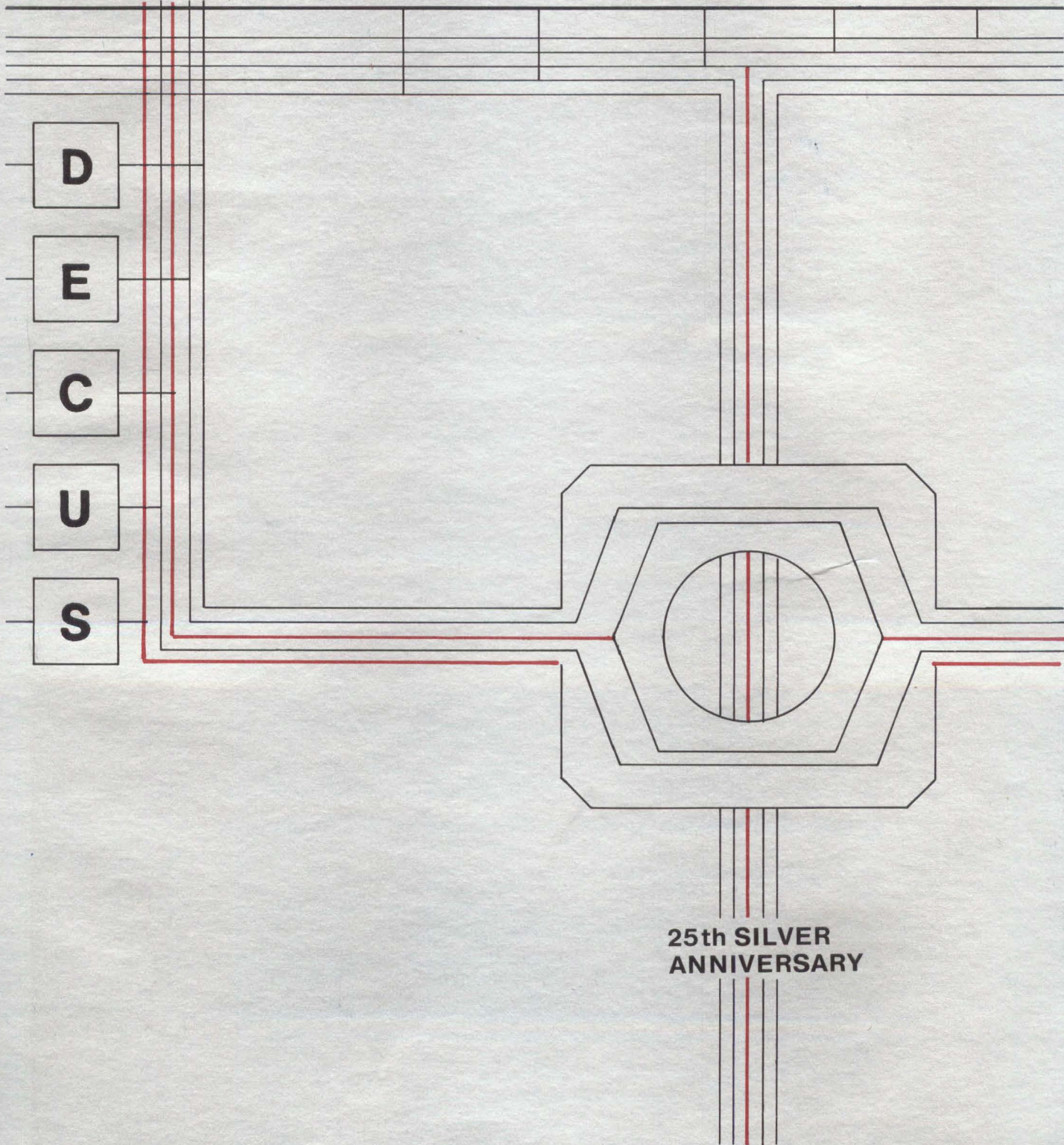


USA

1986 SPRING

PROCEEDINGS OF THE DIGITAL EQUIPMENT COMPUTER USERS SOCIETY

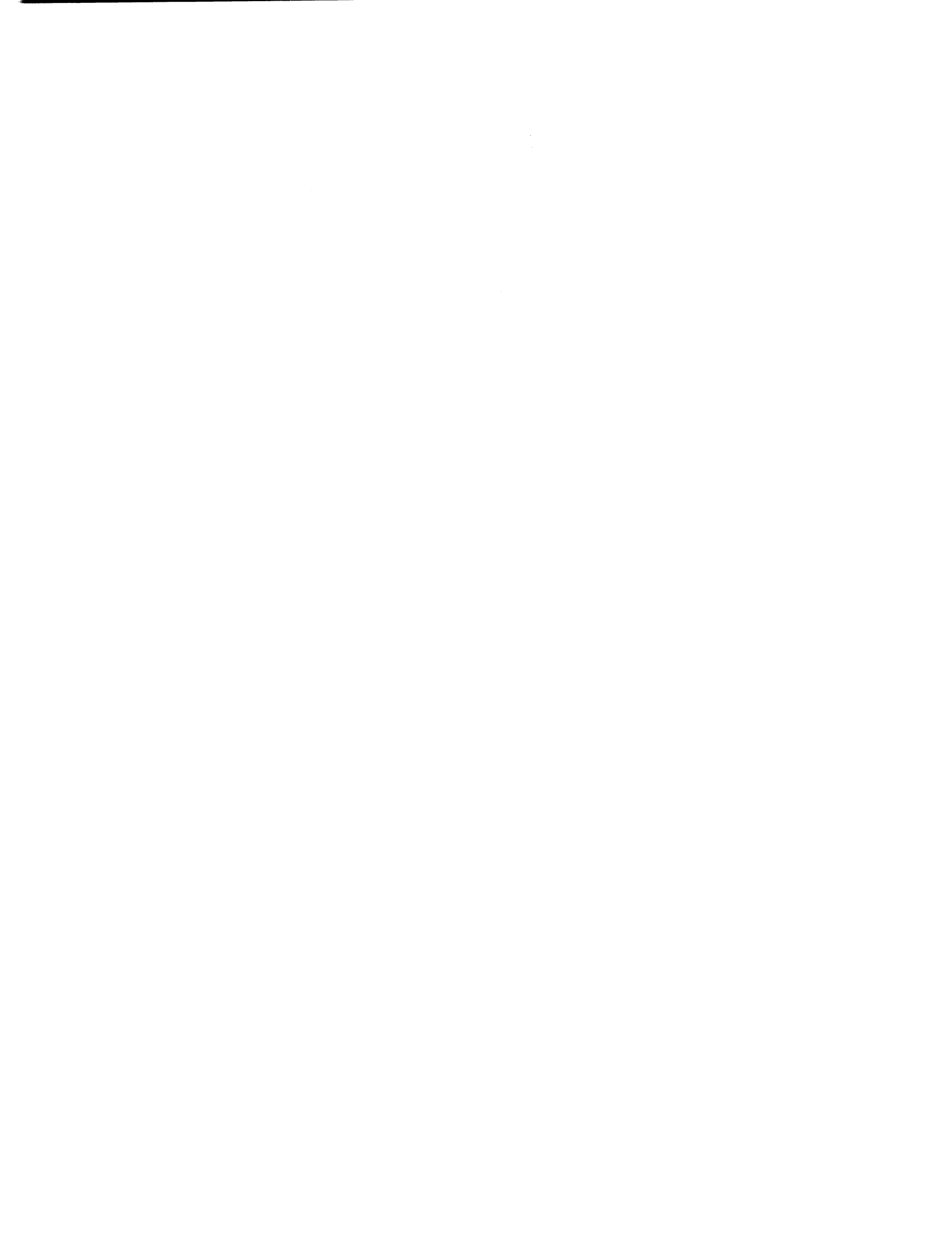




**PROCEEDINGS
OF THE
DIGITAL EQUIPMENT
COMPUTER USERS
SOCIETY**

**Presentations and Reports
USA Spring 1986**

**Dallas Texas
April 28-May 2, 1986**



Printed in the U.S.A.

“The Following are trademarks of Digital Equipment Corporation

ALL-IN-1	LA100	RSX-11 M
BASEWAY	MicroPDP-11	RT-11
DATATRIEVE	Micro Power/Pascal	RX02
DEC	Micro/R SX	TOPS-10
DEClab	MicroVAX	TOPS-20
DECmail	MicroVMS	VAX
DECnet	PDP-11	VAX-11/730 (et al.)
DECpage	PDP-11/23	VAXCluster
DECSYSTEM-10/20	PDP-11/44	VAXELN
DECUS	Professional 350	VAX/VMS
FALCON	Q-bus	VMS
FMS	Rainbow	VT100 (et al.)
HSC50	RSX	WPS-PLUS

Copyright© DECUS and Digital Equipment Corporation 1986
All Rights Reserved

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation or DECUS. Digital Equipment Corporation and DECUS assume no responsibility for any errors that may appear in this document.

Apple II is a trademark of Apple Computer Inc.; UNIX is a trademark of AT&T Bell Laboratories; Scribe is a trademark of Unilogic Ltd.; TeX is a trademark of American Mathematical Society; UniLINK is a trademark of Applitek; HYPERchannel is a trademark of Network Systems Corporation; 68000 is a trademark of Motorola, Inc.; TIway is a trademark of Texas Instruments, Inc.; XNS is a trademark of Xerox Corporation; IBM, PC-XT, BITNET are trademarks of International Business Machines Corporation; TCP/IP is a trademark of Darpa; 32000 is a trademark of National; Cyber 180 is a trademark of Control Data; Modbus is a trademark of Gould, Inc.

FOREWORD

This Proceedings is published by DECUS (Digital Equipment Computer Users Society), a world-wide society of users of computers, computer peripheral equipment and software manufactured by Digital Equipment Corporation. The U. S. Chapter of DECUS has approximately 45,000 active members.

DECUS maintains a library of programs for exchange among members and organizes meetings on local, national and international levels to fulfill its primary functions of advancing the art of computation and providing a means of interchange of information ideas among members. Two major technical symposia are held annually in the United States.

For information on the availability of back issues of Proceedings as well as forthcoming DECUS symposia, contact the following:

DECUS U. S. Chapter
219 Boston Post Road, BP02
Marlboro, Massachusetts 01752-1850

All issues of past Proceedings are available on microfilm from:

University Microfilms International
300 North Zeeb Road
Ann Arbor, MI 48106

PREFACE

This volume of the Proceedings contains papers which were presented at symposia sponsored by the Digital Equipment Computer Users Society during the Spring and Summer of 1986. It includes submissions from the Spring National Symposium and the Northeast DECUS Regional Conference.

The Spring 1986 Symposium was held at the Dallas Convention Center in Dallas, Texas, from April 28 through May 2, 1986. 4605 DECUS members attended the Spring Symposium in Dallas. They took part in birds-of-a-feather sessions, 74 pre-symposium seminars, and over 925 presentations made by both DECUS and Digital. The majority of this volume of the Proceedings is from that symposium.

However, there are two unusual features about this volume that make it a very exciting one for me.

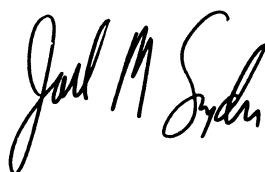
The first starts with the silver cover. This is the silver anniversary of DECUS! To commemorate this special event, I have chosen some excerpts from the first Proceedings of DECUS, published in 1962. The original Preface and Foreword are here, containing references to the PDP-1 and the new and exciting PDP-4. Mr. Walter, the DECUS President, described his pleasure of being in a position to observe the evolution of a "Society which spans such a diversity of on-line processor configurations and uses." I suspect that the present Digital product line would have been entirely inconceivable in 1962. Then, for those of you who have had to wade through the over 1500 sessions at a national symposium lately, comes the program for the Fall, 1962 annual meeting. This takes almost two pages. An attendance list follows; at some point it is mentioned that "all of the DECUS members attended this meeting."

The first paper in this special section is entitled "Translation Problems of a Peripheral Computer in a Multi-Lingual House," from what is now known as Lawrence Livermore Labs. This is the first published paper about IBM-to-Digital interconnection.

The authors note, with astonishing foresight, that "one cannot afford to approach the trivial problem of data conversions with careless contempt." Next the first interactive computer game, Spacewar, is described. Readers familiar with the game "Asteroids" will see its roots. Then, in "A Time-Sharing System for the PDP-1 Computer," an MIT student describes his idea of an operating system. Finally, and my personal favorite, "MACRO, DECAL, and the PDP-1" is a transcript of a discussion on the relative merits of assembly languages versus higher-level languages.

The second unusual feature about this Proceedings is the inclusion of regional conference papers. The Northeast DECUS Regional Conference was held in Boxboro, Massachusetts June 4, 5, and 6. Over 120 people attended 40 sessions and two post-symposium seminars and, of these sessions, 7 were also submitted as papers for the Proceedings. All future issues of the Proceedings will have space allocated for papers from the U. S. DECUS Regional Conferences. I would like to thank Dennis Costello, of that symposium, for helping get those papers into the Proceedings. By adding the regional conference papers, we hope to make the Proceedings more valuable to the membership.

My thanks on behalf of the attendees of the Spring National Symposium go out to Ms. Sandra Traylor and Dr. Jeffrey Jalbert, the DECUS volunteers who led the Symposium Committee. They worked together with DECUS staff members Ms. Nancy Wilga, Ms. Joan Mann, and Ms. Gloria Caputo to put together an exciting, impressive, and informative meeting. The leadership of the entire Symposium Committee is sincerely appreciated. For her special work on the Proceedings, I would also like to thank my colleague, DECUS staff member Ms. Cheryl Smith. In addition, it is important for me to express my thanks to Ms. Judith Arsenault and Mr. Mark Grundler for their continuing support of this work.



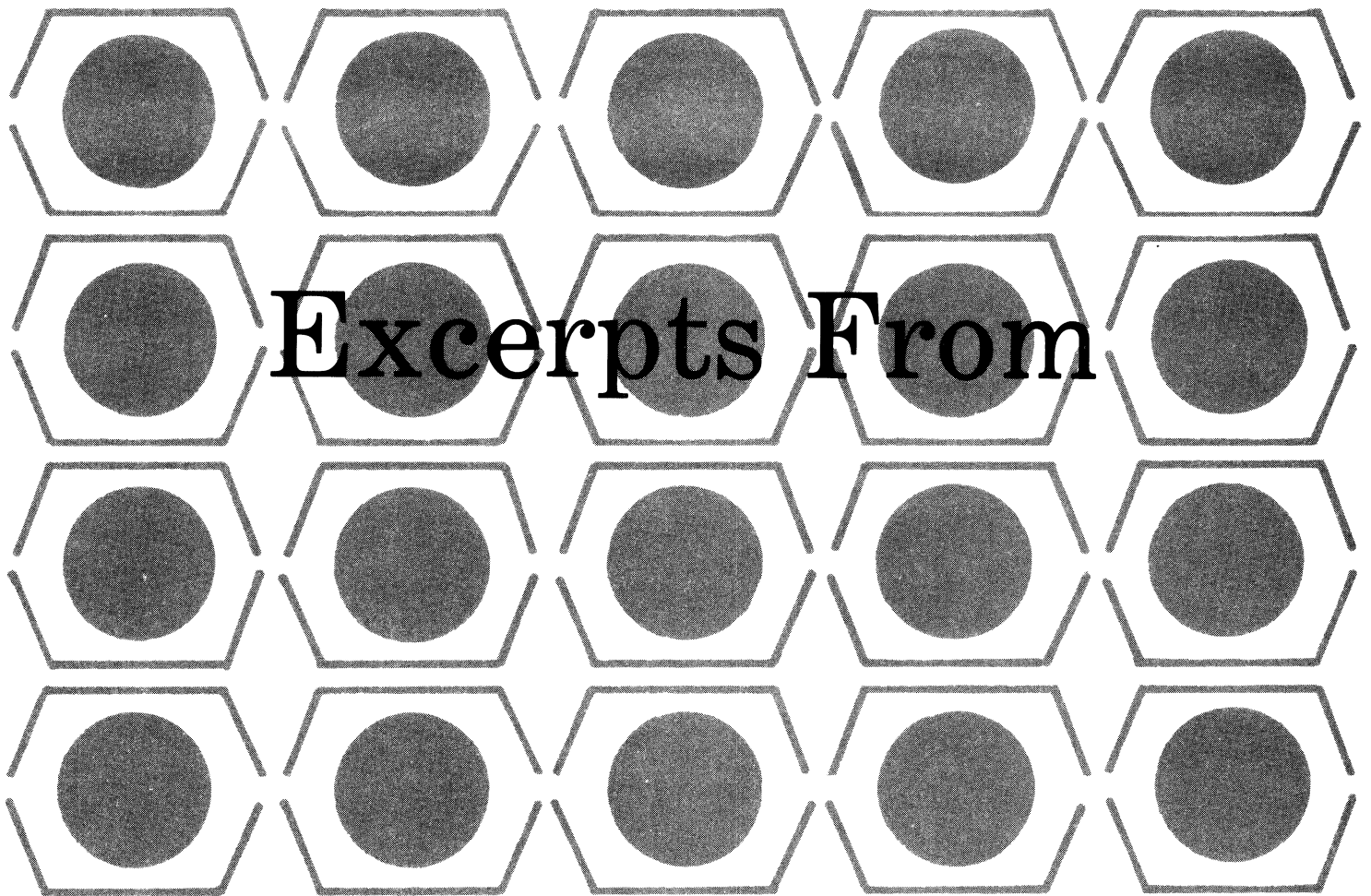
Proceedings Editor
DECUS U. S. Chapter Publications Committee

TABLE OF CONTENTS

ARTICLE	PAGE	ARTICLE	PAGE
25th ANNIVERSARY		A DBMS Performance Evaluation Tool Description and Methodology of Use	
Excerpts from 1962 Proceedings.....	1	Alexander B. Wasilow, Robert L. Ewing, Gary B. Lamont.....	149
SPRINGS 1986 NATIONAL SYMPOSIUM		The Complex Network: A Database Definition Dilemma	
ARTIFICIAL INTELLIGENCE SIG		Mildred D. Lintner, David W. Chilson	155
Development of a VAX Tuner Using OPS5		DATATRIEVE SIG	
Robert A. Small.....	33	DATATRIEVE Novice Questions and Answers	
BUSINESS APPLICATIONS SIG		Joe Gallagher, Chris Wool, Bart Lederman, Larry Jasmann, et. al	161
Project Management in the New Micro/Mini World		Record Definition Tutorial	
Raymond J. Doubleday.....	41	B. Z. Lederman.....	169
Packaged Software and the International Market		The SAR Data Catalog System: An Interface Between the Scientist and the Data System	
Chandan W. Seernani	49	A. A. Pang, A. L. Holmes, J. C. Curlander.....	191
Productivity Tools Improve More Than Productivity		Using DATATRIEVE as a COBOL Code Generator	
Chandan W. Seernani	53	Lynn D. Duncan.....	195
COMMERCIAL LANGUAGES SIG		Commonly Asked DATATRIEVE Questions	
COBOL: An Endangered Species?		Larry Jasmann, et. al	201
Edward W. Woodward	59	DATATRIEVE and RMS	
DATA ACQUISITION, ANALYSIS, RESEARCH, AND CONTROL SIG		Joe Gallagher, Gary Friedman, Bart Lederman	211
Interfacing the Stomach with a Computer: An Automated Analysis of Gastric Electrical Activity		EDUSIG	
O. Guetta, J. Hamilton, J. J. Conklin, A. Dubois.....	73	Test Generation and Course Managing with Digital's Computer Managed Learning Software	
BASWWAY Implementation Issues		Claude M. Watson	221
Daniel J. Drislane	79	The Management of Computer Resources with a Well Designed Accounting Structure	
Using MicroPower/Pascal to Implement a Data Acquisition and Analysis System		Philip A. Dawdy.....	229
Paul Brown.....	87	Using VMS to Teach Operating Systems	
An Ultra-High Speed Data Acquisition Front-End for PDP-11s		Jerry Scott	235
Edward R. Darken.....	91	Student Information Systems	
Data Acquisition with the MicroVAX II		Warren Alkire	239
W. M. Foreman, J. F. Amann, T. Kozlowski, M. A. Oothoudt.....	95	GRAPHICS APPLICATIONS SIG	
Conversion from RT-11 to Micro/RXS for Real-Time Data Acquisition and Analysis		Computer Assisted Course Development and Instructional System (CACDIS)	
Mitchell E. King.....	97	A. L. Narasimhan	245
Data Acquisition and Valve Control for Onboard Oxygen Generating System Chemical Contamination Studies		IAS SIG	
Paul A. Lozano.....	103	Intertask Communication	
Real-Time Performance of MicroVAX II and MicroVMS		Ted Smith	253
Richard K. Somes	107	LARGE SYSTEMS SIG	
Interfacing to Data Acquisition Systems		TOPS-20 Technical Update	
George Sirois	121	Donald A. Kassebaum.....	265
DATA MANAGEMENT SIG		TOPS-10 Technical Update	
Encryption for Beginners		Frank J. Francois.....	269
Bart Z. Lederman	129	TOPS-10 V7.03 Users Panel	
Designing Front-End and Interface Systems for the Casual End User		Frank J. Francois.....	275
Bud Pine.....	145	LCG Software Products Update	
		Carla J. Rissmeyer.....	277

ARTICLE	PAGE
LANGUAGES AND TOOLS SIG	
Typesetting Articles for the DECUS Proceedings with LATEX Barbara N. Beeton.....	281
An Introduction to TEX and LATEX Samuel B. Whidden.....	289
LATEX Examples Samuel B. Whidden, J. R. Westmoreland.....	307
C Program Portability Michael D. Tilson.....	333
NETWORKS SIG	
A High Speed Local Area Computer Network Across the Goodard Space Flight Center James P. Gary, et. al.....	339
OFFICE AUTOMATION SIG	
ALL-IN-1: A New Road to Effective Applications Barclay Brown.....	371
User Communications for Office Automation Systems Peter LaQuerre	379
PERSONAL COMPUTER SIG	
Putting the Reader Back in Manuals: Computer Manuals and the Problems of Readability - V2.0 Thomas L. Warren.....	387
RT-11 SIG	
TSXLIB: Updated for TSX-PLUS V6.0 Nick A. Bourgeois	399
RSX-11 SIG	
“MACHIAVILLI”: An Engineering Applications Development Environment in DECUS C Under RSX-11M-PLUS R. A. Wittenoom.....	405
Developing Large Programs on the PDP: The Spawn Process Walter Hayes	409
Development of a Computer Based Data Acquisition System S. K. R. Iyenger, R. P. Schmidt	413

ARTICLE	PAGE
VAX SYSTEMS SIG	
HSC50 Operations in a VAXcluster Larry Harzlich.....	425
VAXclusters -- Expectations and Experiences Gary Grebus.....	431
High-End VAX I/O Benchmark Don Hamparian, Michael Huffenberger.....	435
A Robust VMS LOGOUT Driver Activator Larry L. Johnson.....	443
The Time Warp Simulator J. Steven Hughes.....	495
POSTER PAPERS	
Distributed Batch Queues on MicroVAX IIs Michael A. Oothoudt, J. F. Amann, M. V. Hoehn	505
1986NORTHEAST DECUS REGIONAL CONFERENCE	
Adding Devices to RSX without a Sysgen Dennis P. Costello	511
Remote Bridge Management John Heffernam, Donna Ritter	527
Productivity Increases with the CORTEX Application Factory: Empirical Survey Results Anthony C. Picardi	545
Rainbow Color/Graphics Option Use in an Assembler Language Programming Course Robert Workman.....	561
Introduction to Speakeasy David H. Saxe	567
Extraction of 1022 Data to PC Files: New INIT and PRINT Features in V117B John Duesenberry	577
Using Mobius to Extend 1022 and 1032 Capabilities to Personal Computers E. William Merriam.....	595



Excerpts From

DECUS PROCEEDINGS

1962

PAPERS AND PRESENTATIONS
of
The Digital Equipment Computer Users Society
Maynard, Massachusetts

FOREWORD

These Proceedings comprise a broad spectrum of papers whose color, in a figurative sense, ranges from the deep blues of special utility programs and debugging aids, through the lush greens of problem oriented techniques, to the rosey hues of new hardware aids designed to enhance the on-line use of computers. In organizing the papers we have attempted to portray the typical cycle of events centered about the utilization of a new class of computers.

