file? Message file? TurboIMAGE data base? Allbase/SQL data base?

Mapped file?) Local or remote file access? Character mode terminal I/O or block mode? Second, how are the resources being used? Is the data base access primarily read access, write access, update access, or delete access? Is the file access random or sequential? Trying to compare these two factors between your workload and a benchmark's workload for all possible resources will require a lot of time and effort. Don't devote the rest of your life to discovering the intricacies of your workload, instead focus on the major resources used.

The second aspect of the workloads you need to consider is what the bottlenecks are. For example, if you have an update intensive data base application (with all users accessing the same data sets in the same data base), the bottleneck might be caused by data set lock conflicts. A benchmark that consists of all users updating a common data base will be more representative of your workload than a benchmark where each set of five users share a common data base. If, on the other hand, your workload consists of many discrete applications, each with its own database, then the second benchmark might be more representative. The common bottlenecks to be aware of are cpu, disc, memory, and shared lock conflicts.

A class of bottlenecks that are frequently overlooked is processing constraints, bottlenecks that are the result of the work flow of an application. For example, if the window available for non-interactive processing is ten hours long, your nightly backup takes seven hours, and your nightly batch process runs four hours then your bottleneck is backup more than it is batch processing. Maybe you should be more interested in the results of a benchmark that compares different backup alternatives and configurations than in the results of a benchmark that compares batch processing performance. Some other non-traditional bottlenecks that may be important to you are application initialization time, single threading of processes because a resource can not be shared (e.g. a tape drive or a dedicated data communications circuit), and time to recover after a system failure or system abort.

In summary, if you are concerned with performance in a specific computing environment instead of across product lines you are more likely to be able to be able to find a benchmark report that is representative of your system. First you have to measure the performance and workload on your own system, then you have to compare your system to available benchmarks. You may be able to get benchmark results from your vendor, from trade journals, and from fellow members of your users' group. Even though you probably won't find a benchmark that is exactly like your applications, the representative benchmark will be much more informative than a non-representative benchmark. Remember the differences between your workload and the benchmark workload and how this affects applying the benchmark results to your environment. And finally, don't forget there are other methods of capacity planning. For instance, if you need a high level of precision then a capacity planning tool will probably provide better results.

XIII. HOW ARE METRICS MISINTERPRETED?

Metrics are not mysterious but it it easy to reach incorrect conclusions if you do not analyze the meaning of a metric sufficiently. For instance, if you are told that system A has a throughput of 100 TPS and system B has a throughput of 30 TPS then which system has better performance? System A obviously has better throughput, right? Suppose you then ask your informant the source of this data and he tells you that he has a Debit Credit benchmark for system A and a RAMP-C benchmark for system B. Which system is better? Now you have insufficient information to decide which system is better because Debit Credit TPS and RAMP-C TPS are not comparable. This example is a little contrived, but more realistic examples often follow the same pattern; information is interpreted out of context, a piece of information is missing, or an incorrect assumption is made. I will discuss some more realistic examples below.

As I have mentioned before, if a benchmark is not representative of your workload and environment then the usefulness of the benchmark is lessened. A good example of this was the table in the "sizing for growth" section which showed that different benchmarks provided a large range of the relative performance of two systems. The margin of error for the performance metrics is increased when the benchmark is not representative. The second important example is that if the environment of the benchmark is not cost effective (e.g. excessive amounts of memory and disc are configured on the test system) then the performance metrics will not be representative of the performance an average system would achieve.

Another issue related to representativeness is whether the benchmark exercised the features that are relevant to you. Not only the obvious examples, like whether cpu, disc, memory, and lock contention is representative, but also more easily overlooked features like whether the benchmark is a single user test on a multi-user system and whether the on-line benchmark included a batch component competing for resources. A final easily overlooked feature is HOW the features are used. For example, does your usage of the data base (usually visible as how much I/O is performed) require the integrity of ACID? Does the benchmark reflect the extra overhead of ACID?

When examining a metric, be sure to examine all the measures of the metric that are available. (This is the point that was made in the section "REPORTING METRICS".) The average value of a metric is extremely easy to compare across benchmarks but also looking at the standard deviation of the metric may provide a more valid comparison. Examining a plot of the actual data may provide even more useful insights to the performance behavior of the systems.

Next, what metrics are important to you? Common metrics are:

average response time
maximum response time
variability of response time
throughput
elapsed time
annual downtime
hours each day that the system is available to users
cost

If your primary application is comprised of PCs sending background task requests to your system and checking back for the results several hours later then you will be more interested in throughput than with response time. If your primary application is an OLTP system then you will be more interested in response time. The important point to remember is to make sure that the benchmark does not negatively impact one metric in an effort to optimize another. Don't just look at the metric you care about most, look at all the metrics.

Graphs of metrics often are in the form of a line graph where the metric is on the Y axis and the number of users is on the X axis. The immediate temptation when looking at the graph is to presume that the system will support the same number of users for an actual application. Do not do this!! Even a representative benchmark will not accurately predict the number of users a system can support. Recall (from the section "sizing for growth") that benchmarks can be one of the factors that you use to calculate the correct size for your system but it does not directly specify the number of users that can use the system. Capacity planning tools will generally provide a more accurate result.

When a total amount of work performed metric, such as TPS, is reported, it is constrained by some factor. That factor can be that one of the system resources is a bottleneck or an artificial constraint. The typical artificial constraints are:

A response time cutoff (e.g. 90% of all transactions must have a response time less than two seconds).

A cpu busy cutoff (e.g. the cpu must be 70% busy).
The knee of a curve has been reached (e.g. the knee is the point at which the slope of the response time vs. load curve sharply increases or the point at which the slope of the throughput vs. load curve sharply decreases).

The artificial constraints are introduced because otherwise the total work performed would be measured at a point where most users would consider the system to have bad performance (e.g. a response time of five minutes for a simple transaction).

One argument advanced against response time constraints is that a badly chosen cutoff would "encourage significant tuning efforts since a slight decrease in overall response time can result in a large increase in throughput. Consequently the systems may be tuned to a degree not normally found in customer environments." [LASKOWSKI]. Of course this same argument also applies to the other types of artificial constraint.

If a benchmark report includes a graph of the distribution of the values of the metric that is used as the constraint then it is a simple task to verify that the cutoff point is beyond the knee of the curve and thus has not encouraged excessive tuning efforts. By this measure, the TPC Benchmark A cutoff point is an example that has proven to be well chosen, given empirical results of all benchmarks published as of now (5/1/90) [HP].