Much initial energy has to be expended on the creation and improvement of utility programs and systems before anything very useful can be accomplished with our systems. To those of us who are strictly problem oriented, this is an extremely frustrating time, made bearable by the naive hope that it might be brief and end with some powerful general problem solving language in our possession. Unfortunately, this dream is inevitably dispelled as we proceed to call for a diversity of modes of control, and of action, which strain the existing hardware and programming systems to their technological limits, in our quest for useful results.

From the insight thus gained, however, is created the structure of new programming systems, and of processor configurations better fitted to provide each particular user with assistance in solving the problems of interest to him. The onset of the second stage of activity is already clearly discernible from the orientation of a majority of the papers in these Proceedings. The theme is closer man-machine interaction. This theme is present, both in the increased emphasis on on-line programming, debugging and problem solving aids utilizing scope and light-pencil communication, and in the requisite improvements in flicker-free scopes, time sharing hardware, and optical I-O devices.

It has been a singular pleasure, during the past two years, to have been in a position to observe the evolution of a Society which spans such a diversity of on-line processor configurations and uses. In this brief interval of time the small scale processor has evolved from a meager and inadequate substitute for a large central computer, into a formidable device whose flexibility and increasingly lower cost makes it the logical candidate for a multitude of real-time information processing operations.

The ease with which hardware can be tailored to particular applications has already out-stripped the software development problem. However, as the engineering technology rapidly improves, and the ultimate user becomes more intimately tied to the operating system, we may look forward to an era in which better control can be achieved and maintained over the growing software domain.

C. M. Walter,
DECUS President

PREFACE

This is the first Proceedings of meetings of the Digital Equipment Computer Users Society. Formed in March 1961, for the purpose of fostering the interchange of information, ideas, and the advancement of the art of programmed data processing - particularly with application to the Digital PDP-1, the Society (DECUS) has grown in numbers and in scope. DECUS now maintains a programming library facility for its members and issues DECUSCOPE, a technical newsbulletin, every month.

The papers presented at two Meetings which took place in 1962 are the subject of these Proceedings. A one-day Symposium was held May 17, 1962 at ITEK Corporation in Lexington on the subject: "Image Processing and Displays." A two-day Annual Meeting, in October 1962, was hosted by the Computation and Mathematical Sciences Laboratory, AFCRL, Hanscom Field, Bedford. The papers presented covered a wide range of subjects and the meeting was highlighted by a lively Panel Discussion called: MACRO, DECAL, and the PDP-1. Some of the papers given then are still in the germinal state but the authors were prevailed upon to contribute them. During 1962, users of a second Programmed Data Processor, the PDP-4, were welcomed to DECUS. More will be reported in the 1963 meetings about this data processor.

The rapid growth of DECUS and its diverse interests are evidenced by the presentations themselves. What may not be clearly visible is the remarkable spirit of cooperation in the interchange of such diverse information. The 1962 Proceedings are a testimonial of this cooperative spirit and a tribute to the authors. I regret that there was not space for the sparkling good humor and even wit, which enlivened the discussion between papers and during the questioning periods. Every user member was represented and participated fully.

DECUS is deeply grateful to all who have contributed to the substance and embellishment of this first endeavor.

Elsa Newman
DECUS Secretary



ANNUAL MEETING

Place: Air Force Cambridge Research Laboratories
L. G. Hanscom Field
Bedford, Massachusetts

Date: October 10, 11, 1962

PROGRAM

October 10 - Wednesday

- 0900 Registration
- 0930 Introductory Remarks - Charlton M. Walter, President of DECUS
- 0945 The PDP-4 Programming System - H. Morse, DEC
- 1030 Reading Film with a Computer - M. Cappelletti, Information International, Inc.
- 1100 A World Oceanographic Data Display System - Edward Fredkin, Information International, Inc.
- 1215 Lunch - Officers' Club
- 1330 Minimax Detection Station Placement - Richard D. Smallwood, AFCRL
- 1400 Displays -
- Group I: to the DX-1 Experimental Dynamic Processor Room
- Display of Minimax Detection Station Placement
- Dynamic Attribute Extraction Display & Discussion - Charlton M. Walter, AFCRL
- Display - Steven Bernstein, AFCRL
- Group II: to the Operations Applications Laboratory
- Displays & Discussion
- 1600 Reconvene in Main Conference Room, Building 1105A - General Discussion and Security Check

October 11 - Thursday

- 0900 Matrix Package for the DX-1 System - Carmine Caso, Wolf R & D
- 0920 Lawrence Radiation Laboratory's PDP-1
1. A Peripheral Processor for Large Computers - Mrs. Dorothy Monk
 2. A PDP Systems Tape - Fraser Bonnell
 3. Translation Problems of a Peripheral Computer in a Multilingual House - R. P. Abbott and L. E. Mish

- 1100 Playing Music in Real Time - Peter R. Samson, MIT
- 1115 Business, Introduction of Newly Elected Officers
 1962-63 Officers
 Edward Fredkin, President
 Elsa Newman, Secretary
 Committee Chairman
 Eunice Cronin, Meetings
 William Fletcher, Equipment
 John R. Hayes, Programming
 Elsa Newman, Publications
- 1230 Lunch
- 1330 The BBN Symbolic Version of DECAL - R. J. McQuillin, Bolt, Beranek &
 Newman, Inc.
- 1530 DECAL, MACRO and the PDP-1 (Panel Discussion)
 Moderator John Hayes, OAL, Air Force Systems Command
 Panel
 Professor Jack Dennis, Massachusetts Institute of
 (for MACRO) Technology
 Harrison Morse, Digital Equipment Corporation
 Alan Kotok, Massachusetts Institute of Technology
 Edward Fredkin, Information International, Inc.
 (for DECAL) Theodore Strollo, AFCRL, BBN
- 1730 Concluding Remarks
 Edward Fredkin, Decus President, (1962-1963)

ATTENDANCE

ANNUAL MEETING

October 10 and 11, 1962

Air Force Cambridge Research Laboratories

CHARLES W. ADAMS ASSOCIATES

Bedford, Massachusetts
John Gilmore - D
Mary Lanahan
Al Rousseau
Paul Rodenhiser

ATOMIC ENERGY OF CANADA, LIMITED

Chalk River, Canada
J. Quarrington - D

BIO-DYNAMICS

Cambridge, Massachusetts
Avery Johnson

AIR FORCE CAMBRIDGE RESEARCH LABS.

Bedford, Massachusetts
Frank Balzer, Jr.
B. Bernstein - pd
Harry Blum
Roger E. Bove
Eunice C. Cronin
Robert Duncan
Donald Easterday
Stuart Gygi
Edward LeFebvre
Philip Lieberman
John Mott-Smith
Vera Pless
Eugene Prange
Richard D. Smallwood P, pd
Charlton M. Walter - P, D
Weiant Wathen-Dunn - D

BOLT, BERANEK & NEWMAN, INC.

Cambridge, Massachusetts
Los Angeles, California
M. Breen
Lucy Darley
Thomas Evans
William Mann
Thomas Marill - D
Richard J. McQuillin - P
David Park
Theodore Strollo - pd

JET PROPULSION LABORATORY

(California Institute of Technology)
Pasadena, California
William Sholey

AIR FORCE SYSTEMS COMMAND

(Electronic System Division)

Bedford, Massachusetts
Charles R. Brown - pd, D
Donald W. Connolly - pd
James Duva
Ira Goldstein
John B. Goodenough
John R. Hayes - P, D
Sylvia Mayer
Raymond Nickerson
Anne Story
Paul Wein
Robert Westfield
Major John T. Willis

DATA PROCESSING, INC.

Waltham, Massachusetts
Richard Mills - D

DIGITAL EQUIPMENT CORPORATION

Maynard, Massachusetts
Harlan Anderson
Robert Beckman
Gordon Bell
Peter Bonner
Martin Graetz
Benjamin Gurley
John Koudela
Nancy Lambert
Harrison Morse - D
Elsa Newman
George Rice

GEOTECHNICAL INFORMATION

Garland, Texas
Gerald Clawson - D

INFORMATION INTERNATIONAL, INC.

Maynard, Massachusetts
Michael Cappelletti - P
Edward Fredkin - P, D
John Wood

INFORONICS, INC.

Maynard, Massachusetts
Lawrence Buckland - D
William Nugent

INFORMATION SYSTEMS DIVISION

(International Telephone & Telegraph)
Paramus, New Jersey
H. Gould - D

ITEK CORPORATION

Lexington, Massachusetts
William Blotnick
Charles Burgess
Terrence R. Cullen
Doris Gagnon
Richard Hagan
H. P. Peterson
Earle Pughe
Edward Radkowski
Robert Rizzo
Edward Spignise
T. R. Stansfield

LAWRENCE RADIATION LABORATORY

Livermore, California
Frazer Bonnell - D
Lloyd Mish - P
Dorothy T. Monk - P, D

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Cambridge, Massachusetts
Professor Jack B. Dennis - P, D
Alan Kotok
Peter Samson - P
Robert Saunders
Jackson Wright

MASSEY DICKENSEN COMPANY

Waltham, Massachusetts
W. J. Lennon

OREGON PRIMATE RESEARCH CENTER

Beaverton, Oregon
Robert W. Coffin - D

RAYTHEON COMPANY

Wayland, Massachusetts
Ralph W. Zaorski

SYSTEMS RESEARCH LABORATORIES

Dayton, Ohio
W. Fahle - D

UNITED AIRCRAFT RESEARCH LABORATORIES

East Hartford, Connecticut
Gerard A. Paquette - D
David Sirota

WOLF RESEARCH & DEVELOPMENT CORP.

West Concord, Massachusetts
D. B. Brzezinski - D
Carmine Caso - P
Norman Hirst
Janet Seltzer

Notes: D - DECUS Delegate or designated representative .

P - Speaker or Panel member .

pd - Displayed CRT showing special programs .

TRANSLATION PROBLEMS OF A PERIPHERAL COMPUTER IN A MULTI-LINGUAL HOUSE

R. P. Abbott and L. E. Mish

Introduction

Datamation - the oracle of our burgeoning industry - has mentioned, on more than one occasion, that Lawrence Radiation Laboratory is a hodge-podge of incompatible machines. This paper might be considered, somewhat, as an "Insider's Confession." Our present computer configuration consists of three 7090's, two 1401's, one LARC, one STRETCH, one 650, and, of course, one PDP. We are in the process of converting the three 7090's to two 7094's. In the foreseeable future, the computer complex will be expanded to include a CDC 3600, as well as a CDC 6600. These machines speak such different languages as Decimal, Binary, Concise, BCD-eeze, XS3-eeze, and, of course, that old standby, Hollerith. Frequently, the output from one of these machines is needed as input to a code which is on some other machine. Usually, the two machines do not speak the same language or the data must be rearranged or both.

In addition to the computer complex, there are many data gathering devices located at various testing stations, both at LRL and at other agencies. These devices generally speak one of the aforementioned languages, but the dialects include 35 mm photographic negatives, 5 or 8 channel paper tape, punched cards, and 7 or 8 channel magnetic tape. Most of these may be odd parity, even parity, or both. Thus, we are called upon to make a translation, conversion rearrangement, or both. The PDP, to which can be attached enough IO gear to be able to accept and generate all of the languages and dialects, was programmed to make these translations, conversion, etc.

Programming for the PDP

Two ground rules were established prior to initiating the programming.

- 1) Because there are at least two IO devices involved in each translation code, the processing speed of each code should be equal to the maximum rate of the slowest of the involved devices.
- 2) Whenever possible, all Input and Output data

must be checked for validity and parity.

IBM cards - intermixed Hollerith and binary - to IBM magnetic tape

The translation from intermixed Hollerith and binary has been done on our 1401's, but to take advantage of the faster card reader and for backup reasons, it was decided to make this available on the PDP also.

Ground rule 1) calls for an examination of rates, so-oo. The card reader can run at 2100 cards/min. The maximum rate for putting card images on 15KC tape is about 3750 card images/min. The processing rate, by rule 1), shall be 2100 cards/min. Rule 2) says that we shall validity check the Hollerith cards. The binary cards will be checked by the large computers at read-in time. Each record on magnetic tape shall be checked and "standard tape techniques" shall be used. "Standard tape technique" means that if a bad record is encountered, that record shall be backspaced, a space equal to one record gap shall be erased, and the record shall be rewritten and rechecked. If ten erasure attempts fail to yield a good record, the problem shall be restarted and the tape replaced.

The conversion from Hollerith to BCD is not wired into the PDP so it must be programmed. The first approach took 103 decimal cells and, on the average, took 47.6 ms to empty the card reader buffer and convert. This was a weighted table search where a maximum of 16 searches were made for each character. The only trouble with this method was that the 47.6 ms is much too large for an individual card cycle of 28.4 ms when running at 2100 cards/min.

The method finally used is a direct table-look up using the Hollerith punches as the table address. For each character, the punches in rows 1-9 provide the table address, and the punches in rows 12, 11 and 0 provide a correction factor. This approach empties the buffer and converts in only 10 ms, which is well under the required 28.4 ms. Space-wise, the fast conversion is a dog. It takes 463 decimal cells, of which only 87 contain data or instructions

and the rest contain zeroes. The zeroes are important to the conversion, however, in that an invalid Hollerith punch is converted to one of these zeroes and when it is written on the even parity tape, a character skip will result and the standard tape technique will stop the problem.

An interesting sidelight is to be found in the time versus space analysis in the previous example. Speed was increased by a factor of 4.7. On the other hand, space was increased by 4.6. This rule was found to be true in other examples: That is, an increase in speed by a factor of N causes an increase in space by the same factor N.

Now that the code has been debugged and timed, it is running at 1600 to 1900 cards/min, instead of 2100. This discrepancy seems to be due to the summation of plus and minus fudge factors on the time quotations for the card reader and tape drive. For instance, the card reader time is not a hard fast 28.4 ms, but more like 28.4 ± 2 ms, under ideal conditions.

Tape start time, and tape stop time, are quoted at ± 1 ms. The fact that the PDP is a 5 microsecond machine doesn't begin to dent mechanical delays on the order of 5 ms. If a highly integrated timing study is made, it might be possible to strike 2100 cards/min, but is it really worth the effort?

IBM magnetic tape to LARC magnetic tape

Another translation problem of interest is the translation of IBM magnetic tape to LARC magnetic tape, or vice versa. Our PDP has an IBM compatible tape control and a LARC compatible tape control. Each control has 2 tape drives. To make this a general purpose routine, it was decided that the code should stand ready to accept variable length records. Rule 2) (validity & parity checks) will be satisfied by standard tape techniques. A look at rule 1) (timing) provides the brilliant idea of simultaneity. That is -- let's start one tape reading and, at the same time, start the other tape writing and perform the translation while they are in motion! It is sort of a "he takes the high road and she takes the low road and I'll be in Scotland before them." Only, it doesn't work. The command structure of the PDP (specifically lack of index register) is such that the bookkeeping necessary to convert three separate characters from a PDP word requires more than 200 microseconds per word converted.

PDP Saves Processing Steps

The procedure which was finally used is to start the read tape, convert the first N words: start the write tape, and finish the conversion of the remaining words. The processing speed attained with this method is about 6 ms longer than the read time. This particular translation code is a good illustration of the processing steps which can be saved with the PDP. This code takes output from, say, a 7090 and, in one step, produces LARC input. The old method consisted of an additional processing pass on the 7090 to prepare a Hollerith image tape, which was punched off line on a 1401. The cards were then converted to a LARC tape by a Remington Rand Card-to-tape Converter.

Other routines

Other conversion routines on the PDP include: Paper tape to magnetic tape, with options to check odd or even parity on the paper tape, write odd or even parity IBM or Remington Rand tapes.

Print LARC or IBM tapes on the PDP Printer.

Magnetic tape to visual CRT, and precision CRT with 35 mm camera. This routine handles both characters and graphical data.

35 mm negatives with graphical data to magnetic tape.

IBM tape to IBM tape.

LARC tape to LARC tape.

Magnetic tape to printer and magnetic tape to precision CRT - simultaneously.

Our assembly routine deserves mention in that it reads the instructions from Hollerith cards, uses IBM tape as temporary storage, prints the listing on the Anelex printer at 1000 lines/min (the printer uses the XS3 language), and the object code is punched in binary form on IBM cards.

Conclusion

The coding techniques which were finally used in our conversion routines are not complex, but they do serve to illustrate that one cannot afford to approach the trivial problem of data conversions with careless contempt.

SPACEWAR! REAL-TIME CAPABILITY OF THE PDP-1

J. M. Graetz

Abstract

The game starts with each player in control of a spaceship (shown on PDP scope) equipped with propulsion rockets, rotation gyros, and space torpedos. The use of switches to control apparent motion of displayed objects amply demonstrates the real-time capabilities of the PDP-1.

Introduction

The demonstration program known as SPACEWAR! was first conceived in December, 1961 at an informal gathering of the Hingham Institute where Wayne Wiitanen, Stephen Russell, and the author were discussing some of the possibilities of the use of the large-screen CRT which was to be attached to the new PDP-1 computer at M.I.T. One idea that caught our fancy was the thought of a moving display under the control of the user. We thought that a simulation of ships in space would provide an excellent demonstration and the discussion developed into the Hingham Institute Study Group on Space Warfare, under whose auspices almost all of the work described here was done. The main control and computation programs were written and debugged in the first months of 1962 by Stephen R. Russell of Harvard.

The program is set up in the form of a game for two persons and a PDP-1. Each person has control over one of two displayed spaceship outlines. The object of the game is to destroy the opponent's ship by blasting him out of space with torpedos. Control is maintained over the ship's orientation by simulating rotational gyroscopes. All translation is achieved with the ship's main drive rocket; the ship will accelerate in the direction its nose is pointing as long as the rocket engines are turned on. Both ships are armed with ballistic missiles (torpedos) which are released from the nose of the ship with a velocity equal to the ship's velocity plus that imparted to the missile by the launcher. From then on, the torpedos are in true ballistic flight. Each ship has one other means of getting from one place to another, namely "hyperspace," which allows him to get out of the way quickly.

The display includes a background of stars and a bright, flickering star or "heavy star" in the center of the scope which maintains a rather fierce gravitational field.

The Game

At the beginning of the game two spaceships, equipped with 31 torpedos, are displayed in diagonally opposite quadrants of the scope face. The players operate switches for the purpose of maneuvering into position for joining the fray. (It is unwise to remain in a single position for a very long time, and also fruitless, for the torpedos have only a limited range.) The torpedos have two types of fuze: one is a proximity fuze which causes the torpedo to explode when it comes within a certain critical distance of any other collidable object which will also be caused to explode. The other is a time fuze which causes the torpedo itself to explode if it has not encountered another object after a given length of time.

The "heavy star" in the center is constantly exerting a strong gravitational influence on the two spaceships (torpedos are not affected by gravity). This star also has a very short capture radius; a ship with reasonably large intrinsic velocity can come in quite close to the star without fear of being captured. This maneuver is frequently used to change direction rapidly.

If a ship is captured by this star, it loses all velocity and is thrust into the "anti-point," that point on the surface of a topologically toroidal scope which is represented by the four corners of the face.

All collidable objects explode on coming into critical range. The current rules require that a game is won only if the remaining ship (after the opponent has exploded) can successfully avoid being blown up by any torpedos which may be left over. A tie is declared: when both ships collide (and explode); when an apparent victor is destroyed by a loose torpedo; or when both ships run out of torpedos. (Each ship has 31 torpedos at the start of each game).

The Spaceships

The two ships have different outlines making them more easily distinguishable on the scope face. Rotation is readily apparent and rocket blast is equally detectable. When the ship is blasting, a fiery tail is seen at the base of the ship, where the main rocket exhaust is placed. The spaceship outlines are generated and displayed by a program written by Daniel Edwards of M.I.T. This program provides a very fast and reasonably flicker-free display. Torpedos appear as single moving dots. They resemble stars rather closely.

The Heavy Star

A bright, flickering point in the center of the scope represents the massive star referred to as the "heavy star." This star has a strong effect (which approximates gravitation) on the two spaceships. The program for this was also written by Daniel Edwards. In the final version of SPACEWAR! he is going to provide an improved integration to eliminate some of the more unexpected, albeit interesting, properties of the "heavy star."

The Stars of the Heavens

To add verisimilitude to the display, a background of stars is provided. At first, this was merely a random display of dots. However, Peter Samson of M.I.T. has written a program which displays a star map of the sky as seen from the Earth's equator. The size of the scope limits the extent of the map to a 45° segment of the heavens. Stars down to just above fifth magnitude are displayed. The display moves imperceptibly across the face of the scope from left to right, and, given time, the com-

plete band of stars of this section of the map will be displayed.

Hyperspace

This is an emergency device. It frequently happens that a ship cannot accelerate fast enough to get out of the way of an approaching torpedo. The player may send the ship into hyperspace then. The ship then will disappear and very shortly will reappear somewhere else on the scope. Since this is a way of getting from one place to another without traveling the distance between, the method used must be hyperspace! Each player has exactly three hyperspace jumps.

On most PDP-1s, the ships are controlled by switches in the Test Word. For the M.I.T. machine, however, two control consoles were devised by Robert A. Saunders and Alan Kotok, both of M.I.T. Each console has a double-throw switch to control rotation, a firing button, and a blast lever. Hyperspace is entered by pushing the blast lever forward and releasing.