A bigger problem with an artificial constraint is that it may have no basis in reality. For instance, if a TPC Benchmark A transaction has a response time of two seconds then will your application have adequate response time on the same configuration?

One final problem with choosing cpu as the cutoff constraint is that the cutoff metric does not directly affect the user. The user of a system doesn't care if the cpu is 50% busy or 90% busy; the user cares that his response time is short and consistent or that his throughput is high. The value of cpu busy that indicates a fully loaded system is an architecture dependent value that may vary significantly, even within a product line. If you read a benchmark report (such as RAMP-C) that uses a cpu cutoff constraint to limit throughput then I recommend that you refer to the response time vs. users and throughput vs. users graphs to calculate throughput yourself based either on the knee of one or both of the curves, or on a response time cutoff constraint.

XIV. SUMMARY

Benchmark results are used to compare the performance behavior of software on two or more systems. The usefulness of a particular benchmark is affected by many factors, including what the intended use is, whether the benchmark has the attributes necessary for that use, how the benchmark was built and executed, and the reporting method of the benchmark metrics. Different users of benchmark data have different requirements. For example:

For multi-vendor comparisons, use industry standard benchmarks.
For comparisons within a product line, use representative benchmarks first and non-representative benchmarks second.
For capacity planning, use benchmarks of your specific workload first and representative benchmarks second.

There is always variation between the performance characteristics of any particular benchmark and any particular application, just as there is variation between the performance characteristics of any two particular applications. The more that the benchmark is representative of your application, the closer the performance behavior of the benchmark and your application.

Finally, to take full advantage of benchmark results, examine the report thoroughly. Don't just read the headlines, be sure to look over the details also. Don't jump to quick conclusions about what the data is telling you. When you have formulated your conclusion, go back and double check the assumptions you made and the logic you used to reach the conclusions.

For the careful reader, there is a wealth of information in benchmark reports.

XV. TRADEMARKS & COPYRIGHTS

SPEC is a trademark of Systems Performance Evaluation Cooperative.

TPC Benchmark A is a trademark of the Transaction Processing Performance Council.

UNIX is a trademark of AT&T

XVI. BIBLIOGRAPHY

[BOSCH] Bosch, Bill, Using pn2 Data To Close Series 900 Sales, Performance News Notes (HP Internal Use Only publication), Volume 7, Number 3, March 1989

[FARNED] Farned, Laura, HP 9000 Neal Nelson Results, Performance News Notes (HP Internal Use Only publication), Volume 7, Number 2, February 1989

[GRAY] Gray, Larry, SPEC and Other Benchmarking Groups Seek Common Goals, SPEC Newsletter, Volume 2, Issue 1, Winter 1990

[HP] Hewlett-Packard Company, Commercial Systems Division, HP 3000 Series 960, HP 3000 Series 949, HP 3000 Series 922 TPC Benchmark A Full Disclosure Report, January 1990

[IBM] International Business Machines, Report Of Arthur Anderson & Co. RAMP-C Benchmark, June 1, 1989, IBM Publication No. GG22-9455-00

[JACKSON] Jackson, Terry, Interpreting and Quoting MIPS, Performance News Notes (HP Internal Use Only publication), Volume 8, Number 1, January, 1990

[LASKOWSKI] Laskowski, John, SPEC Developing Criteria For Evaluating Commercial Applications Benchmarks, SPEC Newsletter, Volume 2, Issue 1, Winter 1990

[SAITO] Saito, Glenn, AIM III On The Series 800, Performance News Notes (HP Internal Use Only publication), Volume 7, Number 2, February 1989

[SPEC 1/18/90] SPEC, SPEC Commercial Interactive Workload, January 18, 1990

[SPEC WINTER 90] SPEC, SPEC Benchmark Suite Release 1.0, SPEC Newsletter, Volume 2, Issue 1, Winter 1990

[TPCA] TPCA, TPC Benchmark A Standard Specification, November 10, 1989

[WEBSTER] Webster's Ninth New Collegiate Dictionary, 1988

5/19/90

RELATIONAL VS. NETWORK
THE DILEMMA OF THE 90'S

Shelley Meiklejohn
Price Waterhouse
950 17th Street, Suite 2600
Denver, Colorado 80202
(303) 893-8100

With the advent of relational data bases now being available on the HP 3000, many of us are considering the option of moving off TurboImage onto one of the relational offerings. There are many differences between the two platforms, and alot of things to consider before making the move. This paper will compare and contrast the two technically, provide some "typical" applications which are suited to each, and finally, provide you, the MIS professional, with a set of criteria to help you make your decision.

What is a relational data base?

First off, relational technology is not new. ORACLE began development over 10 years ago. It just seems new to us on HP 3000's because it's just become available to us in the last couple years. Relational data bases grew out of the need for more flexibility in data storage, maintenance, and retrieval. A relational data base is a collection of "flat" files, which may, or may not, be indexed by a B-tree. Indexes are simply there for performance - and they DO make a difference. Users may retrieve by any field in a RDBMS, even if it's not a key.

Another key element is that the RDBMS does the navigation through the data. Users tell a "front-end" which items they want, and the RDBMS finds the most optimal path by invoking an "optimizer". The optimizer's (which all RDBMS's use) main job is to retrieve data efficiently, but there are other functions it does which will be discussed later.

Since there are so many RDBMS vendors on the market, it became increasingly necessary to standardize. The standardization took effect at the inquiry language level, and the most prominent language available in the industry (and HP3000's) is SQL (Structured Query Language). With Image all accesses to data base are via intrinsics, but with relational DB's, they use SQL either interactively or programmatically. SQL is a set of commands which is fairly easy (however, quite wordy) to use. There are 4 commands used the most. They are:

1. SELECT - reads data from one or nine tables
2. UPDATE - updates data in a table
3. DELETE - deletes data in a table
4. INSERT - adds records to a table

These are for the "user". SQL also handles all the creation and maintenance of the actual physical and logical structures of the data base - the typical DBA functions.

All programs which access a SQL data base must be pre-processed. Image programmers compile, prep (or link) and run. With SQL, programmers must pre-process, compile, link and run. The pre-process step does three major functions. They are:

- Comment out the SQL code so the compile won't abort;
- Optimize the SQL calls and store them in a table in the data base; and
- Add additional code to your program to allow it to run (even though programmers embed SQL code, it is converted to intrinsic calls by the pre-processor).

A relational database is a collection of tables, defined by columns and rows. These tables are defined by the DBA via SQL code, and contain related data, i.e., employees. These tables may be joined together by a "like" item for a broader access, or a limited access within a table (for security purposes) may be defined. Relational data bases allow these multi-table access or limited table access through views. These are defined via SQL statements by a DBA-type function.