Acknowledgement

Special thanks from the Hingham Institute are extended to: the various members of the Tech Model Railroad Club for help and encouragement; to Prof. Jack B. Dennis, director of the M.I.T. TX-0 and PDP-1 installations, whose assistance went beyond the generous allowance of time on the computer; and, to Digital Equipment Corporation without whose gift SPACEWAR! would still be wishful thinking at the Hingham Institute.



Figure 1 A Common Opening Maneuver



A TIME-SHARING SYSTEM FOR THE PDP-1 COMPUTER *

John E. Yates

Abstract

A system for time sharing of the PDP-1 digital computer with seven typewriters, two paper tape punches, two paper tape readers and two CRT Displays is described. The additional hardware required for the system and the modification required to a basic PDP-1 are described and a program is presented to handle the monitor of "executive" functions of the system. A System using two typewriters, one punch, one reader and one display based on this design is currently being installed at M.I.T.

Introduction

A time sharing system for the PDP-1 at M.I.T. has been designed and is in the process of construction. It allows for the use of seven typewriters, two paper tape punches, two readers, and two CRT displays simultaneously, by up to seven users. Every effort has been made to make as many features of the basic machine available to users as possible, although some sacrifices must be made to make the computing capacity available to several users simultaneously.

The System

The seven consoles which comprise the system each consist of a typewriter, six sense switches, a console ON switch, a display lever which allows lengthened quantas, a debugging button, and two lights indicating the console is active in core and it is permissible to type in. The two punches, two readers and two displays are shared among the users on an assigned basis. The test word switches are also assigned.

Programming for the System in Two Parts

The programming for the time-sharing system consists of two parts, the executive routine and the administrative routine. The executive routine is a permanent part of core memory (approximately 512 registers) which will handle the needs of the time sharing system on a second-to-second basis. It will handle the so-called instruction traps and time-out interrupts. 2. The administrative routine is a separate program brought into memory on request to

perform such jobs as: assignment of equipment, regulation of memory protection, provision for services such as an assembling, debugging routines, editing programs, error indication for illegal instructions, and other miscellaneous jobs. Let us assume several users are using the computer, a particular program is in core and is being executed. Since one does not wish the computer to stop because of a user's errors, (and thus keep others from executing) certain provisions must be made. All halt instructions, illegal operation codes, requests for manual run, and illegal instruction cause a trap to the executive routine, ER (See Figure 1).

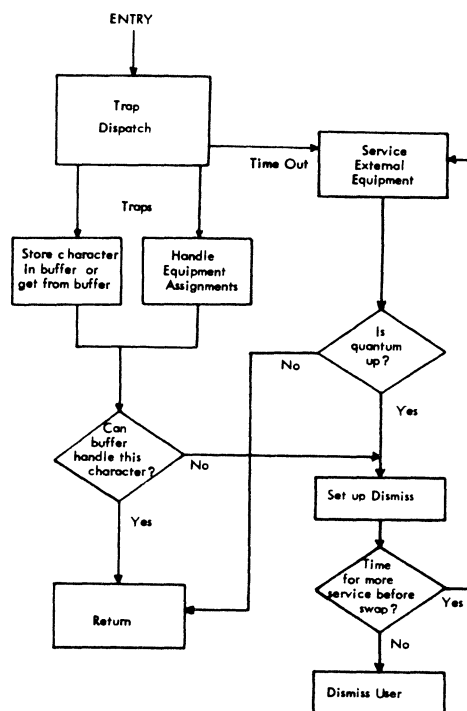


Figure 1 General Flow Diagram of Executive Routine

Similarly certain IOT commands must trap as the program does not know if the equipment has been assigned to it, or which one to address if one has been assigned. The ER then executes the command using the correct assignment, or puts out an error indication thru the administrative routine

Maximum Efficiency

The program may well compute or require characters faster than the I/O equipment can take care of or supply them. Normally, the computer waits in an in-out halt for the completion pulse before processing the next character. Under the time-sharing system it goes to another program while waiting. For maximum efficiency, several characters are computed at once and stored in a buffer in the ER. Then the next program is brought in. At frequent intervals a time-out interrupt occurs where in control is momentarily transferred to the ER. Here one character is taken from each buffer and transmitted, if the I/O device is ready to accept. If not, it is skipped. Control then returns to the program in core. When a certain maximum time has elapsed, or if the ER buffer becomes full, or if the program runs into an error, the program is dismissed and another brought in. A magnetic drum capable of holding twenty - two memories is used as auxiliary storage for the programs not currently active in core. In this way

no time is wasted and each user's program is in memory often enough for the user to think he has the computer to himself.

Several instructions have been added to the machine which are valid during the time the executive routine has control. They are decoded from the IOT 77 class and are used by the executive routine to test the states of the consoles, to make equipment assignments, and to provide the proper information to the status bits for the user current in memory.

*This paper is based on a thesis prepared by the author in partial fulfillment of the requirements for the degree of MS in Electrical Engineering, M.I.T. The complete thesis is available as report ESL-R-140 from:

Publications Department
Electronic Systems Laboratory
Building 32
Massachusetts Institute of Technology

MACRO, DECAL, and the PDP-1

Moderator: Dr. John Hayes*

Panel: (MACRO) Professor Jack Dennis, M.I.T.
Harrison Morse, DEC
Alan Kotok, M.I.T.

(DECAL) Edward Fredkin, Information International, Inc.
Ted Stollo, BBN
Roland Silver, Mitre Corp. (not present)

Dr. Hayes: For some time now, there have been rumblings among programmers about DECAL versus MACRO for the PDP-1. It began to look as though perhaps people wouldn't talk to each other who talked different programming languages. From the discussion today, we should learn a good deal about both of these programming languages.

Among a number of things, I'd like to remind our panelists that some of us know DECAL, some of us know MACRO, some of us don't know either. But very few of us know both MACRO and DECAL. So, I hope that statements may be explained where there may be a language difficulty. In the process of discussion, I would hope that we would develop first of all, for the purpose of new users who are perhaps trying to decide which language to use, the relative advantages of one over the other or perhaps the relative advantages of using both. For old users of MACRO or DECAL, such as the members of my laboratory, we would like to find out enough advantages of one language over the other to justify the time and expense (which may well be considerable) of re-training people to use the other language.

Now I'd like to introduce the panel: For MACRO, Professor Jack Dennis of M.I.T., Harrison Morse of DEC, and Alan Kotok of DEC and M.I.T. For DECAL, Ted Stollo of AFCRL and BBN, and Edward Fredkin of Information International, Inc. I'm sorry that Roland Silver, who was to speak for DECAL could not attend. I am moderator, but I plan to moderate only in the case of severe physical danger to one or more participants. Professor Dennis.

Prof. Dennis: Am I correct in the understanding that so far in this meeting, there has been no presentation of MACRO? How much time do I have for my initial presentation?

Dr. Hayes: The time will be five minutes.

Prof. Dennis: I will start with a brief description of what MACRO is. MACRO is an assembly program (as opposed to a compiler program) and was originally developed in 1958 and 1959 for the TX-O computer at M.I.T. The needs of the users of the TX-O computer, at that time, were the determinants of the features that were placed in the MACRO assembly program. The original version of the MACRO

*Psychologist, Operational Applications Laboratory, Air Force Electronics Systems Division, AFSC, Bedford, Mass.

assembly program was based quite a bit on previous experience in the Whirlwind Laboratory at M.I.T. and the experience of the people who participated in the Whirlwind group. When the TX-O computer was brought to M.I.T. in 1958, we had need for creating a new programming system for the machine. At that time, we asked for an assembly language so that the machine could be used by students and research people at M.I.T. We discovered many features which should be in an assembly program for the type of uses which were being made of it - for example, the automatic macro-instruction feature, whereby a user may assign a name to a sequence of instructions or words and later on in his program use that name to specify the sequence to be placed in the object program. Since then additional features have been added to TX-O MACRO, such as automatic reservation of storage for constants, for variable automatic storage, and automatic reservation of table space by using a dimension statement. Various people here helped with this work. Among them, we should mention Bob Saunders in particular (now with Information International), Bob Wagner who is working for the Rand Corporation, and Alan Kotok who is on this panel. When the PDP Computer was donated to M.I.T. by the Digital Equipment Corporation in the fall of 1961, there was quite a bit of concern about the kind of language that should be provided for the PDP for use by students and staff in the M.I.T. work and after some time we decided to translate the MACRO program so it could be used on the PDP. This was a rather easy job because of the great similarity of the two machines, and I believe it was accomplished in something like three weeks of work on the part of about four people, working part time, which was quite an accomplishment. As to the reasons I think that MACRO is a very useful assembly program, I have the feeling that for the PDP Computer, an assembler is more desirable than a compiler. I feel this way because applications made of the PDP-1 are such that using a compiler would lead to object programs which are relatively inefficient and require considerably more space than required by a program hand-coded for translation by an assembly program. When I say this, I don't mean that a compiler can't be constructed which would be suitable for the PDP-1, but I believe that compilers which are based on the kind of compiling techniques which are now in existence would lead to programs which are long and time-consuming in their operation on this machine. So my feeling is that for many classes of problems for which the PDP is used, an assembly language is the more important language to be concerned with. I believe that the MACRO source language has all of the more important useful features of any assembly language in existence and is very flexible in its use. I think I'll wait until later for any further comments.

Dr. Hayes: Maybe we should now turn to a DECAL representative: Mr. Fredkin.

Mr. Fredkin: Well, I guess I've been labeled a DECAL representative. Let me say something about MACRO, though. I like MACRO and I think it's fine, but I don't think it's fine for the PDP-1 because I don't think the PDP cares what you put in the reader as long as you don't make it shudder too much. I think MACRO is fine for a group of people and there are members of that group here. The fact is, that we could be sitting up here arguing whether we should speak English or some other language and I don't think you can argue this on the merits of the language so much as by looking at the language in terms of its practical uses. DECAL and MACRO are two very different languages. DECAL is a very complicated system, MACRO is a simple system. Simplicity is very nice sometimes and the PDP-1 is perhaps a simple computer, but if you describe the two systems by listing their properties, DECAL includes more of the desirable features of MACRO than vice versa by a big margin.

1. DECAL has one important thing and this is really best described as growth potential. The language is increasing in capability with time. The fact that it's changing may be a disadvantage, but still it is including more and more of the ALGOL language features.

2. DECAL has a library feature. It allows groups, large organizations to set up systems with various individuals' programs. It allows you to use library programs and library tapes and allows you to relocate binary - in general, those things that are oriented for systems programming.

Now MACRO, on the other hand, is a beautiful language for one person who wants to sit down, write his program, and make it work. This is characteristic of many users of MACRO; in particular, I would say, characteristic of M.I.T. students who generally don't get together to write large systems, but write their own programs. They want to assemble, get something out, run it, debug it, etc. That is very different from the way many other organizations use computers. So, I really feel that there are a set of users for which MACRO is better. On the other hand, I feel that there is a much larger set for which DECAL is better because most organizations have invested in systems and these systems are large and quantitative.

Another point...Sure enough, MACRO does result in efficient object codes, but normally I don't care. What I care about is the lapse of time between when I start writing and when I have a finished program. Generally I'm going to write the program over five times and maybe the last time I'll do it in machine code. I want the amount of time I spend doing this to be minimum; I don't care about the machine. I usually write programs for hours that only amount to milliseconds and so sometimes it takes 10 milliseconds (instead of one) for having used the DECAL algebraic compiler. On the other hand, use of the DECAL compiler may cut hours off the time of actually writing the program itself. So, I think that there are very specific issues involved in the choice of a programming language, and I think I'll defer getting down to them until we've heard from each of the people.

Mr. Kotok:

I don't think I can say as much as the two gentlemen who preceded me, but I think they did outline the issues pretty well. One thought, which might be germane, is that maybe we shouldn't be arguing whether MACRO versus DECAL. Instead, whether either one of them or FRAP. It seems I've been informed that a large number of users are still using FRAP for one reason or another and maybe if we come out no where else, at least users will know something about one of these two systems that we are discussing here.

I'm certainly not trying to claim that MACRO is as general a system as DECAL, especially, the new DECAL described in the paper that was presented before this discussion.* I think that you obtain through this generality, the facility of the use of DECAL (as was shown in the discussion that preceded this one) the description of instruction generators and action operators, which caused Mr. McQuillin to indicate that even he gets confused occasionally. The macro-instruction feature of MACRO is somewhat akin to the instruction generator feature of DECAL and our system is, we think, somewhat easier to use. I think that most of the complaints against MACRO and why people say DECAL

*DECAL-BBN - Symbolic Version of DECAL - by R. J. McQuillin - p. 19 of these proceedings.

over MACRO as an assembler is that, first of all, MACRO has a limited set of symbols: 1, 2, and 3 characters. I can certainly see that people can get unhappy with this. Maybe I'm on the wrong side of the fence, too, but, like Ed, I can see where there might be objection to this. Also, in MACRO there is an absence of a linking facility between programs. However, the linking facility, it seems, as provided by DECAL, is a mixed blessing. Since it is a one pass system, there is no way to directly get a loadable binary type which can be read in right away. The second pass of the assembly that MACRO does do is often necessary in DECAL if you do wish just a self-loading tape causing you to go through two passes of punching. These are just a few of the points.

Mr. Stollo:

I'd like to say that I don't think it's different types of programmers who should use MACRO or DECAL. It would seem to me it depended on the type of program to be written. Sometimes when writing a short program which one would like to get into the machine as quickly as possible, MACRO has advantages. But, if one were working on a long system and expected to link several short programs together then I think DECAL is the better system to use because programs which other people have already written can easily be incorporated. A library tape, for example, could be used. Thus available programs which have already been debugged can be linked with the recently written program. Certainly this is a lot easier than recompiling all programs over again and going through a new process with each program and then reading it into the computer.

I think there are two philosophies you can have when writing programs. You can either write one very large program (and you would almost have to do it with MACRO where all the symbols are linked together and where you stand a good chance of not getting the program debugged for quite a while) or you can write several short programs debugging each program as you write it. When you are certain that programs A & B are working, then you can write a program C and get it working and then try it in conjunction with A and B. I think the latter is a strong point of DECAL. You can take all of your shorter programs which you know are working now and link them together with your recently written program and be pretty much certain that you're not going to have a major debugging problem.

Mr. Morse:

I would like, first of all, to make one thing very clear. An impression, I think, Ed Fredkin and Ted Stollo have given is that it's difficult for more than one person to work on one program in MACRO and it's difficult to write a program that isn't one big chunk of coding. This is not true. I have many times written programs which consist of a big glob of subroutines (literally 20 or 40 to 100) and a large control program, with the subroutines on separate symbolic tapes. The subroutines are assembled and checked out separately, prior to putting together the whole system. Once the subroutines are checked out, these are punched out on a binary tape and a symbol punch gotten from MACRO. The subroutines sit in a fixed place and remain there while you go to work on a control program. One great advantage of this is that you can use MACRO instructions as a means of calling these subroutines. In particular, one person can write all the subroutines, define how they're used, and another person, not knowing a thing about this big black box, can use MACRO instructions to call the subroutines. This is one way of performing the same function that DECAL does with the relocatable subroutines which are called by system symbols when the main program is loaded. Just this to counteract the impression that MACRO

is for one-man programs only. There are advantages to both systems. If you're doing mainly arithmetic processing, DECAL does have the advantage that you can write a program much more quickly and possibly get it debugged much more quickly. A disadvantage here is that at the present time DECAL's programs must be debugged in octal. This will eventually be counteracted by using DDT and symbols from DECAL for debugging. Another disadvantage is that if the DECAL program is large and has many systems symbols which must be stored in memory while loading the program, then you also have storage problems that are alleviated by using MACRO since you can use all of core except the last 27 registers or so.

Mr. Stollo: If I understand MACRO correctly all symbols are three characters in MACRO. Is that correct? For example if someone else were working on a program could you say to them don't use the symbol A because I'm using the symbol A in my program and you can't use it in yours? Is this what you would have to do? I think there should be a feature for external symbols because there are a certain group of symbols that I use over and over again in several of my programs and even if I were working on a system I like to use these symbols within the program like "move" or something like that.

I think the communications problem when you're writing a long system would be enormous if you had to eliminate all the symbols you use and pass it on and say don't use these symbols in your program.

Mr. Fredkin: Well, when writing with FRAP we used to break things up. We used to say I'll start all my symbols with my initials. This is sort of hard when you're limited to three characters because it doesn't leave too many initials.

Dr. Hayes: Especially if you have a long name.

Mr. Fredkin: That's right. If you have four initials.

Prof. Dennis: Take the example that Mr. Morse gave in which you compile a set of subroutines and then define a set of MACRO instructions to be the calling sequences for the subroutines. After you've got to that stage, you may dispense with the symbols which are involved in the subroutines and simply use the MACRO instructions in your main control program. So once you have coded the subroutines and defined the calling sequences and debugged them you may dispense with all of the symbols involved in these programs in the subroutines and refer to them only through the MACRO instructions.

Mr. Fredkin: Isn't that true only if you know the binary locations?

Prof. Dennis: No.

Mr. Fredkin: Dispensing with all the symbols?

Prof. Dennis: One way of doing this is to define the calling sequences as macro-instructions on a separate tape which is assembled with the subroutines. Then a definitions tape is obtained containing only the macro-definitions. If you have used the system correctly, the macro-instructions defined on the tape provide the correct calling sequences for the subroutines, but the tape will not have any of the symbol definitions of the subroutines.

- Mr. Fredkin: However, these subroutines will not be able to link if you get rid of their definitions, unless you pick the binary location.
- Prof. Dennis: That is correct.
- Mr. Fredkin: Now for instance on a DECAL library tape you might have 20,000 instructions worth of program. You can't fix the binary locations and pick any subset so it's impossible to have such facility in MACRO where you do in DECAL. By the way, when we talk about library tape I thought I'd mention one thing. One of the advantages of MACRO is the ease of tape handling. With DECAL, especially with this library tape and such, we had in mind from the very beginning that this should be a magnetic tape feature eventually. It should work with paper tape, and in addition it should get into magnetic tape. It is on magnetic tape here and there are people who put in a little paper tape, maybe about 20 fanfolds, where they crowd a whole slew of things in the library and just go whizzing through this mag tape and they pick up all of these routines so that you do get access easily in a relocatable form.
- Prof. Dennis: I would like to point out that for the TX-O computer there is a relocating version of MACRO assembly program. The relocating features were not translated into the PDP version because of space limitations in memory of the PDP. However, I expect that this is something that the Digital Equipment Company would be interested in doing, but we don't have the manpower at this time.
- Another way, is to store the symbolic version of each subroutine on tape and add to MACRO a facility which could be done with about as much trouble as putting in the DECAL library tape to call the subroutines wanted in symbolic, assembling these at that time. This means a double tape handling, but when you're handling magnetic tape the extra time needed is still so much less than the time used to handle the paper tape it becomes a very workable scheme and does not entail large changes to the MACRO system itself.
- Mr. Kotok: Just to comment on the thing Prof. Dennis was talking about before - assembling a large number of subroutines and using the MACRO instructions with these subroutines as calling sequences may be done by using the symbol punch facility in MACRO. The symbols may be punched for use with DDT, or the MACRO instructions without the symbols may be punched for use at a later date. The MACRO calling sequences would be absolute addresses of the subroutines for later use.
- Mr. Fredkin: Three things occur to me: First, how about the length of symbols because we can't name everything you want with three letters? Second, what about relocation? Third, what about library tapes? These are all features of DECAL now and they could be a part of MACRO.
- Dr. Hayes: Yes, our discussion seems to have boiled down to the properties of future programs. Are there any further comments from the panel or is it now time to entertain questions from the floor?
- Note: (Questions from the audience were not audible for purposes of recording them. One question to Dit prompted the explanation of macro-instructions.)
- Mr. Morse: I would like to give a brief description of how to use macro-instructions. The

macro-instruction facility is a way of naming a series of instructions which are commonly used in the program, which can be put in the program by writing the name of the macro-instruction.

For instance to define the MACRO instruction load:

```
define          load B, A
                lac (A
                dac B
                terminate
```

This MACRO instruction is commonly used to load register B with the constant A.

Now to use the instruction in the program to load 3 with register zzz one need only write:

```
load zzz, 3
```

I may also use other MACRO instructions within a MACRO definition:

```
define          load 2 z, one, two
                load z, one
                load z+1, two
                terminate
```

The use of this

```
load 2 g, 4, 20
```

will cause the following instruction to be assembled

```
lac (4
dac g
lac (20
dac g+1
```

This operation will be performed many times. The argument A will be cycle lac 9 times and that can be used as part of the later work. For example, this is essentially the MACRO feature.

Mrs. Newman: A good, brief description of MACRO appeared in the May 1962 issue of DECUSCOPE.

Mr. Morse: Thank you.

Prof. Dennis: In programs written in a large interpretive system (for example, a system for floating point computation), the interpreted instructions may be given names with mnemonic significance by parameter assignments or macro-instruction definitions. With macro-instructions, specifying the parameters of an interpreted instruction is far more convenient. Of course, an interpreted instruction may occupy two or three registers, depending on how many arguments must be specified to the interpreter. This makes no difference when you are using macro-instructions. The macro-instruction may have a length of 1, 2, or 3

registers depending on the particular instruction it represents.

- Editor's Note: There was a comment from the floor about the ease of writing macros.
- Mr. Fredkin: There is one thing about macros. They are easy to write, but I would rather work with "instruction generators" which are easy to use. You use things more often than you write them and since you are only going to write it once you don't need MACRO to do it. Let me give you an example of this. What do you do when you want 39 in register? In DECAL you write: 39⇒A but in MACRO you have to remember whether it is: LOAD A, 39 or: LOAD 39, A (which goes into which). I guess Dit made a mistake in the definition and you can write into A, put 39. His results will involve the equivalent law 47, and dac into A. Taking Dit Morse's example in the May DECUSCOPE; I showed him a program in DECAL which did the same thing and it was easily 1/8 as long and he said that's not fair because I used existing subroutines. I didn't use anything but a single DECAL library tape. So the program was shorter and much easier to write.
- Mr. Morse: This is true, but first of all, the example was to illustrate the use of macro-instructions and was not intended to compare MACRO's virtues with those of any other programming system. However, let's use it for that as Ed has, and compare the effort needed to run the programs. Using MACRO, you need only do two passes on the symbolic tape and you have a binary tape which may be loaded and run. Using DECAL, you must first assemble the program, then load the linking loader, load the program, load the library tape, and if you do not wish to do this every time the program is run, you must load punch-off and punch out a binary tape.
- Editor's Note: There was much reaction in the audience, especially from DECAL users.
- Dr. Hayes: I think the audience is getting jittery because they cannot participate. Are there any questions from the audience?
- Questions from the floor: One of the features of DECAL is the instruction generator. I think this is equivalent to definitions. Is this correct?
- Prof. Dennis: Yes, in the form of macro-instruction definitions.
- Mr. Saunders: What you can do, for instance, is to have additional MACRO instructions written into the programs. What one cannot do is have the MACRO instructions written in duplicate on certain substructures depending on the value.
- Mr. Stollo: If you can't get all of the instructions in on DECAL, you can insert a new tape in DECAL. Can you do this in MACRO?
- Mr. Morse: Yes, it is possible.
- Dr. Hayes: Any comments from the floor?
- Question: Not audible but Moderator repeated.
- Dr. Hayes: The question has to do with the use of magnetic tape with DECAL.

Mr. Fredkin: When you use it, my experience with DECAL is that even paper tape tears much less. DECAL definitely has growth potential with respect to magnetic tape.

(A question was directed to Mr. Fredkin about writing programs.)

Mr. Fredkin: The thing is that DECAL has facility for doing things. In MACRO you write the programs over and over, but in DECAL we only do it once. A very important thing is the joining of binary programs. You can do it in MACRO, but in DECAL we put them in locations and never bother with them again. In general, if you have a very complicated mathematical thing and you have to be fast, you can do parts of it in DECAL algebraic language and then maybe convert.

Prof. Dennis: The language I would use would depend on whether my program could be divided into subroutines. Certain programs are impossible to divide into subroutines. Then the question of MACRO versus DECAL depends on whether the macro-instruction feature of MACRO turns out to be useful with reference to what you are doing, and in most cases it is. The advantage of using MACRO for programs with many subroutines is that you can give nice names to their calling sequences and refer to them by convenient names. You have the advantage in DECAL which is given by the linking loader feature. I prefer the coding format of MACRO to the coding format of DECAL. This, of course, is something outside of what either program can do for you and I admit that this is a matter of opinion and my personal bias. It may also have something to do with my experience with MACRO.

Editor's Note: Discussion from the floor became more lively but speakers were heard by those sitting close by. A question was raised about the effect of DECAL on the PDP causing strain on input-output devices and it was pointed out that the M.I.T. machine had been modified for MACRO and didn't accept DECAL. Jackson Wright repeated that the format of MACRO was easier for a program writer. A little excitement was engendered at this point. It was obvious that the audience was having a good time and that the DECAL users thought it more advantages for them in its present form.

Dr. Hayes: Yes, Mr. Kotok.

Mr. Kotok: I saw Ted Stollo working on a program on the flexowriter. I didn't see any algebraic statements in it at all. He mentioned the manipulations which you will have to go through to type the DECAL program, some of which have to do with just which characters to choose. All the upper cases were troubling him. Also, a system where you have to put in information as to where you are assembling and not compiling has many difficulties such as the difficulty of putting in addresses alone.

Prof. Dennis: It depends on whether you are talking about compiling. If you are doing your own programming and have no typist, the more characters you have the more chance for errors.

Mr. Stollo: This could be remedied by the action operators in DECAL.

Mr. Fredkin: Ease of typing should not be the basis for evaluating a system.

Dr. Hayes: It is difficult to evaluate on the basis of how many keys you have to strike to make a comma.

Mr. Fredkin: There was a time when I, too, used to program in MACRO. I liked MACRO instructions but I've made progress. The algebraic statement is the best although I'd like to have a combined system.

Mr. Kotok: We ought to ask the audience what they like, we have been talking mainly about what we have to offer. It would be interesting to find out what they use and what they like. (Many voices and affirmative nods.) Who are DECAL users?

Editor's Note: The moderator asked for a show of hands. The number of people using MACRO and the number of people using DECAL were about the same. The count for each system is given below.

16 DECAL
16 MACRO - (M.I.T. programmers)
9 FRAP

Dr. Hayes: About one-half of the audience did not indicate a preference. That is very interesting. Yes, Ed.

Mr. Fredkin: MACRO is 5 years old and has reached some maturity. It has a good write-up. DECAL hasn't reached the same state of maturity but seems to be getting there. I think that within the not too distant future we will see DECAL with a good up-to-date Symbolic and a good write-up.

Prof. Dennis: DECAL as it is presently offered, does not have the possibility of subscripted variables -- the most important feature of the algebraic language. I understand a version of DECAL is being prepared now which does offer subscripts but I have the feeling that putting subscripts in DECAL is going to increase the inefficiency of object programs over programs created with the absence of subscripts and I think it is possible to create a compiler language for a computer like the PDP-1 which could compile efficient object programs better than any today in that it would not be a one to one translation between source programs and object representations. I believe that such a program is possible and I would like to see one prepared and I would then be sure to use a compiler for any program I would write, but until such time I will use the assembler.

Dr. Hayes: Yes. Would the other members of the panel like to give some conclusions now?

Mr. Morse: I believe MACRO is a better system for writing programs in which you need close control over the resulting object code and storage allocation, for example a real-time control program. In contrast, DECAL is more efficient from the point of view of the lapse time of beginning a program and getting it running.

Mr. Stollo: It's a matter of what type of program one is writing and whether it is desirable to use programs other people have worked out. When linking a group of programs together, one saves time with the DECAL system.

Mr. Wright: Can you link programs with different symbols and different programs?

- Mr. Fredkin: Yes! DECAL does it! BBN has it - In summary; an interesting thing happened some time ago - Elsa Newman got after me. (She's the greatest weapon DECUS has!) With reference to outlining virtues for DECAL or MACRO, her idea was to do something like this debate, but in written form for the DECUSCOPE. So Dit Morse and I got together to have a debate and what happened was that I agreed with nearly every statement he made and I think, vice versa. We got so bored with this, that after three-quarters of an hour, we went home. On the panel today, I decided that I would argue more strongly in behalf of DECAL, but my feeling is that both systems are good, for the reasons I've mentioned earlier.
- Mr. Kotok: I must agree with Ed. I argued for MACRO, but I feel as Ed does that both systems are worthwhile. I would have liked to find out about what others like. If one sees something that neither of these systems has or can find a compromise that you think is better drop a line to DECUSCOPE and we'll start something like the ACM debates.
- Audience: (laughed)
- Prof. Dennis: The discussion this afternoon served a very good purpose in bringing to light the features of these two systems to the audience. If this is so, it has served its purpose.
- Dr. Hayes: I hope, in spite of the good-fellowship and gemütlichkeit we have generated, that the audience will have gained some appreciation of the differences between these two systems and that they will now be able to ask better questions about them for their own applications.

ARTIFICIAL INTELLIGENCE SIG

DEVELOPMENT OF A VAX TUNER USING OPS5

Robert A. Small
Vitro Corporation
New London, Connecticut

ABSTRACT

The VAX Tuner was conceived of as a learning experience in our attempt to gain mastery of knowledge based system development skills and techniques. Using a Lisp machine and a knowledge based system development tool, a prototype was developed quickly to validate the concept of a computer-based Vax Tuner. Several simplifying assumptions were made and OPS5 programming techniques were adapted to the problem.

I. Introduction

This paper describes the development of a prototype VAX Tuner knowledge based (KB) system using OPS5e. The project reflects the efforts of Karen Kennedy, David Kennedy, and myself; it was completed in the summer of 1985. We learned several lessons in the course of developing this prototype -- some were technical, others were not. The remainder of this paper will attempt to share what we have learned.

II. How we got started

Our company's long range objective is to sell Artificial Intelligence (AI) solutions to both our existing Department of Defense (DOD) customers and new customers. To meet this goal, a strategy involving hardware, software, and trained people (fleshware) was formulated. To begin, we purchased several Lisp machines, one of which is located in our office. Additionally, the Lisp-based version of OPS5 (OPS5e) the tool for the construction of KB systems, was purchased for our site.

Our group of three is comprised of a former sonar engineer and two software engineers with sonar software experience. One of us (Small) had several years experience in VAX/VMS system programming.

The first step in our preparation was to attend seminars on AI and expert systems as well as trade shows. We also joined appropriate professional organizations and began to read the literature. Our Lisp machine purchase came with some Lisp training which helped us to use our new computer and introduced us to the language.

We had developed some momentum; our experience at seminars was becoming repetitive. There was nothing left to do but begin.

III. Choosing a Project

Several applications were explored; most were dismissed because they were seen as too difficult both in terms of our limited experience and our need to create a meaningful demonstration system in a short period of time.

We settled on the VAX Tuner as our first project for several reasons. First, my background

in VMS coupled with the VAX/VMS Release 4 documentation, specifically, its enhanced tuning discussion, gave us easy access to sufficient expertise to get started. Secondly, although VAX/VMS tuning is a broad problem, we felt it could be segmented and a meaningful demonstration system could be created in a reasonable amount of time (not more than a few months). Third was the appropriateness of the tool for the job. It is usually important to select the right tool for the given problem. In our case, we had a tool and were in search of a problem so we matched the application to the tool. The procedural knowledge in the DEC documentation seemed to lend itself to the kind representation that OPS5 easily supported.

The lesson to be shared here is that in order to develop a KB system, you must have access to sufficient information about the domain. The classical approach to construction of a KB system is to give a knowledge engineer access to a domain expert and some hardware/software and he/she can synthesize an expert system. While we did not employ a recognized expert in VAX Tuning, we did mine and refine "textbook knowledge" and tempered it with experience. The fruit of our labor is a prototype that successfully demonstrates that a computer program can recognize VAX resource utilization patterns that can be improved with tuning and it offers advice as to how to achieve this improvement.

IV. Narrowing the Scope of Vax Tuning for Concept Demonstration

The tuning manual gives a stern warning to those who would tinker with the system parameters:

"Tuning is a delicate and often time-consuming operation that requires both thorough familiarity with the system workload and a deep understanding of VAX/VMS resource management mechanisms. An attempt to tune a system without the proper level of understanding may very well degrade, rather than improve, system performance.... Too many users assume incorrectly that tuning is a first rather than a last resort solution."
(1)

Our objective was to demonstrate a central concept -- that we could construct a system to

make meaningful tuning recommendations given a state snapshot of the VMS workload environment. To be sure, the variety of hardware configurations available and workload differences make tuning a very broad problem. There are, however, core issues that we needed to abstract for our demonstration system. We, therefore, made several simplifying assumptions.

The process of VAX tuning begins with the identification of a bottleneck in a major computer system resource: memory, CPU, or I/O. We selected the memory subsystem for our prototype. The procedural rules of our system focus on the VAX/VMS resource management mechanisms. The issue of system workload familiarity was temporarily deferred. For our prototype, we "hardwired" acceptable system performance thresholds. In a fully developed product, there would have to be an "installation procedure" where the Tuner could observe the workload and gain a sense of what is "normal" for its host system.

A VAX tuner, as a product, would no doubt process system performance data that it collected on-line. Perhaps it would be raw data from the VMS Monitor Utility or DEC's System Performance Monitor (SPM). Since we did not have on-line access to a VAX we collected Monitor Utility output in files and extracted system performance data reflecting different system conditions. These snapshot files were then loaded into the Lisp machine.

The issue of delivery vehicles was ignored. Since OPS5 written in Bliss runs on the VAX, transportability of the Tuner knowledge base was not seen as an issue worthy of immediate consideration. A fortiori, it was not at all clear that the tuner would ever develop into a product; the issue of delivery might be moot.

V. Overview of OPS5

OPS5e on the Lisp machine is an environment that includes many resources. A rule editor allows production rules to be created and modified. Rules are of the form "IF conditions THEN conclusions;" the conditions are referred to as the left-hand side (LHS) and the conclusions, the right-hand side (RHS). A detailed example of the structure of a rule is given in Figure 1. The state of knowledge is maintained in working memory. These working memory elements (WME) are structures called objects represented by a class name and associated attributes. The attributes have values that may change in the course of reasoning.

The system does pattern matching between WME and condition elements of rules. The resource that performs the pattern matching is the inference engine. When a match is found between a WME and the LHS of a rule, that rule is a candidate to fire and is displayed in the conflict set. The working memory elements that would cause the rule in Figure 1 to enter the conflict set are shown in Figure 2. In firing a rule's RHS is executed. If more than one rule is a candidate to fire, a conflict resolution strategy selects only one rule from the conflict set to be fired. As rules are fired, they are displayed in the history set.

There are also resources available to facilitate debugging of the knowledge base. The user can choose a single step mode in either the forward or backward direction. In forward

stepping, a rule is fired and any changes its RHS induces will be represented in the next state of

```
(defo sample-rule
  ((class-1 ^attribute-1 xyz
            ^attribute-2 zyx) lhs-1)
  - (class-2 ^attribute-1 abc)
  (class-3 ^attribute-1 ghi)
  -->
  (modify lhs-1 ^attribute-2 qrs)
  (remove 3))
```

Figure 1. Detailed example of the structure of a rule. The keyword DEFP defines a production rule. It has the name SAMPLE-RULE and two left-hand side (LHS) elements. The rule will enter the conflict set if working memory contains an instance of CLASS-1 ATTRIBUTE-1 equal to XYZ, CLASS-1 ATTRIBUTE-2 equal to ZYX, and CLASS-2 ATTRIBUTE-1 not equal to ABC and CLASS-3 ATTRIBUTE-1 equal to GHI. The first LHS element has a clause name of LHS-1 that is local to the rule. The second LHS element does not have a clause name. The arrow demarcates the LHS from the right-hand side (RHS). When SAMPLE-RULE fires, the two RHS elements will be executed. The first will change the value of ATTRIBUTE-2 in the LHS element named LHS-1 to QRS. The second will remove the element from working memory that matches the third LHS element.

```
-! A !-
(class-1 ^attribute-1 xyz ^attribute-2 zyx)
(class-2 ^attribute-1 mnop)
(class-3 ^attribute-1 ghi)

-! B !-
(class-1 ^attribute-1 xyz ^attribute-2 qrs)
(class-2 ^attribute-1 mnop)
```

Figure 2. Working memory elements associated with SAMPLE-RULE. 2-A shows working memory before SAMPLE-RULE fires and 2-B shows working memory after it fires. It will fire if it is either the only rule in the conflict set or it is selected based on the criteria of the conflict resolution strategy.

knowledge. In backward stepping, the previous changes to working memory are retracted, if possible, and the state of knowledge returns to its condition prior to the firing of the last rule.

Programming in the pattern matching model of OPS5 is unlike programming in a conventional High Order Language (HOL) like Pascal; there are no branching structures and code is not executed sequentially. If the contents of working memory pattern-match the LHS of a rule, it is a candidate to fire. If the rule triggering pattern reoccurs, the rule will fire again.

The process of pattern matching on the LHS is characteristic of the reasoning strategy called forward chaining where the system considers what is initially known and uses the rules to conclude whatever the knowledge base will support. This process of drawing conclusions (firing of rules) is repeated until nothing further can be concluded (no more rules can fire).

VI. Building the Tuner

The process of building the Tuner was iterative and exploratory. We did not work from design specifications and our requirements were at a very high level. Using the fault logic tree in the Release 4 Tuning Manual, we created rules and data structures as needed to represent a path of the tree from root to twelve terminal nodes. Many of the observations that the tuning process required could not be made directly from the VAX Monitor data so OPS5 programming techniques were used to fulfill these data requirements as needed. An example of this is given below. A data file representing each VAX performance scenario was available to test each logic path that we implemented.

This approach, if continued, would yield a KB system that would approximate the competence of the tuning approach documented by the fault logic tree. Assuming that experts in VAX tuning possess heuristic or rule-of-thumb knowledge that is not contained in the fault logic tree, one or more people who are pre-eminent in the art of VAX tuning would certainly need to be consulted and their knowledge of tuning codified. This step is essential if one were to develop an expert VAX Tuner.

VII. Programming Examples from the Tuner

To motivate the examples, consider the preliminary milestones in the tuning fault logic tree. In the Guide to VAX/VMS Performance Management, Figure 3-1 initiates the tuning investigation; the entry point to the memory analysis is in Figure 3-2 where phase I of the excessive paging investigation is considered. Phase II is reached in Figure 3-3 by two alternate paths in phase I. The first example illustrates how some of the fault tree analysis is captured in OPS5 rules.

Figure 3 contains a reprint of the first phase of the excessive paging investigation (Figure 3-2) from the Guide to VAX/VMS Performance Management, for reference. Figure 4 contains five of the rules used in this phase of the investigation. Notice that the first LHS element in each rule refers to the class DIAGNOSIS and attribute NEXT. This will be abbreviated as D-N. The value of D-N is modified by each rule as it fires. Each value was chosen to help associate the rules in the knowledge base with the decision nodes in the fault logic tree.

This was useful for two reasons. It facilitated programmer understanding during

development and it also insured that a rule would only fire in the desired context. This issue is illustrated in Figure 1. Notice that the same question (Overall Fault Rate High?) is asked whether the answer to the previous question (High rate of Hard Page Faults?) is yes or no.

In Figure 4, R₁ (inv-memory-limitations) enters the conflict set when the value of D-N is "inv-memory-limitations." When it fires, D-N becomes "inv-high-page-fault-rates." This rule really serves as a placeholder. Our notion was to have a complete mapping of rules to decisions nodes.

With this value of D-N, three rules match this WME. Each of the three rules, R₂, R₃, and R₄ (high-page-fault-rate-from-disk, high-cache-fault-rate, and not-high-page-fault-rates) have other LHS elements. Since the second and third LHS elements in R₄ are mutually exclusive with the second LHS element of R₂ and R₃, at most, only two rules (R₂ and R₃) can be in the conflict set. The

Figure 3-2 Investigating Excessive Paging—Phase I

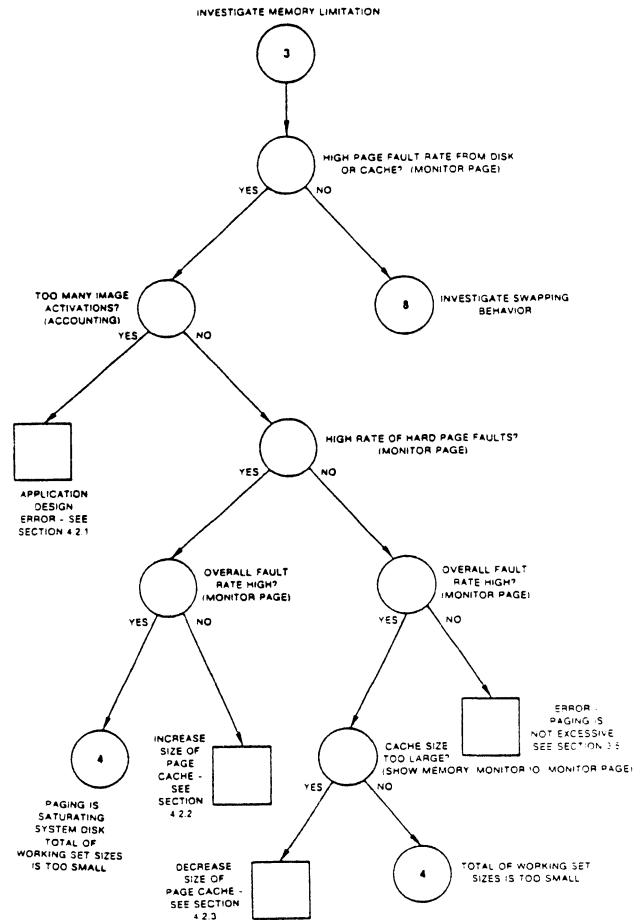


Figure 3. Investigation Excessive Paging - Phase I, from Digital's Guide to VAX/VMS Performance Management.

```

(defp inv-memory-limitation ; R1
  {(diagnosis
    ^next inv-memory-limitation)
   inv-memory-limitation-match}
  -->
  (modify inv-memory-limitation-match
    ^next inv-high-page-fault-rates))

(defp high-page-fault-rate-from-disk ; R2
  {(diagnosis
    ^next inv-high-page-fault-rates)
   high-hard-fault-rate-1-match}
  (spv ^page-read-io-rate > 5.)
  ; temp. threshold for high = 5
  -->
  (modify high-hard-fault-rate-1-match
    ^next inv-image-activations))

(defp high-cache-fault-rate ; R3
  {(diagnosis
    ^next inv-high-page-fault-rates)
   high-soft-fault-match}
  (spv ^page-fault-rate > 20.)
  ; temp. threshold for high = 20
  -->
  (modify high-soft-fault-match
    ^next inv-image-activations))

(defp not-high-page-fault-rates ; R4
  {(diagnosis
    ^next inv-high-page-fault-rates)
   not-high-page-fault-rates-match}
  - (spv ^page-fault-rate > 20.)
  ; temp. threshold for high = 20
  - (spv ^page-read-io-rate > 5.)
  ; temp. threshold for high = 5
  -->
  (modify
    not-high-page-fault-rates-match
    ^next inv-swapping-behavior))

(defp image-activations ; R5
  {(diagnosis
    ^next inv-image-activations)
   image-activation-match}
  (spv ^image-activations > 20.)
  ; temp. threshold for high = 20
  (last-scenario ^scenario <scenario>)
  -->
  (modify 3
    ^scenario "image-activation")
  (modify image-activation-match
    ^next clean-up))

```

Figure 4. First five VAX Tuner rules to perform phase I of the excessive paging investigation.

decision node in the tree at this point suggests disjunction. That is, a rule with LHS elements ORed together. Unfortunately, OPS5 treats all LHS elements as a conjunction; they are ANDed.

If either of the two rules R_2 or R_3 fire, due to a high page fault rate from disk or cache, they modify D-N to "inv-image-activations." This, in turn, matches a LHS element of two more rules, R_5 and R_6 . If the IMAGE-ACTIVATIONS attribute of class SPV (system performance values) exceeds 20 (or whatever threshold is used for this parameter), then the R_5 (image-activations) is fired. This corresponds to the terminal node in the fault logic tree for application design error. At this point in the prototype, the manual's advice is synopsisized and presented to the user.

The term SCENARIO in class LAST-SCENARIO is used to solve an internal tuner problem -- keeping track of which scenario is currently being diagnosed. It has no bearing on the tuning analysis itself. At the conclusion of the analysis, after the results and advice are given, working memory is cleared and a menu is presented for the user to select another tuning scenario for analysis.

To this point in the tuning investigation, questions are asked that can be answered directly by looking at values for system and user processes. The first step in phase II of the excessive paging investigation is to determine which processes are faulting the most (Monitor Processes/Topf). The second example shows how OPS5 rules were used to create a sequence of the top faulting processes based on each process' page fault value. This sequence was not available directly from the VAX performance measurements.

The class definitions of the objects used by the rules in this example, PC (process characteristics), Utility, and Diagnosis, are shown in Figure 5. There are four rules involved and they work in two pairs; TOPF and ORDERING-TOPF are shown in Figure 6; TOPF-POST-PROCESS and TOPF-EXIT are shown in Figure 7.