What is a Network data base?

A network data base is a multi-level structure which separates data on a one-to-many relationship. TurboImage/3000 is a 2-level network, but there are multiple levels also available. (HPImage which runs on the HP/9000 is an example.) In Image there are master sets and detail sets. The masters typically serve as indexes into the data. The programmer must know the structure and will navigate his/her self through the data in the most optimal way possible.

As was stated before, there are system-level intrinsics which allow access to the data. There is a different interface for DBA type activities, DBUTIL, and yet a third interface, for the data base definition, DBSCHEMA. All three of the pieces make up the Image environment.

How do they compare?

Let's look at a sample data base in both environments. (See Exhibit A)

With Image, the indexes (or pointers) are stored in the data, but with Relational, they are stored external in B-tree index files.

In order to compare the two environments, lets look at some general areas. 1. Multi-User Consideration; 2. Conversion Issues; 3. 4GL Interfaces; and 4. Application Software.

1. Multi-User Consideration

Since most data bases run in a multi-user environment today, we should examine the impact of each design. Image has had its weaknesses, possibly the biggest two were single control block (fixed by Turbo) and serial reads. Since Image is fairly limited in it's inherent access methods (key value, record #, chained) it forces many serial reads in order to get the flexibility necessary today. Relationals offer a combination of B-tree and hashed access in order to try to provide the flexibility, but not at the price of serial reads. They still serial read on occasion, but those times are getting less and less.

In multi-users access, locking becomes very important. Image allows programmers to lock the database, the data set, an entry or set of related entries. This is fairly optimal in a network data base. Relational still has some room for improvement. When HP/SQL first came out, it had two locking options, both imposed. Allow read while read, or exclusive write. This led to some interesting performance issues. Vendors have improved on this some, but they still have a long way to go.

Security is another area of great difference. Image has always embedded passwords in the DBSCHEMA. The access level is then defined on an item or set level, defining read or write access. Relationals, on the other hand rely on the MPE accounting structure, (particularly HP/SQL) and define security via SQL commands by the DBA. It goes like this: When you logon to the 3000, you supply "user.account, (group)". HP/SQL takes the user and concatenates it to the account after an "@" sign (i.e., :Hello MGR.SALES becomes MGR@SALES to HP/SQL). All security is then "GRANT"ed on an entity and activity basis. This can get very complex defining all the access necessary to the users.

Relational data bases also provide an additional function which Image does not (without a front-end). These are a logical structure, called Views. Views allow the DBA to define additional structures, from the physical tables, without adding additional data storage. As stated, these views are a very powerful part of the security, as you can define a view which limits access on a table, not only by column, but also by row. For example, access could be allowed to our Emp-table to MGR@SALES on columns "no", "name", "level", "emp-no" where "level is equal to 5". The DBA could also join the two tables by "no" to allow access across the two, thereby setting up a view.

Image does not support views, and it is difficult to get the multi-set access, without forcing serial reads.

2. Conversion Issues

When you are comparing the two platforms, you must be concerned with what it would take to convert from Image to Relational. There are some specific areas which need to be considered. They are:

- Security

  As already mentioned, the security schemes are quite different. With Image, users supply a password, and with HP/SQL it comes from logon user and account. Where this would usually become an issue, is if you are currently allowing users to all logon the same, i.e., USER.MRKT, but have supplied them with unique passwords. If you convert to HP/SQL you would have to change your account to provide users with unique logons, and then grant them the proper access on a table-by-table basis.

- Data Conversion

  In a benchmark I did recently, I had to set up two scenarios, one in Image, and one in HP/SQL. To create the HP/SQL data base, I needed the data from the Image data base copied into the HP/SQL tables. From what I could find, there is no HP supported utility to do this. (The Query report facility may work.) I used Datadex from Dynamic Information Systems Corp., which will unload a set to a flat file. Then, HP/SQL has a import facility which will load a flat file into a table. There was one more glich here in that the import facility will only load ASCII data. I had some binary fields in my Image data base which had to be converted to ASCII before I could load them in.

Since design is a very important performance consideration, I don't recommend doing an exact set to table conversion. The data base should be re-designed to fit the relational model, which will make the conversion even more challenging.

- Personnel

The group affected most will be the programmers. As stated, the coding necessary to access the data base is completely different. There is also some additional linkage areas defined for the data. Below is an example of an intrinsic call to our sample data base, and the same access in SQL (in COBOL).

```
    (previous "moves")
    CALL "DBGET" using BASE, DSET, MODE, STATUS, LIST,
    BUFFER, ARG.
vs.
    EXEC.SQL
    SELECT * INTO :NO
                  :NAME,
                  :LEVEL,
                  :SALARY,
                  :EMPNO,
    WHERE EMP-NO = :EMP-NO
    END-EXEC.
```

All programs accessing Image data bases would need to be changed to issue SQL statements instead of intrinsic calls.

(* means <u>all</u> columns)

- Administration

With any RDBMS, you will need a full-time DBA. There are so many changing parameters, and maintenance requirements, someone needs to control the entire environment. Since relational data base offers so much flexibility, like file size changes, structural changes, a full-time person is required. This may be an addition to your staff, or shifted responsibilities.

3. <u>4GL Interfaces</u>

Since so many shops are using 4GL's it is necessary to fully evaluate whether your 4GL of choice supports, or will support, the RDBMS you have chosen. This may become part of your decision of whether to do the conversion and how quickly. Most popular 4GL's are moving to one of the popular RDBMS's.

4. <u>Application Software</u>

If you are running a third party software package, it will be necessary for them to support the RDBMS. Once again, it may be in their future plans, and may effect your decision of when and where to convert.

To summarize, below are some points of why to stay network, and why go RDBMS.

<u>Why Stay Network?</u>

Currently, Image is still the best performer in OLTP (on-line transaction processing) application. RDBMS's are catching up, but Image is still #1. You may want to do a benchmark to confirm this on your application, however.

If you are running a MPE/V based machine, stay with Image. There just aren't enough resources on those machines for the relationals to perform, unless you strictly want the DBA flexibility feature. Also, alot of the RDBMS's in the market aren't, and won't be, supported on those platforms.

If your application software isn't going relational, and you're satisfied with it, stay with Image.

If money is an issue, you should probably stay with Image. If you consider all the training, time, and hardware (if needed), this can be a very costly endeavor.

Most HP/3000 programmers know Image, and are probably fairly proficient at its use.

Also, today there is more application software available on Image. This will probably change, but once again, may effect your timing.

Why Go RDBMS?