The tuner uses a rule of thumb or heuristic in determining which processes are the most faulting. The 80% rule, as it is called, finds the highest faulting process and then the next highest faulting process as long as the next

```

(defliterate pc
  pid
  name
  wssize
  wsquota
  page-faults
  swapper-trimmed
  topf-flaq)

(defliterate diagnosis
  next
  initialization
  file)

(defliterate utility
  max-topf-pid
  prev-max-topf
  max-topf
  %80-value
  status)

```

Figure 5. Class and attributes used in determining the top faulting processes.

```

(defp topf
  ((diagnosis
    ^next
    inv-voluntary-decrementing)
   match)

  ((utility ^max-topf <max-topf>
    ^prev-max-topf
    <prev-max-topf>
    ^%80-value <%80-value>
    ^max-topf-pid
    <topf-pid>)
   max-match)

  ((pc ^page-faults > <max-topf>
    ^page-faults >= <%80-value>
    ^page-faults <
    <prev-max-topf>
    ^pid <pid>
    ^page-faults <page-faults>)
   pq-match)

  -->

  (modify max-match
    ^max-topf
    <page-faults>
    ^%80-value
    (compute .8 *
      <page-faults>)
    ^max-topf-pid <pid>))

(defp ordering-topf
  ((diagnosis
    ^next
    inv-voluntary-decrementing)
   topf-diagnosis)

  -->

  (modify topf-diagnosis
    ^next topf-post-process))

```

Figure 6. TOPF and ORDERING-TOPF rules used in determining the top faulting processes.

highest faulting process has a page fault value that is greater than or equal to the page fault value of the previously found process. The search for the highest faulting processes continues until the page fault value for the i th + 1 process is less than 80% of the page fault value for the i th process.

When the NEXT attribute of the class DIAGNOSIS is set to "inv-voluntary-decrementing" the rule ORDERING-TOPF enters the conflict set. The rule TOPF also enters the conflict set if its other two LHS components are satisfied. The LHS element labelled "max-match," involving the attributes of the class UTILITY, does not contain any test; it is included in the rule so that variables local to the rule can be established. The first time the rule is considered, these attributes have been set to initial conditions by a previously fired rule. The attribute MAX-TOPF stores the page fault value associated with the currently considered process. That process is

```

(defp topf-post-process
  ((diagnosis ^next
    topf-post-process)
   post-diagnosis-match)

  ((utility ^max-topf <max-topf>
    ^max-topf-pid <topf-pid>
    ^%80-value <%80-value>)
   post-match)

  ((pc ^page-faults > <%80-value>
    ^topf-flag nil) pc-match)

  -->

  (make topf-processes
    ^topf-pid <topf-pid>
    ^topf-faults <max-topf>
    ^topf-status nil)

  (modify post-match
    ^prev-max-topf <max-topf>
    ^%80-value
    (compute .8 * <max-topf>)
    ^max-topf 0)

  (modify pc-match
    ^topf-flag t)

  (modify post-diagnosis-match
    ^next
    inv-voluntary-decrementing))

(defp topf-exit
  ((diagnosis
    ^next topf-post-process)
   exit-match)

  -->

  (modify exit-match
    ^next inv-excessive-paging))

```

Figure 7. TOPF-POST-PROCESS and TOPF-EXIT rules used in determining the top faulting processes.

identified by its PID in the attribute MAX-TOPF-PID. The 80% value and the page fault value of the previously considered process is stored in PREV-MAX-TOPF. The third LHS element requires that, for a process to be considered, its page fault value must be greater than the MAX-TOPF term, not less than the PREV-MAX-TOPF term defined in the immediately preceding LHS element.

For each WME representing a process that satisfies the conditions of TOPF, TOPF will appear once in the conflict set; if there are five process that satisfy its LHS, five instances of TOPF, each associated with the working memory element that it matches will be shown in the conflict set. The conflict resolution strategy prefers rules with more LHS terms since they are more specific than simpler rules. Therefore, all the TOPF rules in the conflict set will be preferred over ORDERING-TOPF. Among rules of equal complexity, the conflict resolution strategy prefers rules that match newer working memory elements. Since the working memory elements being matched represent process characteristics that were sequentially read from a file, the last one read was the newest; the associated rule instance is chosen to fire.

In firing, the page fault values of the current process are applied to the attributes of UTILITY. Specifically, the process' page fault value becomes the value of MAX-TOPF, 80% of the page fault value becomes the value of 80%-VALUE, and the process ID number becomes the value of MAX-TOPF-PID. The state of knowledge is therefore subtly changed and the inference engine finds all pattern matches between working memory elements and LHS elements of rules. The previously fired rule migrates to the history set and is not a candidate to fire again.

This process continues with an instance TOPF firing as long as its LHS matches some working memory element. When no more matches exist, two things happen. First the system has found the largest page fault value associated with a process and stored this in the attributes of UTILITY. Second, the conflict set contains only ORDER-TOPF, which fires.

ORDERING-TOPF changes the value of DIAGNOSIS NEXT so that neither it nor TOPF are candidates to fire. The value it assigns allows the rules TOPF-POST-PROCESS and TOPF-EXIT to enter the conflict set. TOPF-POST-PROCESS, however, will only enter the conflict set if there exist a process whose page fault value exceeds the 80%-VALUE of UTILITY. UTILITY is included as a LHS element in this rule to establish local variables as described above for TOPF.

Since the sequence of firings of TOPF and ORDERING-TOPF formed the highest page fault value, TOPF-POST-PROCESS will only be instantiated in the conflict set for working set elements whose page fault value exceed the 80%-VALUE; the highest faulting process will be identified. When the rule fires, a new working memory element will be created representing the identified process as a top faulting one. This is accomplished by the 'MAKE TOPF-PROCESSES expression. The attributes of UTILITY are again updated by the MODIFY term and the DIAGNOSIS NEXT value is reset to induce TOPF and ORDERING-TOPF to enter the conflict set. When all the processes have been found that meet the 80% rule, neither TOPF nor TOPF-POST-PROCESS will enter the conflict set in their turn. ORDERING-TOPF will fire allowing TOPF-EXIT to fire which then changes the DIAGNOSIS NEXT value to change so that this tuning analysis can continue.

The result of these four rules firing is that when they are done, working memory contains new elements, one for each process that passes the 80% page fault rule.

VII. Summary

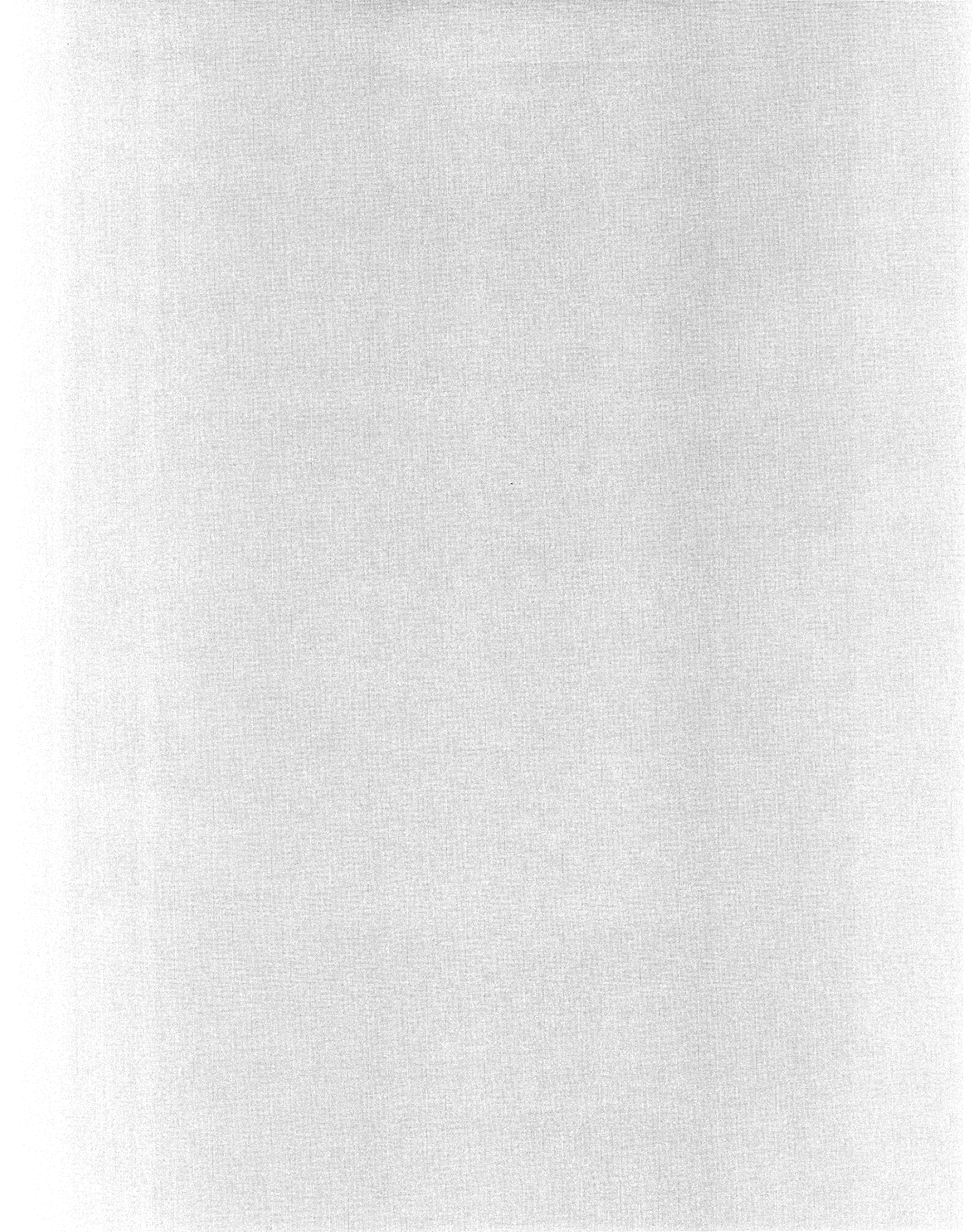
We have completed a prototype tuner that demonstrates the feasibility of the concept of automated VAX tuning. The effort required less than half a man-year. In the process, we learned a great deal about OPS5 and the art of knowledge engineering. The most important factors in the success of a KB system development project are choosing a suitable problem and having appropriate resources to solve it. The Lisp machine is a very powerful development tool with its integrated environment. Software shells take the agony out of developing KB systems by allowing the user to solve the problem at a high level. In most cases, these commercial tools are far superior to working directly in Lisp. Although not the most powerful or complete shell available, OPS5 is well suited

to many problems and it does run on a variety of machines including VAXes.

VIII. References

- (1) Digital Equipment Corporation. Guide to VAX/VMS Performance Management, Version 4.0, September 1984.
- (2) Forgy, Charles L. OPS5 User's Manual, Carnegie-Mellon University, July, 1981.

BUSINESS APPLICATIONS SIG



PROJECT MANAGEMENT IN THE NEW MICRO/MINI WORLD

Raymond J. Doubleday
Advanced Technology, Inc.
Two Shaw's Cove, Suite 205
New London, Connecticut 06320

1.0 INTRODUCTION

There are over 40 Project Management software packages currently available on the market. These packages range from the very simple and inexpensive, capable of handling only 50 events at a cost of \$80, to the sophisticated, capable of planning the construction of a space station at a cost of more than \$100,000. With this wide variety of features, functions, and capabilities, selecting the appropriate system for your needs would appear to be an overwhelming task.

The purpose of this paper is to focus on what these automated tools can do for you, the project manager; what to look for; how to define your requirements; and how to evaluate packages that might fulfill those requirements. I also hope to point out some of the gains you should expect from an automated project management system. Specifically, what I hope you get from this paper is:

- o An understanding of what you should look for in Project Management tools.
- o An understanding of whether or not you require automated project management tools.
- o An understanding of what features and tools you need to fill your specific requirements.

What you won't get from this paper is:

- o A tutorial on project management and project management techniques.
- o A recommendation of the "right" package for you.

2.0 BACKGROUND

2.1 History

Before beginning the main part of this paper, I would like to discuss how Project Management software has changed over the past years and what has happened in the marketplace to warrant a discussion such as presented in this paper.

We have been part of a revolution in computing power. We have gone from large mainframe computers to microcomputers and now, to what I would call super-micro or small mini-computers. Originally, Project Management software was developed on mainframe computers. These Project Management systems had enormous capacity for project management data and literally unlimited capacity for handling that information. These systems typically ran in a batch mode, which made them extremely slow in terms of user response. They required a "guru" to care and feed the system and to analyze the data that came out of it. The graphics capabilities of these early machines were limited, if available at all.

However, there was no meaningful limit to what these machines could do. ARTEMIS is an example of a typical project management system with this legacy, as is PSD from Cambridge, Massachusetts (see Figure 1).

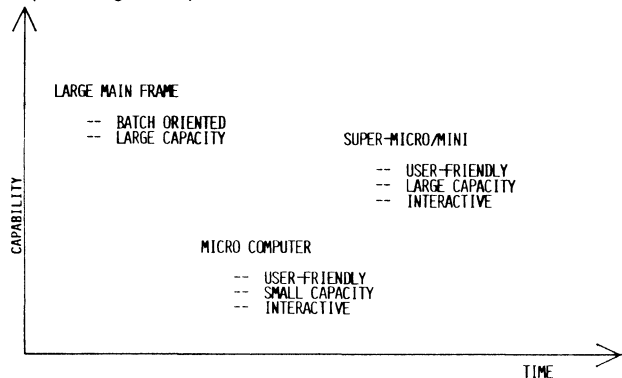


Figure 1. Automated Project Management Capabilities

However, with the advent of the microcomputer revolution (typified by machines such as the DEC Rainbow, Apple II, IBM PC, and others), we found a new kind of Project Management software. The capacity and capabilities of this software were limited; however, the packages were very friendly, easy to use, and provided immediate response for the project manager. There was no expert required to input data or interpret results; hence, the manager found a real-time decision support tool for his desktop. Typically, the graphics provided by these micros were of very poor quality (graphics were produced using either a dot-matrix or a line printer) but were sufficient to get the job done.

But, now, what do we have today? We have the super-micro, typified by machines such as the DEC Professional 350, the IBM PC XT/AT, and the MicroVAX I and II. Typically, these are the fast, powerful, single or few user machines with a large storage capacity built in. What has happened is that we have regained the data storage and speed of the mainframe computer.

Fortunately, current software has been able to maintain the user-friendliness of the micro machine. We now have real-time decision support software that is easy to use and has no realistic limitations to the quantity and complexity of data that can be handled.

The current systems are also able to generate high-quality graphics. Now we have the best of both worlds: we have a machine at the project manager's desk with the capacity of a mainframe and can provide him with real-time, real world answers to his project management needs.

2.2 New Ideas

I would like to propose two themes for the evaluation of all tools and controls to be discussed in the remainder of the paper. These themes are abstraction and communication.

In everything that you do in a project, a software development program, or real life, it is important to be able to break the project into manageable, definable, understandable tasks (i.e., abstraction). Then, it is equally important to be able to meaningfully communicate that information.

There are three major features that should be part of the fundamental design of any Project Management package. These three features carry through the fundamental theme of abstraction and communication.

2.2.1. Abstraction. A package should support the concept of abstraction. By being able to abstract a project, you are able to take multiple-level views of your program (i.e., decomposition). Then, you can deal with it from the beginning (the Concept stage), through other successive levels of detail, down to the last possible level of detail (such as fabrication and assembly of a product).

2.2.2. Representation/communication. The choice of activities and milestones must be such that their representation on paper can be used as a means of communication. This is important because unless you can communicate the needs of the project to your staff, nothing can get done. Communication must be clear and unequivocal.

2.2.3. Manipulation. The automated tools must act on these representations of activities and milestones to ensure consistency, feasibility, and, most of all, achievability.

When looking at Project Management tools, you should look at the tools in the light of these themes as stated above.

3.0 PROJECT MANAGEMENT

This paper is not meant to be a tutorial on project management, but I would like to briefly go over what project management is to establish a common framework. The point of this paper is to highlight the benefits of automated project management and the gains that are achievable through the use of project management.

A project can be broken down into five major phases: Conception, Planning, Scheduling, Monitoring, and Action. Very often action involves the replanning and rescheduling of activities, as shown in Figure 2. We will look at each phase in detail from the point of view of Project Management systems.

Project Management systems have two major functions. They can be used either as tools or as

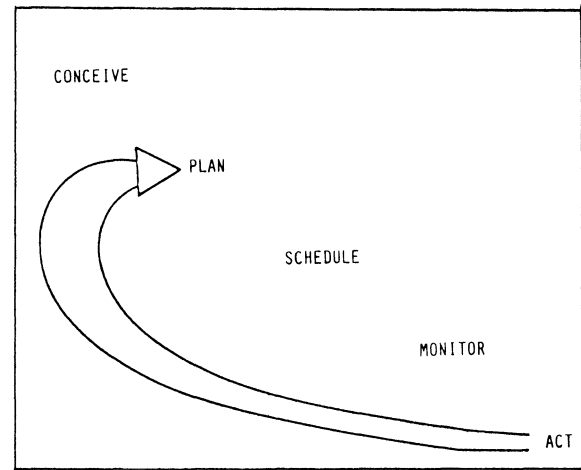


Figure 2. Project Phases

controls. As tools, they help you to organize, plan, and schedule; as controls, they monitor progress of the program (in terms of time and money). Tools help you to plan; controls tell you if your plan is working. If you are evaluating a feature of a Project Management system to be used as a tool, you should ask yourself how it will help you to plan your project; if you are evaluating a feature to be used as a control, you should ask yourself how it will help you to monitor your job.

3.1 Concept

The first phase of the project is the Conception phase. This is the definition of the program or the project and, in fact, becomes its charter. There are certainly no computers here; this is where insight, intuition, and depth of human understanding play a part in defining the project, its goals, and its requirements. This is where the goals of the project are established and the tempo of the program set.

3.2 Planning

3.2.1 Work Breakdown Structure. The planning stage is the decomposition of the project as conceived into its logical structure. In the initial planning stage, no schedule or resources have been assigned yet.

Top view planning. This is the first place a computer Project Management package should be able to do something for you. First of all, it should support multiple views of the project and secondly, have the capacity to move down the project in detail. This is analogous to a top-down step wise refinement of the project. This is a place where the concept of being able to abstract a project or to push down the details of the project becomes very important because what you want as a project manager is to deal with a larger picture first and then to fill in the details of each phase. In essence, you are creating a management outline for your project managers to complete; and they in turn may provide the same sort of outline to their subordinates.

Let's look at what a typical software development project might look like as shown in Figure 3.

ASTROLOGICAL ORGANIZATION

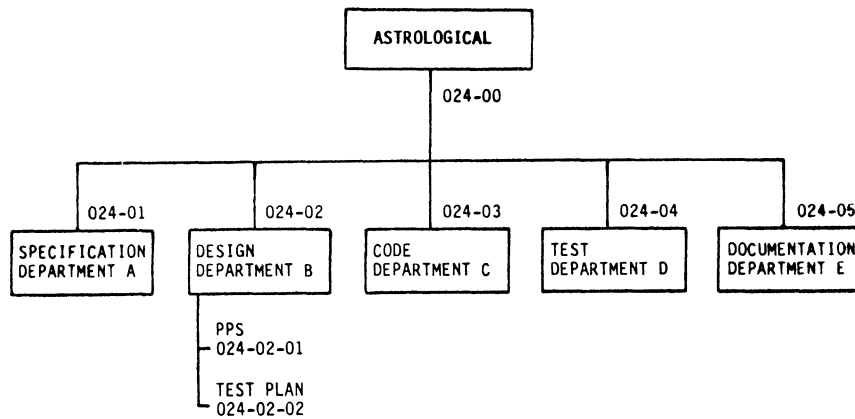


Figure 3. Astrological Organization

This is the development of a program called Astrological to analyze digitized images of the night sky. The product breaks down into typical software development components. The specification, design, coding, testing, and documentation. The package is meant for in-house use; therefore, the manufacturing and marketing functions are not included on this particular product. After the concept development, the next thing that the senior manager must do is to assign responsibility for each of these major phases to a person or department and then produce a rough schedule or goals for the project completion. Once this preliminary schedule and assignment have been achieved, the senior manager will ask the department managers to produce their own schedules, budgets, and resource requirements within the limits of their schedule.

How do you do this? You do this by having a project management package that supports various levels of hierarchy. One way to do this is through the use of work breakdown structure numbers, although there are a number of other schemes that may work equally well. Briefly, work breakdown structure is a hierarchical numbering system similar to the concept of a work outline where the order and the number that each work assignment has has meaning. Typically, a work breakdown structure number is associated with the concept of a work package, which is the smallest measurable unit of work. In our example, Figure 3, the Astrological analyzer is given the number 024. This code indicates that this particular software product is one of at least 24 different jobs that are taking place or have taken place within the organization. Looking underneath that, we see that the number 024-02 is the design function for Job 024. Looking at the design function in more detail, we see that the preparation of the program performance specification is given the number 024-02-01. Development of the test plan is given the number 024-02-02. It is possible, of course, for this numbering scheme to continue down in more detail as required within each function and, of course, to go across to support more than the five functions shown here.

Why is this work breakdown structure important? It is important for two reasons. First of all, it

allows you to assign responsibility and a budget for a category of work such as the specifications 024-01 to Department A for completion. Secondly, it allows you to isolate your view of the project to the higher level. From now on, you as senior manager, will only be looking at things down to the second level; that is, you will be looking at tasks 024-01, 024-02, etc., leaving the specific details of the project to the managers of each of those particular departments. Your management, in turn, may look at Jobs 022, 023, and 024 to supervise the overall performance of the departments.

The next thing you should look for in a Project Management package is the ability to support various levels of hierarchy through the use of work breakdown number structuring or other means.

3.2.2 PERT/CPM. Now that we have established the major phases of the program, we need to go into more detail on how the Astrological program can be realized. The major tool you have available for this is network analysis. Network analysis is also known as either PERT (Program Evaluation and Review Technique) or CPM (Critical Path Method).

PERT/CPM are synonymous today; we will use the term network analysis to stand for a combination of PERT and CPM. The idea behind network analysis is to represent a complex project as a series of interconnected activities that must be performed. The description of the project is then used to analyze the project and answer the following questions.

- o How long will the project take?
- o Which jobs are most critical to the project?
- o How should the project be scheduled?

An activity is a time/resource consuming event in the project. I will use the arc in my discussion to represent an activity. A point in time corresponding to the start or completion of an activity is a milestone; they are represented by a triangle on the schedule. On a network drawing, they are represented as nodes or circles (note, however, that all nodes are not necessarily milestones).

Now, let's look at the network for our Astrological package as shown in Figure 4.