One of the most important offerings of the RDBMS is its flexibility - both user and DBA. Users have the ability to look at data any way they want, and from anywhere. There's no restriction on number of keys, or where the data's located. The more sophisticated the user the more access they will have. They can also use the "LIKE" verb in the select statement which will match a pattern anywhere in the column. Relational data bases also support boolean and relational logic, i.e., "and", "or", "not", " > ", " < ", etc.

The DBA functions are alot easier on an RDBMS. Within SQL, even programmatically, you can detect if you're out of space, and add additional, if necessary. There is a SQL statement to add a column to a table, on-line. Indexes can be added at any time, or removed at any time. Since indexes do increase performance, this is a very powerful feature. (I added an index to 111,000 row table, it only took 3 minutes on a HP/3000/935.)

Portability may be a requirement for your shop. If so, RDBMS's are the way to go. ORACLE runs on at least 80 hardware platforms, and the other leading RDBMS's are on all the popular machines. Supposedly, SQL is SQL, so you could have one program accessing different SQL data bases on different machines.

Since SQL is fast becoming a standard, there are some reasons to standardize in your shop. SQL - literate professionals are going to get easier and easier to find, and many people find this a very easy language to use. Since SQL is used programmatically and interactively, you only have one access language to learn.

SQL is a very powerful language which will allow almost any access to the data base.

As you move up the RISC machines, your SQL applications will continue to run even better. The more CPU, the faster the RDBMS's will run.

Client/server application are being targeted to the RDBMS's. INGRES is a leader in this technology, already. If PC's are a large part of your systems, RDBMS's should be evaluated as to how they may enhance them.

Which Applications are Suited to which DB?

In general, any application requiring DBA flexibility should go relational, i.e., sales & marketing, trend analysis. General ledgers are good because of all the serial processing (relationals perform as well as a network on a serial read).

OLTP application should stay Image. Some typical applications as order processing, point-of-sale, other financials. Currently, the major strength Image has is performance on keyed reads.

A Checklist:

Below are some questions you should ask before making a change. Weight the questions appropriately for your environment to help determine the fit.

1. Does your current 4GL support?
2. Does your application software support?
3. Does it run on your hardware?
4. Does it support client/server capabilities?
5. Does it have a forms interface?
6. Does it have an interactive interface?
7. Does it support remote data bases?

8. Is it standard SQL?
9. Does it support your third GL?
10. Does it have an import facility (to load your existing data)?
11. Does it have a report facility?
12. Will the vendor benchmark?
13. Reference sites?
14. Can you modify your MPE accounting structure (if necessary)?
15. Does the vendor provide conversion services?
16. Do you have a DBA available?

Summary

As I hope is obvious, both platforms have alot to offer on the HP/3000. The network data base is still a very powerful performer, while the relational data base offers some new flexibility. My feeling is that most of us will end up running a little of both, depending on the application.

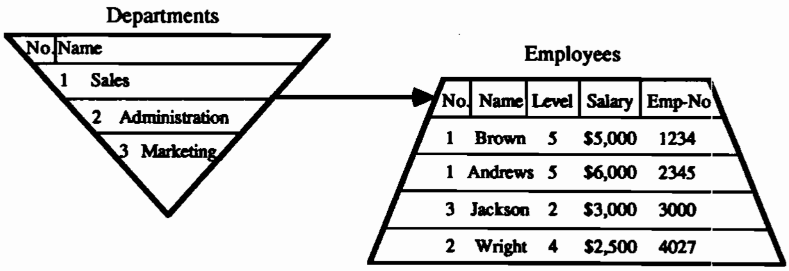
This paper introduced you to relational technology, compared and contrasted it to network data bases, gave you some reasons for staying or switching, discussed some typical application for each, and finally outlined some general questions you may want to ask yourself before making a switch.

The good news is we have a choice. This hasn't always been the case. Also, I don't think you can go wrong. They are both good options and the more you know about each will certainly make the decision easier.

EXHIBIT A

SAMPLE DATA BASE IN IMAGE AND HP/SQL

IMAGE



Departments

No.	Name
1	Sales
2	Administration
3	Marketing

Employees

No.	Name	Level	Salary	Emp-No
1	Brown	5	$5,000	1234
1	Andrews	5	$6,000	2345
3	Jackson	2	$3,000	3000
2	Wright	4	$2,500	4027

HP/SQL

Department Table

No.	Name
1	Sales
2	Administration
3	Marketing

Emp-Table

No.	Name	Level	Salary	Emp-No
1	Brown	5	$5,000	1234
1	Andrews	5	$6,000	2345
3	Jackson	2	$3,000	3000
2	Wright	4	$2,500	4027

Effective Systems Development Techniques for the 1990s

Douglas Colter
Infocentre Corporation
7420 Airport Road
Suite 201
Mississauga, Ontario
Canada L4T 4E5
(416) 678-1841

Introduction:

In today's competitive markets executives are continually searching for ways to help their organization remain competitive. The demand for innovative system designs enabling them to make proactive decisions has increased to the point where the software development process is regarded as an obstacle rather than a progressive step. As we enter the 1990's MIS departments must find ways to become more responsive to the changing requirements that are challenging them.

Advances in hardware and software have provided both challenges and new opportunities for systems development. Therefore, advances in the way we develop systems designed for executive decision making will only occur if we adopt methodologies which take advantage of the strengths offered by these new technologies.

Fourth Generation Language software has been prevalent in the HP3000 community for over ten years and are used with varying degrees of success. This paper reviews how the traditional methodologies once worked, but now fail; and how 4GL shortened the development process; and finally, why they are not measuring up to their expected potential. This paper also will examine the techniques which can be applied to achieve the full potential of 4GL where the user takes an active participatory role, and the DP staff while getting off to a faster start, can remain flexible and responsive to changes in user requirements.

Traditional Stuctured Methodology

Let's begin by examining how we approached things in the past. Structured design methodologies gained acceptance and developed a large following during the late seventies and early eighties. This methodology systemized the application development process. It typically involves the formal definition of phases, into which every development project is subdivided. Each phase is completed, one after the other in a linear fashion. Signoffs, intended to signify user acceptance (compliance maybe) mark the conclusion of each phase. Throughout each phase, the project team members work in strict adherence to pre-defined standards and procedures.

The phases followed in a development project vary from one installation to the next, but typically follow the same pattern:

Phase	Description
1.	Requirements Definition
2.	Analysis
3.	Physical Design
4.	Programming
5.	Testing and Documentation
6.	Implementation.

Usually there is little or no overlap between the phases. The outputs generated by the first three phases entails a lot of paperwork. It's not until the programming phase gets underway that the computer system plays an active role, and in this phase the role consists of 3GL compiles and creation of Database and file structures.

The design of this methodology is based on several assumptions:

1) That the process of developing application systems is linear, and can be approached in a multi-phased linear fashion.

2) Users know all (or at least most) of their needs up front and can effectively communicate those needs to DP analysts.

3) The users needs won't change during the development process.

This methodology enjoys varying degrees of success, depending on: the length of the project, the DP sophistication of the user, and the seasoning of the DP staff.

In reality the progression of the development project through the phases tends not to be linear but rather somewhat iterative. Users typically have difficulty knowing/expressing all of their system requirements up front. Analysts discover missing pieces of the puzzle when the design phase is underway. Programmers stumble across design inadequacies or deficiencies during the programming phase. Each of these roadblocks in turn send the project temporarily back to an earlier phase.

Even without these regular setbacks, the development process takes a long time because it is labour intensive. The user interviews and fact finding undertaken in the early stages drags through seemingly endless meetings. Data flow diagrams take an eternity to produce, while the programming staff spends many hours with a text editor writing and fixing programs in a Third Generation language. By the time the system is ready for implementation it's already partially obsolete due to the constantly changing business environment. To add to our problems, we are chasing a moving target because we live in a changing and inperfect world.

Adopting Fourth Generation Software

Adopting Fourth Generation software presents some differences in the application development tools. Let's take a few minutes to identify the major differences between Third and Fourth Generation software.

As mentioned above, with Third Generation software the process is labour intensive. The programmer works with a text editor to create/maintain his source file(s). The programmer works at a relatively low level concerning himself with tasks such as: routine data editing, screen handling, accessing the file system, creating/maintaining the Data structures and so on.

Fourth Generation software should provide you with a complete application development environment. The software is more than just a language, complemented by other modules which provide:

- automated generation of 4GL code and database structures;
- relational access to Database files;
- automated production of user documentation;
- end user report generation;
- micro-mainframe networking capability;
- production of presentation graphics; and
- client server cooperative processing.

The language component lying at the heart of the software environment, is non procedural, action driven enabling the programmer to work at a much higher level than is possible with a traditional Third Generation language. Typically with a 4GL, the programmer no longer concerns himself with coding the lower level routines handling Screen I/O, File/Database I/O, data editing and validation basic reporting and so on. With a non procedural language, the programmers job is to provide the language processor with the specification of what is to be done, leaving the *hows* (detailed processing logic) the language processor. Built into the language processor are standard pre-determined algorithms dictating the method by which the typical business oriented DP functions are to be processed. These typical functions are:

- menus;
- data entry/update/inquiry Screens;

* reports; and
* batch update routines.

The language component aside, the development environment provided by the Fourth Generation software opens up some new frontiers. With a system designer tool and a documentation module it becomes obvious that we can let the computer do a lot of the work for us. The process can become far less labour intensive. Other modules open up the possibility of more flexible Database access and end user computing, thereby relieving the DP staff of servicing the ad hoc reporting needs of the application. This end user computing capability can be extended through the micro-mainframe networking link, porting the users data to the PC world where the end user can further process it with the adopted PC software.

Current Methods Using 4GL

The kind of systems we are developing today are much more advanced than those developed a few years ago. Now senior management is becoming computer literate and demanding access to the corporate information. However, we cannot always predict exactly how they will access the data because their requirements are constantly changing. We must respond by adopting development methodologies which take advantage of the sophisticated hardware and software available and yet be flexible enough so that we can remain *proactive.*

If Fourth Generation Software proposes to assist us in developing systems faster, why then are do the actual gains not quite measure up to the achievements and benefits anticipated?

Julia Markham of HP Software Engineering Systems Division answers this best by saying "without training and properly adopted methodologies, today's software tools are used to about 15 to 20 percent of their potential." The full potential of 4GL software is inhibited due to the lack of user involvement. Users are typically involved very little in the development of a computer application. They provide a description of current problems, and limited detail regarding solutions and requests for new applications. Once the system requirements have been fully analysed and documented, the analyst provides a somewhat detailed, and often tentative solution in the form of a design specification. Once the specification has been approved by the users, communication between user and analyst is minimal. The users often want to be left alone to tackle the stack of work still sitting in the in basket, trusting the DP staff will take full ownership of the project.

Since 4GL accommodates fast and easy development of software applications, the DP staff presents the 'finished' system to the user prior to implementation. The user might make a few suggestions regarding cosmetic changes, but they are still busy with their daily chores and are anxious to have the new system installed so all dreams may be realized.

The fact is their problems have only just begun. It is often not until post implementation that the user becomes committed to the success of the system. At this time problems are identified and have to be corrected. Using 4GL, software modifications are implemented with a moderate amount of effort. The DP staff typically spends considerable time during post implementation making changes to design inadequacies.

Even with the added speed of 4GL many development teams are only utilizing this technology to automate most of the programming functions. They are not using it as a tool to assist the analyst during the design phase, where problems might be identified.

The New Approach

The new approach to development bring both the user and programmer into the design phase. Active participation by all parties during this phase will ensure a better fit to the user requirements. MIS must take a proactive stance where the objective is to learn more about the business problems so as to be better equipped to face the corporate challenges.

Although MIS departments have had varying degrees of success implementing systems using a 4GL, we are still not taking advantage of all the benefits offered by these tools. Fourth Generation languages allow designers to build prototype systems in the early stages of development. DP people have always maintained that users do not know what they want so we usually tell them what they can get. This is why users have rarely participated during the development of an application. This is both true and understandable.

Engineers build models of cars and architects build models of buildings before vast resources are spent completing the projects. Information system users should not be expected to know exactly what they want just as car manufacturers or builders do not, until they can see a model. If in the early phases of system development we can build a prototype and let both the users and designers *experience* the system, then both may see and avoid problems that would have been very expensive to correct later on.

Prototyping is a technique used to clarify the functional requirements of a system and to ensure the system meets the users' needs. James Martin said "The process is inherently flexible and recognizes that the system specification cannot usually be written down very precisely in advance of development. Instead, the process advocates a dialogue between the user and the analyst that leads to a better understanding by both parties."

Although coding applications using 4GL software reduces the development time, there still can be some tedious programming. For instance screen programs all have the same general structure where data fields have screen coordinates, prompting labels, plus other attributes must be defined. If we are to develop systems in a more timely manner, we must also automate the task of coding 4GL syntax. Advanced 4GL environments have a tool or workbench module which will assist these manual tasks. The application of these tools, often referred to as CASE, automates the generation of all system components including database structures. Such a tool ideally would be capable of implementing any data structure supported on the HP3000 so we can build a prototype model around any logical database.