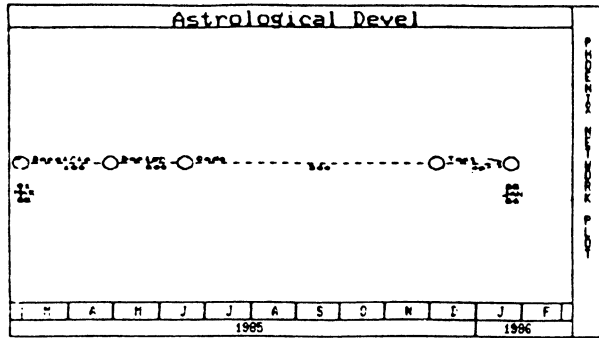


Figure 4. Astrological Development

What is wrong with this figure? Well, fundamentally, it is too simple; however, to introduce the detail necessary to understand the project from beginning to end would be too hard; the graph would be too hard to read, the program would be too hard to manage and control.

Again, we must be able to do a top-down refinement of the tasks. Tasks at a higher level can be broken down and should be broken down in order to understand the problem. Figure 5 shows an example of the proper kind of decomposition when applied to writing a book. As you can see from the figure, the book has been divided into a number of chapters, each chapter into a number of sections, each section into a number of paragraphs, and each paragraph into a number of sentences. This, of course, is a very manageable approach with the appropriate work breakdown structure numbers being shown in the right part of the picture.

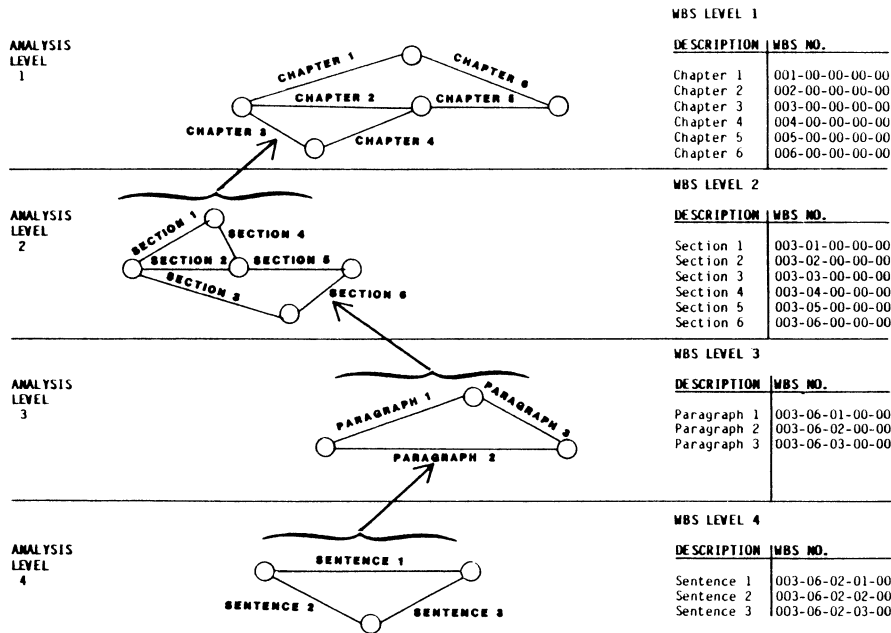


Figure 5. Network Decomposition

Returning to our example, Figure 6 shows the code portion of our task broken down into more detail beginning with the review of the performance specification and ending with the final integration of the package.

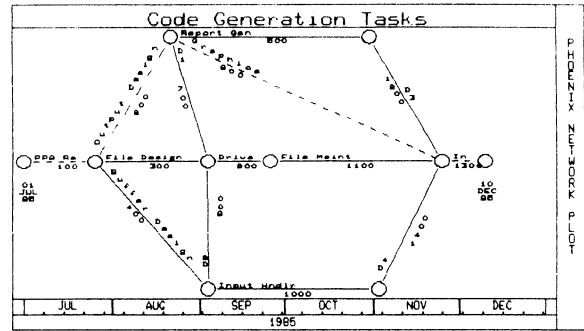


Figure 6. Network Plot

What have we done? We have been able to isolate our tasks into the correct areas of responsibility and we have been able to decompose the coding job to manageable units. If you were the head of the programming department, you might want to have even more detail for a particular task such as the coding of the input handlers and you could, in fact, do that for yourself. The output for this section, the overall time from the beginning to the end, can now be passed back up the management chain and the time put in for coding on the network drawing as shown previously in Figure 4.

In examining Figure 6, our tool has answered the first two questions: how long will it take and which jobs are critical to the project.

Now that we have our network drawing, what other kinds of planning tools are available? Next is a Gantt chart, shown in Figure 7.

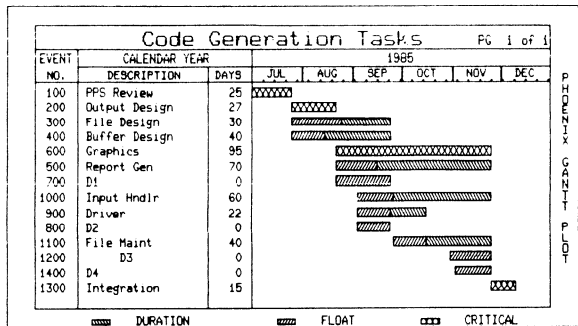


Figure 7. Gantt Plot

The Gantt chart is the first depiction of a schedule. The critical activities are shown in red on the Gantt chart as they were shown in red on the network drawing. Additionally, there should be a number of tabular reports provided with the network analysis to bring out the necessary detail in order to properly analyze the schedule. The kind of reports that you should expect to see again support the ideas of abstraction and decomposition, and are listed in Figure 8. There should be an executive summary, something that provides an overview of the time and resources consumed for the project, and a variety of reports getting down to a final detailed report showing for each of the activities the time estimates, the scheduled early start and late start, the early finish and late finish dates, as well as the float, the slack time, and the identification of the activities and resources that are critical to the time of completion of your task.

- o DATA SUMMARY
- o EXECUTIVE SUMMARY
- o DETAIL REPORTS
- o CRITICAL PATH REPORT

Figure 8. Network Analysis Management Reports

This in essence becomes the plan for your project. However, it is necessary now to generate a firm, fixed schedule or baseline.

3.3 Schedule

The Gantt plot is a candidate schedule. What you must do is use it to develop a firm, fixed schedule or baseline. The final schedule represents the plan of the Gantt Plot, with real-world constraints applied to the plan. This schedule is one that you will manage to and report on. All your progress will be measured against this baseline schedule. The schedule for the coding effort is shown in Figure 9.

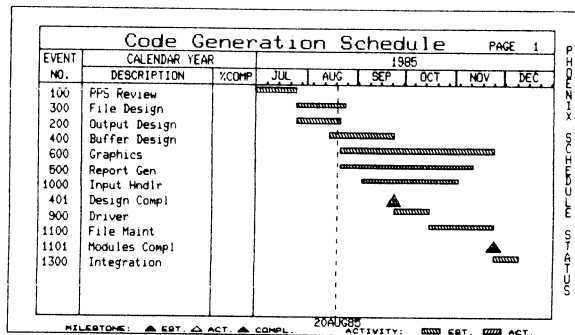


Figure 9. Schedule Plot

3.4 Monitor

3.4.1 Controls. You have passed the planning stage. Now that you have established a schedule, you need to have a number of automated project controls that will let you examine the schedule and examine the financials for your project to make sure you are both within budget and on schedule. Before we begin discussion of some controls you should look for in a project management package, let's take a look at our project.

First of all, the job spans five departments. The initial time estimates were that the job would take two years to complete, cost \$1.5 million dollars, and would be composed of approximately 5,000 separable and discrete activities. Given this size, how are you going to control it? Well, taking a step back, you have to look at why you are a project manager. Most likely, it is because of your ability to thoroughly understand your job and to almost be intuitive about the nature of the work you do. A project of the magnitude of Astrological would require a database so large it would negate your ability to be intuitive. What you need from a project management package is the ability to be dynamic in monitoring your project to be able to develop various views downward into the database until you can focus on the issues that are pertinent to the project. You need to be able to select or segment the database so that you can get an accurate, concise view of a limited segment of the database.

Again, this supports the concept of abstraction. You want to be able to look at the data in varying degrees of detail; only the detail necessary to give you the insights that you need to do your job. Your Project Management controls should provide unlimited query capability on the database.

3.4.1 Schedule Status. The first thing you should look at is the schedule. This is shown in Figure 10, which is a schedule with milestones for monitoring the progress of each task.

In this particular example we are showing a graphic depiction, one that is very important and gives a quick indication of how we are doing and where we should be today for the project. As you can see, immediately below the baseline schedule is the actual start and completion of each of the activities in the project as well as percentage complete. The percentage complete for each task is indicated by how much of the lower bar is filled in.

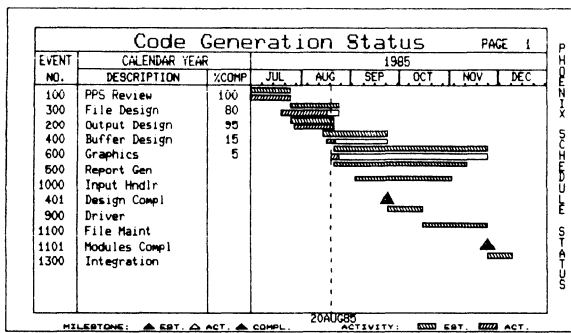


Figure 10. Schedule Status Plot

3.4.2 Completion Status. The second chart, Figure 11, is a Completion Status Plot which gives us another view of the data. It indicates which events are early, which events are late, percentage complete, and how many days remain until the completion of the job.

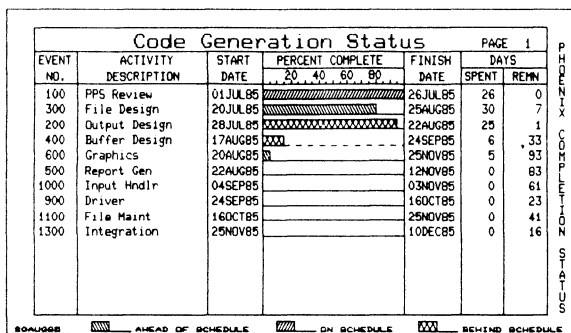


Figure 11. Completion Status Plot

3.4.3 Cost Status. Figure 12 shows a Cost Plot, which is a measure of the budget, the dollars spent, and the work achieved for those dollars spent.

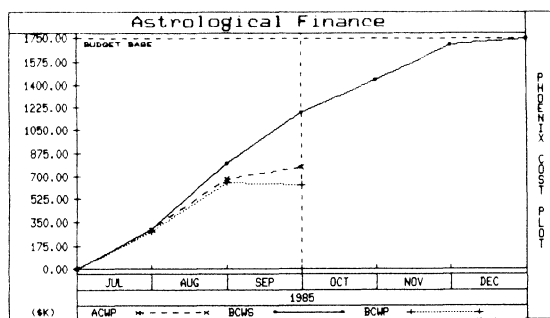


Figure 12. Cost Plot

This plot gives you a feel for the rate at which the funds of the project are being used and the amount of work that is actually being performed for your project. This brings up a number of ideas, such as the budgeted cost of the work scheduled, the budgeted cost of the work performed, and the actual cost of the work performed.

3.4.4 Cost Variance. Figure 13 shows a Cost Variance Plot. It is the difference plot of the data that was previously shown in Figure 12 and gives us a measure of how well we are progressing against the schedule. The closer these curves are to zero, the more accurate were our project predictions and the better our project performance.

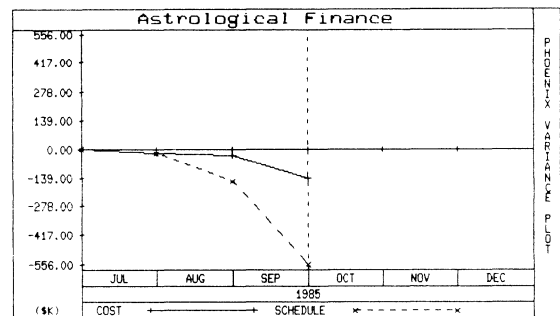


Figure 13. Cost Variance Plot

3.4.5 Communication. What is significant is that the previous four figures provide accurate and timely information that may be communicated easily. Large stacks of computer runs are not required, and it is not necessary to connect dots and asterisks because the plot was prepared on a printer. The control reports provided are presentation-quality graphics.

The monitoring tools that you should select should be suitable for all levels of management. In management reporting, you certainly don't want to have separate tools for different levels of management. What you should expect from your project management tools is that for high level meetings, briefings, and presentations, they should support full-color graphics with figures that are easy to read and understand. They should be crisp and to the point. For reports, figures should be done in black and white so they can be clearly reproduced by either printing or copying.

Your project controls should also support graphics with tabular reports which contain all necessary back-up data.

3.5 Act

Management must manage. Now that you have read the reports, reviewed the project data from your subordinates, you must identify the causes of any problems and act on them. Therefore, it is very important that the project management system you select be able to perform what-if analyses to aid in replanning and redefining the project as it progresses.

At this point, the idea of representation of the ideas and their automated manipulation becomes very important. You must be able to easily manipulate the parameters of your program and perform rapid what-if analyses until you have developed an adequate approach to your problem. You must then be able to modify schedules to accommodate this replanning just as easily. With replanning, the cycle begins again.

4.0 CONCLUSIONS

The latest generation of Project Management software has the power and capacity of main-frame type packages and the ease of use of micro-computer software.

Any Project Management system you select should:

- o Be easy to use.
- o Support multiple views of the database (abstraction).
- o Provide presentation-quality graphics (communication).
- o Provide real-time analysis (monitor).
- o Support rapid what-if analyses (plan and replan).

By Chandan W Seernani
Ambase International Corporation
Atlanta, Georgia

ABSTRACT

Many software vendors that are successful in the US market are disappointed because they do not enjoy same amount of success in the international market. This paper discusses the following problems and their possible solutions :-

- Implementation difference
- Marketing strategy
- Software support overseas
- Government red tape

INTRODUCTION :

The concept of packaged software is here to stay. There are several companies in this business and many more are venturing into it everyday. Many companies that are successful in the US market are very disappointed to discover that they do not enjoy same amount of success in the international market. There is a multi million dollar market for these products overseas. There are several factors which one must consider before investing in any such venture. I will be talking about a few important ones, these are:-

- Implementation differences.
- Marketing strategy.
- Software support overseas.
- Government red tape.

IMPLEMENTATION DIFFERENCES :

This is one of the major factors we overlook while designing a software package. We must remember that some concepts are implemented differently in various countries. A simple example is that of a date. In the US the acceptable format is MM/DD/YYYY. Some countries use the format DD/MM/YYYY or YYYY/MM/DD or YYYY/DD/MM. When we design routines that manipulate dates, we must consider formats other than those acceptable in the US.

Another major difference is the language. We must remember that English is not a universal language. Prompts, defaults, validity lists and help text information could be external to the program. The program could be made smart

enough to retrieve this information and process it. Hence, if we want our system to prompt the operator in the German language and to interpret the responses that will be supplied in German we can do so because the information in the external file can be changed from english to the desired language. Hence the program will do the following in desired language :-

- Prompt the operator
- Display the default response
- Check against a list of valid responses
- Display help message

You may use the example in figure-1 as a guideline to design data entry screens for your system.

Many software packages offer code generation capabilities. Code generated by such facilities is fairly generalized and hence inefficient when executed. In several countries where Machine costs are high and the cost per CPU hour ranges anywhere from \$90 to \$150 it is important to assure that software is efficient. Generators are useful in countries where manpower costs are higher than machine time costs but they are practically worthless, where manpower costs are lower than machine time cost. We must design our package so that it is broken into several modules and the user buys only what he needs. It is recommended that each module be sold separately.

MARKETING STRATEGY :

There are several different strategies we can adopt to market the product. I

COMPANY INFORMATION	
1) Reference Number :	_____
2) Company Name :	_____
3) Address line 1 :	_____
4) Address line 2 :	_____
5) State / province :	_____
6) Country :	_____
7) Postal code :	_____

##) Prompt line used with bottom of screen entry

Special messages

?
< Default >

MESSAGE AREA

You may want to consider the following suggestions while designing data entry screens for your software :-

- 1) Row 1 could be the SCREEN BANNER. Information like menu option, description of this screen/menu option, current system date and system time could be displayed on this line.
- 2) Rows 2 thru 17 could be reserved for programmer use.
- 3) Row 18 could be reserved for printing a descriptive version of the prompt currently being processed. Note that '##' is number of the field currently being processed.
- 4) Row 19 could be reserved for special messages printed by the software.

These could be printed when an operator invokes a special command or an error occurs during data entry.

- 5) Row 20 could be reserved for entries made by the operator in response to the current prompt.
- 6) Row 21 could be reserved for display of default response to the current prompt.
- 7) Row 22 could be a blank line to serve as a spacer between the default display line and the message area.
- 8) Rows 23-24 could be reserved for printing help/aid messages. We might consider defining these as scrolling region incase help messages are over 2 lines long.

FIGURE-1

will discuss them one at a time along with their respective pros and cons.

1) The Company could develop its own marketing force. Some of the advantages of this approach are :-

- a) The Company controls all its International operations from US.
- b) The company does not have to share its profits with any third parties.

However this approach has many disadvantages. These are:-

- a) Marketing trips abroad are expensive.
- b) It is expensive to get in touch with every prospect due to high costs of calling overseas.
- c) We may have an inaccurate perception of the market. One way to solve this problem is to conduct a market research, but the cost of such a survey can be prohibitive.
- d) We may not be aware of import regulations in each country. This point will be discussed later in more detail.
- e) Software support to every client abroad can be very expensive.

2) The company could have a distributor in every country who could operate on a commission basis. The main advantages of this approach are :-

- a) Money spent on marketing trips abroad is saved.
- b) The distributor is locally situated, hence he has a better idea about the market.
- c) Since the distributor is working for the company, control is still maintained from the US, this assumes that the security built into the package is tight enough.

Major disadvantages of this strategy are:-

- a) Software support overseas is a major problem that the company must tackle while designing the contract with the distributor. This point

will be discussed later in more detail.

b) The problem of import regulations still exists, however now we have someone who knows the rules of his country.

3) Sell international rights to one company and let this company form its own network of distributors.

Some advantages of this approach are :-

- a) Money spent on marketing trips abroad is saved.
- b) We have only one distributor to deal with, for the entire international market. In the previous approach we had to deal with one per territory.
- c) Control is more centralized.
- d) We might also reduce software support costs, if we design the contract correctly.

Most of the disadvantages mentioned earlier still exist. However, when a big market is given to one company chances are that this company will control the enhancements and future direction of the product. When stakes are high the vendor normally tends to comply with special requests.

4) Locate distributors in each country and sell them the package on an outright basis for that country. This means that we must give them source code. In this case the price tag should be high enough so that there is scope for profit, but at the same time, low enough so that the distributor feels it is a good bargain for him.

The main advantages of this approach are:-

- a) Now the distributor has the source code so he can modify it to suit the needs of his market. Hence, the software will not contain pieces of code that are irrelevant to that environment. Hopefully, the package will execute more efficiently. Also, now that the package will be sold in local currency the seller can price it right.

- b) The problem of import licence will disappear because the package is now being sold locally. However the buyer has to obtain the approval while purchasing software from the vendor.
- c) The software support problem will disappear. The seller now has source code and he can support the package locally.

Some of the disadvantages of this approach are :-

- a) Since the package is being sold on an outright basis it generates only a one time revenue for the vendor.
- b) Since the distributor has to pay a high cost, hence he has to justify a higher amount for an import licence. This can sometimes pose problems.

SOFTWARE SUPPORT OVERSEAS :

Marketing strategy 2 & strategy 3 mentioned above require that support be considered as a very crucial issue and all ambiguities must be resolved in writing. The main area of concern is who would support the software? It will be too expensive for the vendor to do it, however, if the distributor has to support it then he has to be supplied with source code. I am sure nobody would like to share source code with the distributor, as this can be misused in many ways. To get around this problem the contract must be designed such that all support calls go to the distributor, who acts as a buffer between the vendor and the end user. If the distributor is sure that it is a software bug, then he could contact the vendor to fix it. The contract must have penalty clauses clearly stated, if this rule is violated.

GOVERNMENT RED TAPE :

In several countries import regulations require that one has to go through government red tape to obtain an import licence for spending money from that country's foreign exchange reserve. Formalities vary from one country to another. Also the high price of US dollar in the international market may make it all the more difficult for an end user to obtain an import licence. A good way to help the end user will be to sell your package in modules. Modules like the code generators may be sold separately so that the end user pays only for what he needs and not for bells and whistles in the system. This will reduce the amount he may have to justify

to his government and increase the probability of obtaining the licence.

CONCLUSION :

I would like to conclude by saying that this paper is not intended to scare away software vendors from entering the international market, but instead it is meant to warn them about some of the problems they may encounter while doing so. I think it will be appropriate to say that US Department of Commerce prohibits sale of Hi-technology to some countries. One must check with them before signing any overseas agreement.

By Chandan W Seernani
Ambase International Corporation
Atlanta, Georgia

ABSTRACT

This paper discusses how productivity tools help a company cope with the following problems that are common in most DP shops today :-

- Technical manpower shortage
- Standardization
- Support/maintenance
- Application backlog

INTRODUCTION :

What is a productivity tool ?

For this paper a productivity tool will be defined as any tool that helps increase productivity and reduce maintenance overheads. Some of the examples are :-

- DEC's DATATRIEVE.
- Any kind of application generator.
- Any kind of program code generator.

We are aware that a productivity tool that does 100% of what you want it to do does not exist. However, if a proper tool is selected for your shop it must help you cope with the following problems that are common with most DP shops :-

- Technical manpower shortage
- Standardization
- Support/maintenance
- Application backlog

We will now take each one of these factors and talk about them in a greater detail.

TECHNICAL MANPOWER SHORTAGE :

During application development there are two major questions that must be answered. These are :-

- WHAT will be accomplished by writing this application ?
- HOW will we do it ?

The first question is usually answered by potential end users of the application. The technical experts are required to give advice on the various technical considerations. For example hardware and software constraints. To answer the second question we need a programmer. In a shop that does not have productivity tools every program is written from scratch. Some shops have source code libraries which help. However, writing every program from scratch is like inventing the wheel every time. As we have mentioned before a productivity tool that does 100% of what you want does not exist. We must therefore select a tool that :-

- Is easy to use. To assist new users it must have sufficient tutorial text preferably in a multi-level format with each level providing additional detail about the prompt.
- Must allow you to do proto typing easily. This has the following advantages :-
 - Gives the ability to quickly and easily create a program for user testing & approval.
 - If changes are requested a new prototype can be created quickly for user testing and approval.
 - Substantial user involvement via prototypes insures fewer changes after the system has been completed.
 - Being involved with development and testing gives the user confidence that the system will satisfy his needs.
 - By the time the project is completed the user is familiar with the operation of the system.