The Analyst and the User

The analyst must undertake more effective systems analysis and design. The use of a 4GL does not replace the need for solid up front requirements definition and analysis. If anything it makes the successful completion of the early phase more critical because of the collapsed time of the programming phase.

During the design phase, the analyst carefully conceptualizes a 'solution'. The analyst would spend time to build a model using a designer tool intended for the automation of screen and report development, including automated development of the data structures. The screen programs would have all the characteristics in appearance and general functionality of the eventual production programs, but without the detail processing that is normally included.

The kind of functions the analyst should build into an application would include:

* logical/physical data structure
* screen and report appearance;
* simple data checking, initialization, and validation;
* method of screen and report access; and
* data retrieval stategy;

While screen programs are developed, the powerful capabilities of the design tool would also implicitly define the database structure. Optionally, the analyst would at any time have explicit control over the definition of the database structure. The design tool should automatically check to ensure there is consistancy between the application and the database structure. The tool should also provide the means to make rapid changes to the database structure as the prototype application evolves. The analyst works very fast because the design tool generates all 4GL syntax free of errors.

After all of the principal screen programs have been developed in this way, the analyst may bring the user in to *experience* the application. At this time there would be an exchange of ideas regarding the kind of changes which may be applied to the design. The user should expect to meet with the designer in this manner, for at least an hour, three to five times per week. Depending on the size of the project, this process could take a few weeks. Each time, the application comes closer to a better fit so that the final stages of development easily fall into place.

Another possible approach is to develop the model application on the PC using the powerful capabilities of 4GL which, in many cases, have been extended into the PC environment. The benefit here is having a dedicated work station where the application can be developed and tested rapidly. The application can be developed entirely on the PC including database structures using clone database management systems cloned from the HP3000 environment. Optional real time access to the HP3000 resources is an other possibilty. At any point in time, the design may be ported to the HP3000 for continued development or implementation.

The Analyst and the Programmer

Not only should the user be brought into the development phase, but the programmer should be brought into this phase much sooner as well. When both the programmer and user become involved, the opportunity for misunderstandings and misspecifications is reduced. ·The programmer is traditionally brought in once the detail specifications have been documented. This method stifles the programmer's creativity and opportunity to contribute to the design while changes are inexpensive.

The analyst develops screen programs which access primary files such as the customer, product, order-header, and order-detail in an order proccessing application. With a design tool, both the 4GL syntax and data structure are generated implicitly. At any time during development, the designer has the option to override default data structures simply by entering the desired item names and attributes. Similarily, the definition of search items and paths to manual or automatic masters can also be implemented in the same manner.

With design information stored and maintained in an application dictionary or repository, the design can be modified easily. The design tool checks the specification for logical consistancy. Furthermore, after the application has been implemented, its entire life, from a development and maintenance point of view may exist in the dictionary. This means the design tool can be used for on going system maintenance.

The analyst should work very closely with the programmer. Rather than providing detailed specifications in writing, the analyst and programmer hold regular meetings where details are defined and noted in point form by the programmer. The problem with having detailed specs in writing is the programmer would follow the instructions to the letter. However, informal meetings with the analyst would encourage an open discussion where the programmer might ask why, for example, a status field is updated with a particular value. This technique reduces the chance of developing inadequacies which might otherwise have been missed.

The programmer should work closely with the user at times during the development phase. Every time the prototype is tested by a user, the programmer observes the comments and questions from both the analyst and user. This way the programmer gets a better understanding of the requirements and may have valuable suggestions.

Usually the analyst can provide enough detail for the user to see without having to resort to entering syntax. However, when the model calls for complicated functionality which the user must see, the programmer can enter the required amount of syntax to demonstrate a particular feature. Sometimes crucial report programs require additional transaction processing logic, such as the logic found in invoice print programs, where several files must be updated simultaneous to printing. Here too, the programmer is called in to develop the key syntax necessary to demonstrate the functionality of the report program.

Once the model design has been *experienced* by the user, and approved, these programs can be passed over to the programmer. This stage of development does not require the close attention of the user. Detail processing is implemented using an appropriate blend of procedural 4GL syntax, including:

* programmatic data initialization and validation;
* additional transaction processing;
* well defined and optimized database transactions;
* application security;
* customized help text; and
* multiple user capability.

Finally, detailed user manuals are produced automatically using the documentation module of the 4GL environment.

Implement the Methodology

There is no foolproof cookbook approach to follow that will ensure success in prototyping with Fourth Generation software. There are however some issues to address in order to significantly increase the odds of success. In no particular order, they are:

1) Develop an expertise in using the tools. All team players are involved in this education:

   Analysts:

   Must learn how to effectively use the modules that facilitate prototyping. They must also be aware of the underlying functioning of the language processor.

   Programmers:

   Must have a thorough working knowledge of their Database Management System/File System as well as all facets of the 4GL. Programmers in your organization will grow to be elevated to a higher level of DP professional. They will become *Application System Specialists* shifting from lower level programming tasks, to all facets of application system development and functioning.

Users:

Must become acquainted with the prototyping process. This includes learning the rules (by which we will develop systems), the terminology and techniques.

2) Undertake more effective analysis and system design. As pointed out above, the use of a 4GL does not replace the need for solid up front requirements definition and analysis. If anything it makes the successful completion of the early phase more critical because of the collapsed time of the programming phase. The Fourth Generation software can be effectively used as a design aid. Implement your design with a prototype. Use the prototype to test your design hypotheses. Correct the design flaws early in the process. This will avoid any misunderstandings or misspecifications which could result in many hours of costly programming. Your users' active involvment in the development and testing of the prototype will quality assure the accuracy of the design.

3) Encourage changes in the development methodology by:

   a) Placing importance on the methodology. An endorsement by senior management asking for productivity to improve by a factor of 10 to 20 percent has more impact than a similar decree by middle management.

   b) Motivation is improved when the methodology is successful. Success breads success.

   c) Transfers of new technology. Take advantage of the new hardware that compliments a productive 4GL development environment.

4) Foster Innovation in the MIS organization by:

   a) New ideas via college hires. People who are relatively new to the computing world bring fresh ideas, and would not have to break old habits.

b) Continuous training. Allow for about 5 percent or ten days per year for formal training.

c) Set high expectations. People always produce more when they are challenged.

d) Tolerance for failure. Let developers take chances using new methods. Being wrong is only a short delay in being right!

5) Put the necessary resources in place.

People:

Users freed from their regular duties so they can undertake their active participatory role. Analysts and programmers with the required skillset.

Machine:

When an effective 4GL development methodology has been properly implemented, the demand for computer software applications will increase as management sees that MIS is no longer a necessary hurdle towards achieving the company's goals. The need for additional computing resources should be viewed as a sign of success as more applications are loaded onto the system.

The prototyping team should have hassle-free ready access to work stations, printers and any other necessary peripheral devices.

The use of personal computers is a viable alternative to sharing the resources of mini computers during application development.

Software:

The software requirements are: System Design tool that enables fast development of a prototype including the database structure in addition to the procesing. Behind the system designer module must be a powerful, full functional language processor. These core modules should be rounded out with an automatic documentation tool, presentation Graphics system, and end user reporting module in order to provide a complete Fourth Generation development environment.

Bear in mind that none of these resources will stay in place without the complete backing of senior management.

Summary:

A methodology adapted to the strengths of the new software, coupled with a sound working knowledge of the tools will go a long way towards achieving the anticipated benefits offered by the productivity tools.

We must become just as proactive (or perhaps a little more) as the people who require information from our corporate databases. The user must take an active participatory role in the development process and take ownership of the system right from its inception. Programmers must be elevated to take an active role as well.

Break away from the traditional approach to systems development. After all, executives are breaking away from many of the traditional methods of conducting business... we must follow. Senior management should endorce progressive methods and be tollerant to failure. You will have more success stories than imagined.

Performance Measurement Of Multiple CPU's: *A Tale of 200+ Systems*

by

Mark Stevens (Steve) Willis
Senior Programmer/Analyst

of

AmericanAirlines®

SABRE Computer Services Division
4000 N. Mingo Road, Tulsa, OK 74116
918-832-5604

Introduction

It *is* the best of times *and* the worst of times. American Airlines is in the process of transitioning from a virtual HP3000 non-entity to one of the ten largest HP3000 users in the world. This presents some special challenges in a corporation that has historically been exclusively IBM and DEC turf with a central-site-management-oriented mindset. This paper will explain how AA addressed the issues related to selecting and implementing a system performance measurement product for over 200 HP3000's world-wide.

AA's HP3000 Environment

American Airlines is in the process of completely revamping how employees access corporate computing resources. This internal revolution is taking place under the auspices of a project named "InterAAct." InterAAct provides an OpenView-like "virtual-backplane" for the user, whereby he or she can easily access a plethora of computing environments from a single intelligent workstation running under HP NewWave.

One aspect of the InterAAct project is the use of HP3000's to facilitate an international E-mail network using a heavily-customized version of HPDeskManager. AA will have 3000's in virtually every airport in every country in the world to which we fly, as well as in all administrative and maintenance facilities worldwide. At most sites, there will be only one CPU, either a 955 or a 922, depending upon the size of the local user community. At sites that have a sufficiently large user population, bigger and/or multiple CPU's will be installed. All CPU's are networked via private X.25 WAN (AANET), with multi-CPU sites being interconnected locally by HP ThickLAN. All interactive sessions on the HP3000's except the local console are Virtual Terminal (VT) sessions accessing via Novell Token-Ring LAN. The X.25, ThickLAN, and VT session interfaces are all handled by HP's NS/3000 network communications product. Central operational control of the remote CPU's is accomplished through HP's Integrated Network Control System (INCS). INCS allows virtually complete console control of a remote HP3000, including starting/stopping systems and the taking memory dumps.

Due to manpower and other considerations, it was determined early in the project to centrally-manage all of the HP3000's through four departments that correspond to functional entities within existing IBM and DEC shops. One department handles all system management functions for all CPU's, another handles operations, another data security, and the last one (of which I am part) handles system performance and capacity planning.

In general terms, most of the challenges AA face derive from the requirement for centralized control and support of widely-scattered CPU's that were never intended to be managed in that manner. Most of those issues are beyond the scope of this paper—the one I will address today is that of how we took HP's performance measurement product, LaserRX, and adapted it to this type of environment by automating the collection of performance data.

American's HP3000 Performance Software Requirements

AA had several criteria for selection of an HP3000 performance product. They are listed below:

- The ability to measure utilization all of the standard system resources (CPU, memory, disc, response time, etc.) with minimal system resources required for data collection.
- No requirement for AA to write its own software to analyze and plot the data (as in OPT/3000).
- Minimal programming effort required to automate the tasks related to performing that analysis.
- The performance data should be graphed in an easily-interpreted form.
- The ability to both analyze summary data over one-month intervals and to perform *ad hoc* performance troubleshooting when reports of poor response time might occur.
- Sufficient user-friendliness to allow non-technical clerical personnel to produce the graphs.
- Possess a filtering mechanism whereby we could manage-by-exception and graph only those systems whose performance fell outside of the minimum acceptable performance (MAP) limits for an AA HP3000.
- Possess performance trend analysis capabilities to allow us to predict future performance bottlenecks and thereby pro-actively manage the acquisition and installation of performance-related system upgrades and additional CPU's.
- Present a minimal load to the production CPU's when crunching the performance data for graphing.

HP LaserRX vs. AA Requirements

The version of LaserRX that we first received initially met most of our selection criteria. It measures most of the requisite system resources (everything except disc space usage and fragmentation), it provides its own data crunching software, it delivers the desired statistical granularity, its easy to use, and the graphic results are quite easy to interpret. Another strong point was that, because LaserRX is PC-based, it presents minimal overhead cost for the HP3000 system.

On the down side, the current version of LaserRX does not provide a session/job-to-load correlation, no trend analysis could be performed without manually exporting the data into a spreadsheet because the log files are encrypted, and no scripting and filtering capabilities exist to facilitate batch processing. While HP allows the use of thresholds to filter out all performance events that fall *within* MAP standards, these thresholds cause an unacceptable (to AA, at least) loss of statistical data. We want to have *all* events logged and then filter out the graphing of systems that fall within the MAP limits during batch processing so that only "bad" systems are reported.

American went back to HP and requested that LaserRX be enhanced to remedy the short-comings listed. HP agreed to provide in the next release of LaserRX (currently slated for 4Q 1990) the scripting mechanism and the job/session-to-load correlation. The filtering mechanism will appear in the release after that. They have also recently released a separate product (the RXForecast add-on module for LaserRX) to perform performance trend analysis from LaserRX log files, and they have stated that they are actively considering the development and release of disc space utilization and fragmentation functionality.

Automation Strategy

We had several motivations to automate collection and analysis of the performance data. The primary two were:

1. Manpower quantity and quality requirements would be significantly lower. Manually running the graphs for 1-10 systems is one thing—performing the same task for 200+ CPU's is another story altogether. And, as previously mentioned, we wanted the software to be simple enough for a non-technical user to be able to use—complete automation is the ultimate in user-friendliness.
2. Lower CPU and WAN overhead. Manually graphing these systems would take place during prime time business hours, thus creating additional CPU and network overhead when interactive users are accessing the 3000's.