- Should be definitional rather than procedural because :-
- Entering a definition of WHAT the program does is less time consuming than writing source code which details HOW the processing occurs.
 - One definition parameter may expand into multiple procedural steps. This expansion must be done by the tool to minimize development time per program.
- The generated code must be user friendly for an end user. This means that it supports the following as standard features of the generated code:-
 - Several kinds of terminals.
 - Commonly used commands like the HELP command.
 - Cursor control keys to move around on the screen.
 - Any standard screen function like Reverse video, Bold video etc.
 - Capability to refresh a data entry screen incase someone sends messages during a data entry session.
 - Same program must be able to input different sets of data using either overlap screens or windows on the same screen.
 - Complex data verification and validation. In some cases it may involve looking up several files to achieve this.
 - Handling conditions that may be formulated at run time.
 - Security while processing fields or while processing screens. Some examples are :-
 - Conditional skipping of fields. These may either be predefined conditions or conditions that are formulated at run time.
 - Display only fields. The operator is not allowed to alter the value of such a field.
 - Required fields. These are fields for which an operator must enter a valid response before he is allowed to exit from that screen.
 - Jump back fields. The operator will be allowed to alter the value in such a field only if he jumps back to it from another field. During normal processing the default value in this field is displayed and the program moves on to the next field.
 - The generated program must be able to switch itself from production to test mode when desired by the user. This will help a company train new operators directly on the programs they will be using on a day to day basis. In test mode the program does everything else but store records. This will insure that all the files are intact.
- The generated code must be broken up into several modules. These must be logically and physically independent. E.Yourdon explains the advantage of dividing a program into modules in his basic theorem of software engineering. This is stated as :-

$$C(P + Q) > C(P) + C(Q)$$

This means that the cost (C) of solving a problem (P + Q) as a whole is more than the cost of solving it's parts (P) and (Q) seperately.
 - If the generated program follows this fundamental theorem then it is easy to have programmer hook areas. The programmer only codes these areas. As a result of this less time is spent in development and support/maintenance of the application.
 - The generated code must have sufficient comment lines so that the code is well documented.
 - In short the tool must do as much work as possible so that the programmers have lots of time to concentrate their creative efforts on the functional contents of the program where it is most needed.
- STANDARDIZATION :**
- We are all familiar with the high rate of manpower turnover in our industry. We also know that different programmers have different coding styles. A shop without productivity tools will typically have programs written by different people using different styles and structures. A new programmer in the organization will find it very hard to maintain and modify all that code. Some of the advantages of using productivity tools are :-
- Generated code always has the same style and structure; the differences occur only in functional content.
 - Programming errors are minimum because most of the program consists of shells and external subroutines that have been tested over and over.
 - Since the generated code is standard, a maintenance programmer can move from one application to another with ease.
 - Standard code also eases the task of training a new programmer in the shop.
 - Standard code eases the task of modifications and enhancements.

SUPPORT/MAINTAINENCE :

This is another area in which productivity tools will help your organization. Generated programs as a class, are easier to support & enhance than custom programs because :-

- A manually produced program differs from all others in style, structure and techniques. A generated program on the other hand is standardized.
- The programmer writes minimum code since most of the program is in form of pretested shells and external subroutines. This reduces the possibility of programmer error.
- The code in generated programs is well commented, whereas in case of a manual program the programmer may be a bad documenter.
- Since the generated code is broken into modules the programmer doing maintainence looks only at small pieces of code at a time. Yourdon's theorem described above can also be applied in this case.
- If we were to draw a graph of Effort vs Complexity as shown in figure-1 then beyond the point of intersection (this is point of rewrite) the cost of rewriting the software will be lower than cost of continued maintainence.

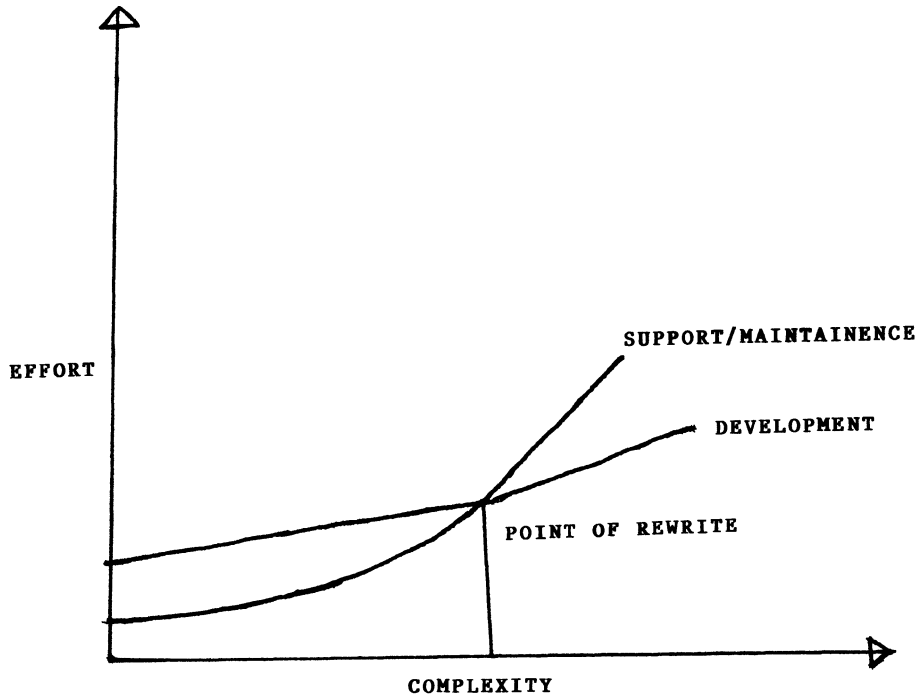
APPLICATION BACKLOG :

There are 2 types of application backlogs, namely :-

- New development
- Rewriting systems that are beyond the point of rewrite

As discussed before, manpower shortage demands that we increase programmer productivity as much as possible. Studies show an improvement ratio of 1:10 can be easily obtained by using productivity tools. Hence we conclude that productivity tools will help us cope with application backlogs because program development will be much faster than before for following reasons :-

- The productivity tool is definitional rather than procedural.
- The tool automatically breaks up the defination into procedural steps hence the programmer does not code every detail of the program.
- Since most of the code is in form of shells the programmer does minimum coding. This reduces the possibility of programmer errors and speeds up development.
- The shells are pretested. This reduces the time that would be spent on debugging.



Effort vs Complexity

FIGURE-1

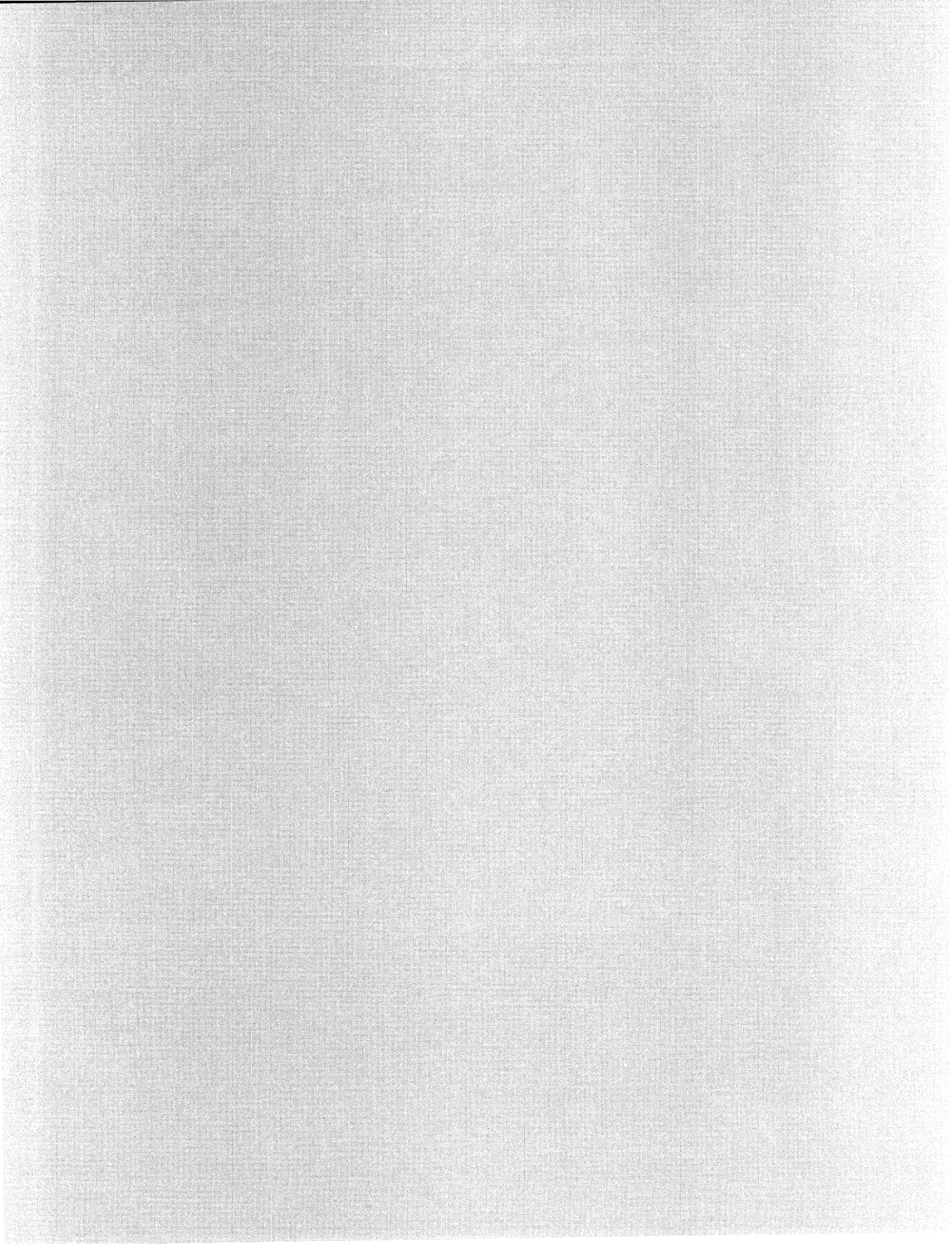
CONCLUSION :

In conclusion I'd like to say that a productivity tool can make or break your development efforts. One must evaluate several tools before investing in one. Some of the questions to ask a vendor would be :-

- References of people currently using that tool. You must call these people and ask them pointed questions like :-
 - What they like about the product and what they don't like about it ?
 - What kind of response do they get when they call for support ?
 - Are the support people knowledgeable about the product ?
 - What kind of training facilities does the vendor have ? If this customer has used any of these then ask for an evaluation of these facilities.
 - How often does the vendor provide enhancements and bug fixes to the software ?
 - Does the vendor usually keep the promises that are made at the time of sale ?
 - Confirm some of the things that the vendor has told you with these customers.
- Check the vendor's credentials. This will include things like :-
 - How long have they been in business
 - Where did their top level management come from ?
 - Do they have any law suits pending against them ? If yes, probe for details.
 - Has this vendor filed for bankruptcy before ? Or have they ever been in financial trouble before ? If the answer is yes, probe for details.
 - Have they been sold to any other corporation ?
- Upward compatability of the product ?
- What is the future direction of the product ?
- Visit the vendor's site.

These may appear to be fairly tough questions to get answers for from various people but it is better to be cautious while buying the tool rather than being sorry after you have bought it.

COMMERCIAL LANGUAGES SIG



COBOL: An Endangered Species?

Edward W. Woodward
Computer Sciences Corporation
443 Inyokern Road
Ridgecrest, California 93555
(619)/446-6585

ABSTRACT

Controlling the life cycle of a commercial project continues to be a critical professional task. One of the most costly elements of this activity is the time used in performing the program maintenance function. This paper will point out some of the reasons why continued use of COBOL as a commercial language may be in jeopardy. We will then look at a variety of techniques which can go a long way toward improving the efficiency of program development throughout the project life cycle.

1 INTRODUCTION

Several developments are currently contributing to the uncertain future of COBOL as a commercial programming language.

1.1 RISING SYSTEM DEVELOPMENT COSTS

The rising costs of developing and maintaining computerized business systems are threatening the way programmers operate. Throughout the industry, the steady rise of development costs using COBOL is endangering the continued use of the language.

1.2 HOW DOES COBOL MEASURE UP?

COBOL was intended to possess a number of benefits for the programmer as well as the business manager. The language is not living up to the expectations. The following list contains a few of the benefits expected from COBOL:

- (a) Satisfy user needs and decrease costs.
- (b) Produce an English-like programming language.
- (c) Provide code which would be readable and maintainable.

1.3 INTRODUCTION OF FOURTH GENERATION LANGUAGES

Several languages have been developed which significantly reduce the time needed to develop business applications. These Fourth Generation Languages, which include DATATRIEVE, are high productivity tools.

2 HOW DOES THIS AFFECT THE COBOL PROGRAMMER?

These trends in the industry are having a significant impact on the future of COBOL programming. We operate in a profession which is subject to continual change. The future of a programming language which fails to serve user needs in an efficient and cost effective manner is doubtful. It is important to recognize these changes and protect ourselves from their effects. There are actions that the COBOL programmer can take to postpone what would otherwise be a limited future.

3 HOW CAN COBOL BE IMPROVED?

To alleviate these problems with COBOL, it is important that programmers use the productivity tools available within the language. This paper will focus on those techniques which can be easily implemented within the body of the COBOL source code. In this way, the techniques will not inhibit the development of an application on a multi-programmer project.

These techniques are not complex and neither are they obscure. Each one is easy to use and is probably fairly well known by a majority of the COBOL programming community. The intent of this discussion is to remind us all that these tools and techniques are available. In addition, it is hoped that this discussion will encourage all of us to consider the possibility that there are other and even better ways of developing systems using COBOL.

There are a number of sources available to the COBOL programmer who wants to improve the quality of program code. The obvious sources for this information are the COBOL manuals and seminars provided by the computer hardware/software manufacturer. There are also the books and technical journals which focus on the topic of program development and efficiency.

A number of techniques are discussed on the following pages which can significantly improve the productivity of COBOL and reduce the cost of developing systems using COBOL over the project life cycle. If these ideas along with other productivity tools within COBOL are used, the future for COBOL can be markedly enhanced.

3.1 USE PREAMBLE COMMENTS IN THE IDENTIFICATION DIVISION

THE NORM

The COBOL programmer has tended to stick with only the required statements. Very little information has been provided in this area of the program. The future maintenance programmer is generally left with the task of trying to determine what the program is doing. This becomes tantamount to redeveloping the program. The thought processes and research required for developing the program must be repeated during the maintenance process.

It is not uncommon to see programs which contain only the required statements in the IDENTIFICATION DIVISION. What we find is no information about the purpose, restrictions, interfaces or requirements of the program. The following illustration provides a graphic example:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.                CREATE_INVOICE_DATA INITIAL.
```

SUGGESTED IMPROVEMENT

In contrast, the following example gives more information and provides the programmer with a better understanding of the purpose and overall scope of the program. With the use of this technique, the maintenance staff has a good overall view of what the program does and how it is done.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      CREATE_INVOICE_DATA INITIAL.
***
*** TITLE:      CREATE INVOICE DATA
***
*** USAGE:      $ RUN CREATE_INVOICE_DATA
***
*** PURPOSE:    This program is used to accept the sales
***             invoice data, validate this data and
***             organize it into the sales invoice file.
***
*** LIMITATIONS: There are no limitations placed on the
***             execution of this program.
***
*** WARNINGS:    There are no special resources required
***             by this program.
***
*** SUBPROGRAM REFERENCES: This program uses no called modules.
***
*** ALGORITHM:  None.
***
*** NOTES:      None.
***
*** WAIVERS:    No waivers exist for this program. All coding
***             conforms to department standards.
***
*** ENVIRONMENT: VMS      Version 4.2
***             COBOL    Version 3.2
***             FMS      Version 2.1.
***
*** RECORD OF MODIFICATION:
***
***             PROGRAMMER:  DATE:      MODIFICATION PERFORMED
***
*** 1. Ed Woodward  03/05/86  INITIAL EDIT.
*** 2.
***
***
*** AUTHOR.      Ed Woodward.
*** INSTALLATION. Computer Sciences Corporation.
*** DATE-WRITTEN. March, 1986.
*** DATE-COMPILED.
```

NOTE

Under the new COBOL-85 standards, AUTHOR, INSTALLATION, DATE-WRITTEN and DATE-COMPILED are obsolete reserve words. These too can be added as actual PREAMBLE COMMENTS.

3.2 USE EXTERNAL ASSIGN STATEMENTS

THE NORM

The tendency is to modify programs for changes to filenames. This then requires the programmer to recompile the source program. If a program can be developed with flexibility in mind, future maintenance can be reduced.

SUGGESTED IMPROVEMENT

An improvement on this would be to set up an external logical for the filename. In this way an external ASSIGN could be used. This would not require program modification or program recompiling. It will reduce the possibility of making the wrong change to a program or inadvertently modifying code which does not require change. An example of this technique is show below.

Command Procedure Using DCL

```
#!/
#!/  The PREAMBLE COMMENTS for this command procedure are located
#!/    at the end of this file.
#!/
#!/
$ ASSIGN CUSTOMER_INVOICE.DAT      INVOICE
$ ASSIGN PRINT_FILE.DAT           PRINTFILE
$ RUN CREATE_INVOICE_DATA
$ EXIT
$
#!/
#!/          PREAMBLE COMMENTS
#!/          .
#!/          .
#!/          .
```

////////////////////////////////////

CREATE_INVOICE_DATA.COB Source File

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.          VAX-11.
OBJECT-COMPUTER.         VAX-11.

INPUT-OUTPUT SECTION.
FILE-CONTROL.

        SELECT  INVOICE-FILE      ASSIGN TO "INVOICE".
        SELECT  PRINT-FILE       ASSIGN TO "PRINTFILE".
```

3.3 USE SOUND FILE, RECORD AND FIELD NAMING CONVENTIONS

THE NORM

There are benefits in developing data names that clearly describe their function. Time normally used to reference back and forth between the PROCEDURE DIVISION and the DATA DIVISION of the source program will be reduced.

SUGGESTED IMPROVEMENT

The following is an example of meaningful names used to define the filename and field names for a file:

```
FD INVOICE-FILE.
01 INVOICE-RECORD.
   05 INVOICE-NUMBER PIC 9(12).
   05 INVOICE-CUSTOMER-NUMBER PIC 9(10).
   05 INVOICE-DATE.
      10 INVOICE-DATE-MO PIC 9(02).
      10 INVOICE-DATE-DY PIC 9(02).
      10 INVOICE-DATE-YR PIC 9(02).
   05 INVOICE-DELIVERY-DATE.
      10 INVOICE-DEL-DATE-MO PIC 9(02).
      10 INVOICE-DEL-DATE-DY PIC 9(02).
      10 INVOICE-DEL-DATE-YR PIC 9(02).
   05 INVOICE-TOTAL-COST PIC 9(10)V9(02).
```

3.4 USE NUMERIC LITERALS

THE NORM

The tendency has been to code numerics in the body of the PROCEDURE DIVISION.

SUGGESTED IMPROVEMENT

Numeric literals defined in the WORKING-STORAGE section of the program will reduce the time required to make program modifications when this variable must be changed.

WORKING-STORAGE SECTION

```
01 MAXIMUM-INVOICE-DETAILS PIC 9(03) VALUE 150.
```

PROCEDURE DIVISION

```
PERFORM 5000-SEARCH-INVOICE-TABLE THRU
       5000-SEARCH-INVOICE-TABLE-EXIT UNTIL
       (INVOICE-TABLE-CTR = MAXIMUM-INVOICE-DETAILS)
```

3.5 ORGANIZE THE WORKING-STORAGE SECTION

THE NORM

There has been very little interest in organizing the contents of the WORKING-STORAGE section. Entries in this portion of the program often do not follow a logical format.

SUGGESTED IMPROVEMENT

It is recommended that the WORKING-STORAGE section of the program be organized in such a way that data elements can be easily found. This will also reduce the possibility of creating duplicate data names. The following example demonstrates the organization of data by type. Within each data type, the entries are listed in alphabetical order.

WORKING STORAGE SECTION

```
*****
***              H O L D   A R E A S              ***
*****

01  HOLD-STORE-DATA                                PIC X(75).

01  TODAYS-DATE                                    PIC 9(06).

01  TODAYS-DATE-BREAKDOWN      REDEFINES          TODAYS-DATE.
   05  TODAYS-MONTH                                                    PIC 9(02).
   05  TODAYS-DAY                                                       PIC 9(02).
   05  TODAYS-YEAR                                                       PIC 9(02).

*****
***              T A B L E S                      ***
*****

***
***  This table is used to build the two records which will
***  be stored for each invoice entered.
***

01  INVOICE-DATA-TABLE.
   05  INVOICE-DATA-TBL                                OCCURS 150 TIMES.
       10  INVOICE-NBR                                PIC X(12).
       10  INVOICE-CUSTOMER-NUMBER                    PIC X(10).
       10  INVOICE-DELIVERY-DATE                       PIC 9(06).

01  UNIT-OF-ISSUE-TABLE.
   05  UNIT-OF-ISSUE-TBL                                OCCURS 300 TIMES
       ASCENDING KEY IS UIT-LITERAL
       INDEXED BY UIT-INDEX.
       10  UIT-LITERAL                                PIC X(02).
```

```
*****
***                               S W I T C H E S                               ***
*****
```

```
01 ANSWER-SWITCH                      PIC 9  VALUE 0.
   88 ANSWERED-YES                     VALUE 1.
   88 ANSWERED-NO                       VALUE 2.
```

```
***
*** Entries exist in this switch in the order each is retrieved.
***
```

```
01 GET-FIELD-SWITCH                  PIC 99 VALUE 0.
   88 GET-INVOICE-NUMBER                VALUE 1.
   88 GET-DATE-SIGNED                   VALUE 2.
   88 GET-DELIVERY-DATE                 VALUE 3.
   88 GET-EXPIRATION-DATE               VALUE 4.
   88 NO-MORE-FIELDS                    VALUE 99.
```

```
*****
***                               C O U N T E R S                               ***
*****
```

```
01 COUNTERS.
   05 DESC-ENTRY-CTR                    PIC 999 USAGE COMP VALUE 0.
   05 INVOICE-DATA-TBL-CTR              PIC 999 USAGE COMP VALUE 0.
   05 INVOICE-DATA-TBL-SRCH-CTR         PIC 999 USAGE COMP VALUE 0.
   05 UNIT-OF-ISSUE-TBL-CTR             PIC 999 USAGE COMP VALUE 0.
```

3.6 USE THE OPTIONS AVAILABLE FOR TABLE SEARCHES

THE NORM

When tables have been used, searches have often been performed by using an iterative process. In this situation, the programmer will execute a paragraph where the programmer is responsible for controlling the value of the subscripted variable. The following gives an example of this tendency:

```
5410-FIND-UNIT-OF-ISSUE.