One of our programming goals was to minimize the amount of custom 3GL code required and to shift as many tasks as possible to jobstreams using "MPE Programming" techniques, as described by Eugene Volokh in his INTEREX paper by the same name.

The following tasks were to be performed by the automation software:

- Automatic installation/localization of SCOPE and related software
- Automatic extraction of previous month's data at month-end
- Centralize physical (CPU) location of performance data
- Automatic job re-launches caused by DSLINE/DSCOPY failures
- Automatic DSCOPY failure reporting
- Automate examination of performance extracts to locate performance anomalies
- Automate graphing of those anomalous nodes

Automation Task Overview

The tasks requisite to successful automation of the data collection process is as follows:

1. **Automatic monthly performance data extraction**
   a. A scheduled job launches on remote CPU on 1st day of month at midnight local time

   b. Previous month's beginning/end dates determined within job by program developed in-house

   c. Performance data for previous month is extracted.

   d. Run programs for disc space utilization and fragmentation data files.
2. **Automatic retrieval of extracted data**
   a. A scheduled job launches on REPO on 2nd day of month at midnight CST/CDT

   b. Produce/run individual jobstreams to retrieve remote data files

   c. Report successful completion of retrieval
3. **Automatic reporting of retrieval failures**
   a. A scheduled job launches on REPO on 4th day of month at midnight CST/CDT

   b. Job compares list of nodes for which jobs were created with list of nodes successfully retrieved

   c. Resulting list of nodes with failed retrievals sent to appropriate department via HPDeskManager

4. **Automatic graphing functionality** (in future LaserRX release)
   a. LaserRX user initiates batch-scan against extracted data files filtering with MAP thresholds
   b. LaserRX plots only those network nodes exceeding MAP thresholds
   c. Resulting graphs are distributed to appropriate departments and action is taken as needed.
5. **Quarterly trend analysis**
   a. LaserRX user initiates trend analysis on accumulated performance data extractions.
   b. Results are distributed and action is taken as needed.

Automation Software Descriptions

EXTRACT Software

1. **EXTRDATE** (Pascal Program)
   uses the system clock to determine the dates for the first and last days of the previous month. It writes the results into the appropriate records in a file that is to be used as $STDIN for the LaserRX EXTRACT program.
2. **EXTRACTJ** (Jobstream)
   runs EXTRDATE to set up the $STDIN file for the EXTRACT program and then runs EXTRACT to perform the data extraction and produce the RXLOG file. The job also runs two other utilities, CAPACITY and FREE5, to provide raw data concerning disc space utilization and disc fragmentation. At the conclusion of the job, EXTRACTJ reschedules itself to run on the first day of the following month at midnight.

RETRIEVE Subsystem

1. **JCWSET** (Pascal Program)
   allows a jobstream to re-launch itself automatically a finite number of times if a fatal error occurs within a jobstream. After the the number of tries exceeds the limit, the operator is notified of the failure.
2. **NEWGROUP** (Pascal Program)
   creates the 3 groups required to receive the previous month's data files from all CPU's. One group receives all of the RXLOG files from the remote CPU's, the second receives the disc space utilization file, and the last gets the FREE5 data. The last four characters

of the group names are the year and the month in YYMM format. The group names are also stored in an ASCII file for use in the retrieval jobstreams created by RETRIEVE.

3. **RETRIEVE** (Pascal Program)
   takes jobstream templates (one for DSCOPYing, the other for FCOPYing) and replicates them by inserting the node name, NEWGROUP group names, etc. into the job templates wherever the appropriate place markers appear. If the node being processed is the repository (REPO) node, the FCOPY template is used for the jobstream, otherwise the DSCOPY template is used. RETRIEVE creates a list file of all nodes for which jobs were created (to be used by FAILURES) and a :STREAM command file to be USEd by EDITOR/3000 to launch all of the jobs.

4. **RETRIEVJ** (Jobstream)
   a) REMOTE HELLO to node containing master node list
   b) DSCOPY master node list to REPO
   c) If DSCOPY fails, run JCWSET, reschedule for re-launch in 15 minutes
   d) Attempt and reschedule every 15 minutes until 10 failed attempts, then notify Operations.
   e) Create reception groups with NEWGROUP
   f) Run RETRIEVE to create DSCOPY/FCOPY jobstream, job list, and :STREAM command USE files
   g) Run EDITOR/3000, USE :STREAM command file to launch all jobs
   h) Reschedule to run one month later

5. **DSCOPY/FCOPY Templates** (Jobstreams)
   a) REMOTE HELLO to remote CPU
   b) DSCOPY performance data files
   c) If fails, run JCWSET and reschedule job for re-launch 15 minutes later
   d) Keep attempting and if still fails, repeat rescheduling every 15 minutes until 10 failed attempts
   e) Notify operations and abort after 10 failed attempts
   f) If successful copy, write an empty one-record file in REPORT group with file name equalling the node name

4. **FAILURES** (Pascal Program)
   Compares list of DSCOPY/FCOPY nodes for which DSCOPY/FCOPY jobs were created with :LISTF results obtained from REPORT group of logon account. If job successful, node name removed from job list. Resulting list of job failures saved to ASCII file.
3. **FAILUREJ** (Jobstream)
   a) Run FAILURES to determine failed nodes
   b) Append resulting list to HPDeskManager message header
   c) Send resulting message to appropriate personnel via E-mail

Conclusion

Our automated LaserRX-based system has been in place for 3 months now and runs flawlessly. Multiple CPU support in the performance measurement arena does not have to be a headache, whether a site uses LaserRX or some other performance product. Minimizing support manpower in this type of environment can be achieved with modest effort by a combination of vendor support and in-house programming. Of the two, vendor responsiveness and their willingness to make the enhancements required for your automation scheme a success are, to me, the key elements. At this time, American Airlines is well-pleased with how LaserRX performs and the future of the product here at AA is bright. The only reservation is that Hewlett-Packard delivers what they have promised in the way of future enhancements and add-on products. Since HP does not lightly make commitments to its customers, this should not present any problem.

Acknowledgments

I would like to express our appreciation to the following HP personnel for their valuable contributions to our success in this effort:

HP Dallas Sales Office
Rick Winford, System Engineering
Steve Rutscher, Sales

HP Performance Management Group
Sue Cook, Marketing
Jay Mellman, Marketing
Ray Ventura, R&D