   IF ( HOLD-UNIT-OF-ISSUE = UNIT-OF-ISSUE (UIT-COUNTER) )
       MOVE HOLD-UNIT-OF-ISSUE TO INVOICE-UNIT-OF-ISSUE
       SET VALID-UNIT-OF-ISSUE TO TRUE
   ELSE
       ADD      1                TO UIT-COUNTER
   END-IF

.

5410-FIND-UNIT-OF-ISSUE-EXIT.
EXIT.
```

SUGGESTED IMPROVEMENT

Use of the PERFORM ... THRU ... VARYING construct:

PROCEDURE DIVISION

```
PERFORM 5410-FIND-UNIT-OF-ISSUE          THRU
       5410-FIND-UNIT-OF-ISSUE-EXIT      VARYING
       UIT-COUNTER FROM 1 BY 1           UNTIL
       (VALID-UNIT-OF-ISSUE              OR
        UIT-COUNTER = MAXIMUM-UIT-ENTRIES)
```

5410-FIND-UNIT-OF-ISSUE.

```
IF ( HOLD-UNIT-OF-ISSUE = UNIT-OF-ISSUE (UIT-COUNTER) )
  MOVE HOLD-UNIT-OF-ISSUE TO INVOICE-UNIT-OF-ISSUE
  SET VALID-UNIT-OF-ISSUE TO TRUE
END-IF
```

.

5410-FIND-UNIT-OF-ISSUE-EXIT.
EXIT.

Use of the SEARCH command:

WORKING-STORAGE SECTION

```
01 UNIT-OF-ISSUE-TABLE.
   05 UNIT-OF-ISSUE-TBL          OCCURS 300 TIMES
      ASCENDING KEY IS UIT-LITERAL
      INDEXED BY UIT-INDEX.
   10 UIT-LITERAL                PIC X(02).
```

PROCEDURE DIVISION

7200-SEARCH-ISSUE-TBL.

```
SEARCH ALL UNIT-OF-ISSUE-TBL OF UNIT-OF-ISSUE-TABLE
  AT END
    INITIALIZE                VALID-FIELD-SWITCH
  WHEN UIT-LITERAL (UIT-INDEX) = HOLD-UNIT-OF-ISSUE
    SET VALID-UNIT-OF-ISSUE TO TRUE
    MOVE HOLD-UNIT-OF-ISSUE TO INVOICE-UNIT-OF-ISSUE
  END-SEARCH
```

.

7200-SEARCH-ISSUE-TBL-EXIT.
EXIT.

3.7 USE THE EVALUATE VERB

NORM

When a number of values for a switch is possible, the programmer has generally used the NESTED IF construct. The complexity of this generally results in confusion as well as maintenance nightmares.

SUGGESTED IMPROVEMENT

The EVALUATE verb has the effect of reducing the complexity of the coding required. This is an improvement over the use of the NESTED IF construct. In effect, the coding is easier to read and more efficiently maintained.

WORKING-STORAGE SECTION

```
01 GET-FIELD-SWITCH          PIC 99 VALUE 0.
   88 GET-INVOICE-NUMBER      VALUE 1.
   88 GET-DATE-SIGNED         VALUE 2.
   88 GET-DELIVERY-DATE       VALUE 3.
   88 GET-EXPIRATION-DATE     VALUE 4.
   88 NO-MORE-FIELDS          VALUE 99.
```

PROCEDURE DIVISION

```
5000-GET-INVOICE-DATA.
    INITIALIZE  VALID-FIELD-SWITCH
    EVALUATE GET-FIELD-SWITCH
        WHEN 1  PERFORM 5010-GET-INVOICE-NUMBER      THRU
                  5010-GET-INVOICE-NUMBER-EXIT
        WHEN 2  PERFORM 5020-GET-DATE-SIGNED         THRU
                  5020-GET-DATE-SIGNED-EXIT
        WHEN 3  PERFORM 5030-GET-DELIVERY-DATE       THRU
                  5030-GET-DELIVERY-DATE-EXIT
        WHEN 4  PERFORM 5040-GET-EXPIRATION-DATE     THRU
                  5040-GET-EXPIRATION-DATE-EXIT
    END-EVALUATE
    .
5000-GET-INVOICE-DATA-EXIT.
EXIT.
```

3.8 IMPROVE PROGRAMMING FORMAT AND FORM

NORM

The tendency has been to produce executable code that is simply workable. Very little attention has been focused on the readability and maintainability of the code produced.

SUGGESTED IMPROVEMENT

This example demonstrates the use of indenting, spacing and inline commenting to improve clarity of the source code.

```
5100-GET-INVOICE-NBR.

      CALL "FDV$GET"      USING BY DESCRIPTOR D-RP0170-INV
                          BY REFERENCE TERMINATOR-SWITCH
                          BY DESCRIPTOR "INV"
                          :
                          :
                          :
IF RETURN-KEY
  SET ACTION-MESSAGE          TO TRUE

***
***       The following message is retrieved via the CALL
***       to the routine ERRO01:
***
***       "DO YOU WANT TO EXIT THIS SCREEN?      (Y/N)".
***

      MOVE 31              TO ERROR-SS
      CALL "ERRO01"        USING X-MESSAG,
                          ERROR-SS

      PERFORM 9100-DISPLAY-MESSAGE      THRU
              9100-DISPLAY-MESSAGE-EXIT

      IF ANSWERED-YES
        SET NO-MORE-FIELDS          TO TRUE
        SET NO-MORE-INVOICES       TO TRUE
      END-IF
END-IF

      IF TAB-KEY
        INITIALIZE                HOLD-INVOICE-NBR
        PERFORM 6000-VALIDATE-INVOICE-NBR THRU
              6000-VALIDATE-INVOICE-NBR-EXIT
        IF INVOICE-NBR-VALID
          SET GET-DATE-SIGNED      TO TRUE
        END-IF
      END-IF

5100-GET-INVOICE-NBR-EXIT.
EXIT.
```

4 RESULTS OF IMPROVING THE PROGRAMMING FUNCTION

This discussion has focused on the productivity of COBOL. At the outset, the coding function for new systems, using these and other techniques, may possibly take more time to produce. It will require that we do a bit more planning and research before we begin the actual coding function. In the long run, the cost savings will be substantial.

We can look at it in terms of an investment. By spending a bit more time and resources initially, we will be able to receive a dividend in reduced maintenance costs in future years. My primary concern is the reduction of costs, improved readability and more efficient program maintenance over the project life cycle. When these benefits are realized, COBOL will be more attractive as a development tool. This will have the effect of at least postponing the migration of data processing shops to Fourth Generation Languages.

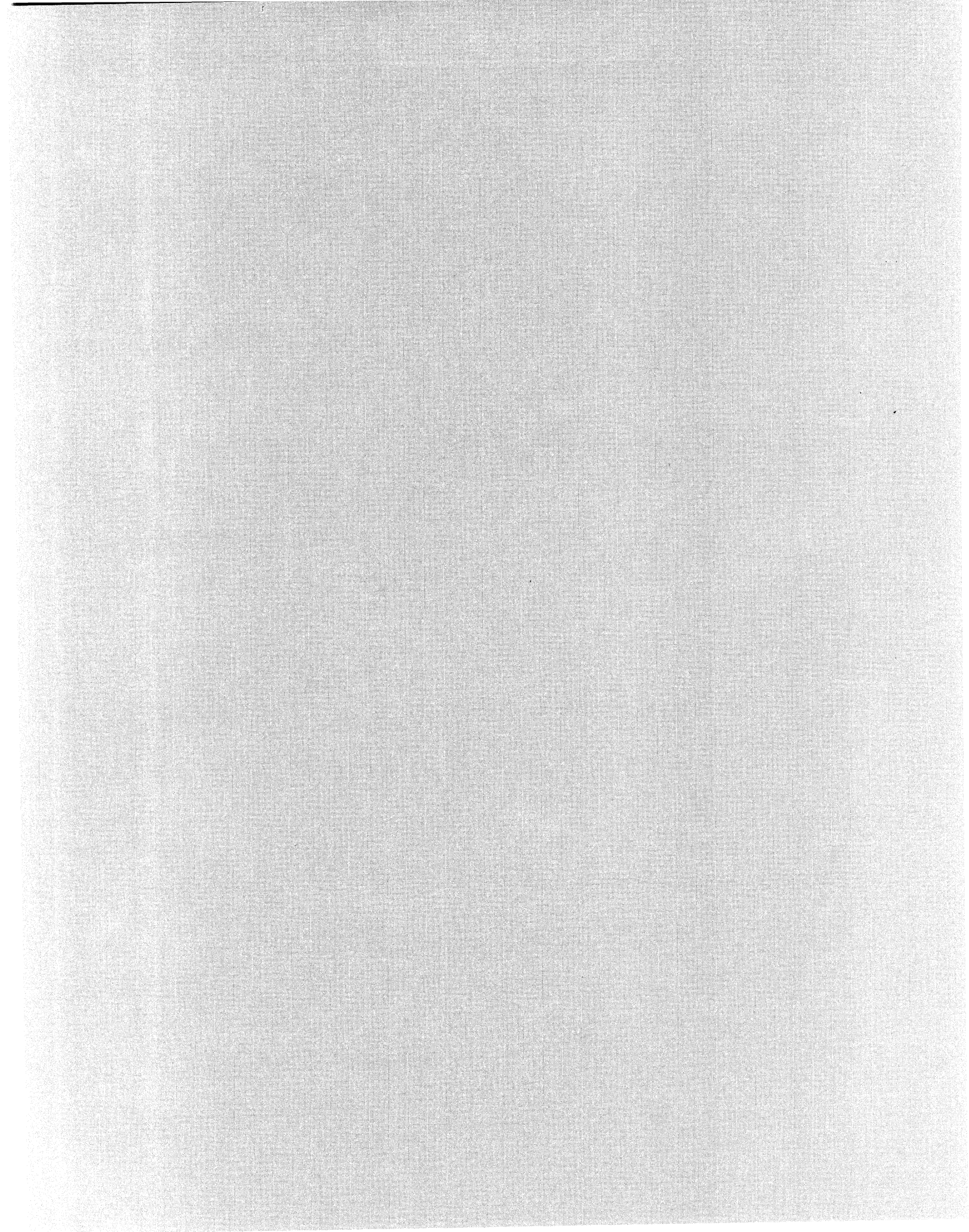
5 REFERENCES FOR FURTHER READING

Chmura, Louis J. and Ledgard, Henry F.; COBOL With Style: Programming Proverbs; Hayden Book Company, Inc.; Rochelle Park, New Jersey, 1976.

Kernighan, Brian W. and Plauger, P.J.; The Elements of Programming Style; McGraw-Hill Book Company; New York, New York, 1978.



**DATA ACQUISITION, ANALYSIS, RESEARCH,
AND CONTROL SIG**



O. Guetta, J. Hamilton, J.J. Conklin, A. Dubois.
Department of Physiology, Armed Forces Radiobiology Research Institute and
Digestive Diseases Division, Department of Medicine,
Uniformed Services University of the Health Sciences
Bethesda, Maryland

ABSTRACT

To improve our understanding of gastric motility, gastric electrical activity was amplified, recorded on an analog tape, and subsequently analyzed using an MNC/DECLAB-23 computer, locally developed programs, and software packages. After digitization, the output of a 2.2 to 5.5 cpm band-pass filter was analyzed using a peak-processing program that calculated the electrical activity and the instantaneous frequency of each peak. In addition, the original electrical signals were filtered using a 54 to 78 cpm band-pass filter and analyzed using a spike processing program to determine the instantaneous frequency and activity of the bursts. Electrical signals were characterized by slow fluctuations occurring at 3.7/min, with an activity of 9.1 mV/min, and by superimposed spikes at 3.3/min with a mean burst activity of 2.9 mV/min. This method provides an objective and precise measurement of gastric electrical activity, and may be applied to study diseases of the stomach, as well as the effect of various medications.

INTRODUCTION

The present paper describes a computerized method for the analysis of gastric electrical activity. These signals are first amplified, without any other processing than a 30 Hz low-pass filter, and then taped on an analog recorder. The taped signals are subsequently digitized and analyzed using an MNC/DECLAB-23 computer (Digital Equipment Corporation, Landover, MD), locally developed programs, and software packages. Using this method, we found that gastric electrical signals were composed of: (1) slow waves occurring at a frequency of about 3 cycles/min; and (2) fast activities, called spikes, in the range of 0.9 Hz to 1.3 Hz.

MATERIAL

The MNC/DECLAB-23 computer is based on the PDP-11/23 processor. RT-11 is the interactive, single-user, real-time operating system used for program development and real time application. The locally developed programs, as well as the programs from software packages, are written in FORTRAN IV. Our hardware includes:

1. The PDP-11/23 central processing units;
2. Two RL01 hard disks;
3. Two RX02 floppy disks;
4. One VT-125 graphic display terminal;

5. One LA-100 letterprinter;
6. A programmable real-time clock module that offers several accurate methods of measuring intervals and counting events and, therefore, controls analog to digital transfer rates;
7. A four-channel analog to digital converter module that has an input range of -5.12 V to +5.12 V and a scale of 4,096 points to represent these 10.24 V; the sensibility limit is, therefore, 2.5 mV. This A/D converter translates the amplitude of the analog voltage received from the tape recorder into a binary value that the processor can accept. It also features a direct connection to the clock module, so that input sampling during a transfer of data will occur at very precise clock-controlled intervals.

METHODS

The processing of the signals includes digitization, filtering, peak-processing analysis, and spikes analysis.

I. Digitization

Digitization is performed using the clock module (MNCKW) and the A/D converter module (MNCAD), in addition to the REAL-11/MNC library software. The REAL-11/MNC software consists of subprograms that allow communications with the

MNCKW and MNCAD modules and perform A/D conversions, buffer management and all data transfer operations.

The REAL-11/MNC software accesses the four channels of the A/D converter sequentially, but the period of time between accessing one channel and accessing the next one is so small (150 microseconds) that the transfer is considered simultaneous on the four channels (1). The smallest amount of data that can be transferred is one word. Data transferred simultaneously over one or more channels is called a sample. The process of transferring one or more samples is called a sweep. To successfully execute a sweep, the main program must:

- first initialize the sweep by defining its characteristics
- then start the sweep with the primary clock
- finally define the event that drives the sweep.

The repetitive occurrence of this event will cause each sample to be transferred.

Our sweeps are driven by hardware, with a clock overflow. A clock interval is defined by a combination of a selected internal oscillation rate and a presettable counter. Each time an oscillation time interval elapses, the clock counter is incremented until it reaches its greatest value. The next interval causes the counter to overflow and reset to the presettable value. This event is called a clock overflow. Each time the clock overflows, a sample is transferred through the A/D converter and is temporarily stored in a user-defined buffer. All REAL-11/MNC subprograms must use at least one buffer. The user-defined buffers that we define are two-dimensional integer arrays. The first dimension represents the number of channels used by the sweep and the second dimension is the number of samples transferred per channel. While a sweep is active, each REAL-11/MNC buffer called by that sweep is always in at least one of the following four states:

- in the device-transfer state, when the software is transferring data to the buffer
- in the user-processing state, when the user program is processing data in the buffer
- available for the device-transfer state and located in the device queue.
- available for the user-processing state and located in the user queue.

Furthermore, the sweep uses a forty-word sweep-information array that contains values that the REAL-11/MNC software uses to control and execute the sweep. The user-defined buffers must be managed very carefully so that the sweep executes successfully. The digitization program must perform certain procedures by calling sub-routines in a very specific order.

Our main program for digitization is thus organized as follows:

- We first define IBUF(40), sweep-information array, and I0(4,200), I1(4,200), I2(4,200) three user-defined buffers. An iteration on three buffers is required to simultaneously digitize at 8 Hz and store the results on a disk.
- We establish, assign identification num-

ber, and place these buffers in the user-processing state by calling SETIBF(IBUF,IND,,I0,I1,I2).

- We call RLSBUF(IBUF,IND,0,1,2) to remove buffers 0 (I0), 1 (I1) and 2 (I2) from the user-processing state and to place them in the device queue.
- We call XRATE(DWELL,IRATE,IPRSET) with DWELL desired intersample interval, IRATE computed clock rate, IPRSET clock preset value, and CLOCKS (IRATE,IPRSET,IND) with IND success or failure code, to set the rate of the clock and, therefore, the rate of digitization.
- A call to ADSWP(IBUF,800,NBUF,MODE,IPRSET,,,0,4) immediately initiates an A/D input sweep through the A/D converter. The analog input word placed in the buffer consists of twelve data bits read from the A/D converter.

MODE is the sum of the code for gain, data representation, sweep start and sampling. We choose MODE=16 to have a gain of 1, to start the sweep immediately, to perform a channel sweep on each clock overflow, and to return only the twelve bits of the A/D converter by forcing bits 15 to 12 to zero. NBUF is the number of buffers to be filled. NBUF=300 allows to digitize two hours of a signal at 8 Hz.

- After buffer 0 (I0) is filled, the REAL-11/MNC software removes it from the device-transfer state and places it in the user queue. It then removes from the device queue buffer 1 (I1), the next available buffer in that queue, and places it in the device-transfer state. Buffer 2 (I2) remains in the device queue.
- A call to IWTBUF(IBUF,,IBUFNO,IND) returns IBUFNO, number of the next buffer available in the user queue (0 (I0) in our example) and places it in the user-processing state. While the sweep continues transferring additional data from the A/D converter to the buffer in the device-transfer state (1 (I1)), the program reads the data from buffer 0 (I0) and writes them in a file on the hard disk.
- When all data have been written, a call to RLSBUF(IBUF,IND,IBUFNO) releases buffer number IBUFNO (I0) and places it in the device queue. The same process is then repeated until NBUF buffers are filled.

II. Filtering

To isolate each type of activity, we need to filter the raw signal obtained after digitization. Therefore, we have designed various digital filters. The general process of designing a digital filter involves several steps:

- choosing a specific structure in which the filter will be realized.
- Solving the approximation problem to determine filter coefficients that satisfy performance specifications.
- verifying by simulation that the resulting design meets the given performance specifications.

A program based on the Remez Exchange Algo-

rithm is used to design specific Finite Impulse Response (FIR) passband/stopband filters (2). Each digital band-pass filter is characterized by:

1. the filter duration, also called length of the filter, which is an odd integer to maintain linear phase.
2. fp1, fp2, fs1, fs2, passband and stopband cut-off frequencies. By convention, fp and fs are expressed in units of normalized frequencies, which are the actual signal frequencies divided by the sampling rate. Because of the Nyquist sampling theorem, the normalized frequency axis extends from 0.0 to 0.5.
3. the desired frequency response, 1 in the passband area and 0 in the stopband area.
4. The weighting function:
 - 1 in the stopband area
 - $k=d2/d1$ in the passband area with $20\log(1+d1)$ representing the passband ripple or deviation in dB, and $20\log(d2)$ being the stopband ripple or attenuation in dB.

We always choose $d1=d2=1$.

A 40 dB attenuation means that the filter suppresses 99% of the original amplitude, and leads to a 2.5 dB deviation.

If the amplitude of the original signal is AMO and the attenuation in the stopband is ATT, the amplitude of the filtered signal AMF in the stopband will be:

$AMF = AMO/F$ with F verifying $ATT = 20\log(F)$. In the same manner, if the amplitude of the original signal is AMO and the deviation in the passband is DEV, the amplitude of the filtered signal AMF in the passband will be:
 $AMF=AMO/F$ with F verifying
 $DEV=20\log(F)$.

Several tradeoffs exist among these design parameters. Obtaining low values of $d1$ and $d2$ requires increasing the length of the filter. A length of 127 is generally required to approximate sharp cutoff filters, although a large value of N increases the number of additions and multiplications that the computer has to do and therefore slows down the process.

The output of each digital filter is calculated using the equation below. If N represents the length of the filter, H the array containing the coefficients of this filter, and I the array containing the input values, then each element of the output array O is defined by:

$$O(z) = \sum_{k=1}^N I(k+z-N)*H(N-k+1)$$

or because of the symmetry of the filter

$$O(z) = \sum_{k=1}^N I(k+z-N)*H(k)$$

Two different band-pass filters are used to isolate the slow activity and the fast activity; a separate processing is done on the output of these two filters.

III. Peak-processing analysis

A peak-processing analysis is done to analyze the slow waves in the signal.

Peak-processing analysis consists in detecting significant fluctuations, called peaks, in data describing a waveform, and reporting definitive characteristics for each peak found. This type of analysis is done by using a peak-processing subroutine from the Digital Laboratory Subroutines Package (3). The peak-processing algorithm is a procedure to detect increasing and decreasing trends in the set of data. Output from this PEAK subroutine is directly related to the points where changes in these trends are observed. When an increasing trend is seen, the point where the increase begins is labeled the start of the peak, and its value the leading minimum height. The point where a subsequent decreasing trend begins is the crest of the peak, and its value is the crest height. The point where the decreasing trend stops is taken as the end of the peak, and its value called trailing minimum height. Input to this subroutine must be a series of discrete positive integers corresponding to values of the signal at evenly spaced intervals. The peak-processing algorithm first takes a linear average of input-data points. The number of points to be averaged, called Original Point Density (OPD), has to be specified by the user. The averaged data points can then be filtered with an internal digital filter. Since our data have already been filtered, we do not use the optional filter, and choose OPD=1 to suppress the averaging. In our case, a greater value for OPD has for effect to smooth the data and prevent the detection of important peaks. Even though the original signal has been smoothed by the digital filter, the resultant data points may still exhibit slight point-to-point fluctuations unrelated to the dominant trend of the data. Three parameters, the baseline test factor, the gate factor, and the minimum increase, can be set so that the algorithm eliminates much of the effect of these fluctuations. The subroutine is called as follows:
 CALL PEAK(ITABLE,INPUT,INLAST,INPTR,OUTPUT, IDIMO,NPEAKS)

- (a) ITABLE is a 79-element integer array used to store intermediate results and other information required by the algorithm. The first five elements are the algorithm variable parameters:
- ITABLE(1), Original Point Density, is chosen to equal 1.
 - ITABLE(2) is the baseline test factor; on a peak with a width of WD, baseline detection begins at time WD x ITABLE(2) past crest time. We choose ITABLE(2)=1. A greater value would delay baseline detection, so every peak detected would be wider, but some peaks would be missed.
 - ITABLE(3), gate factor, specifies a valid directional trend in terms of the number of changes in direction over a series of filtered points. We choose ITABLE(3)=3. A smaller value would allow the detection of

very small amplitude peaks that we consider as insignificant fluctuation. Thus, the algorithm would detect fewer peaks if ITABLE(3) was further increased.

- ITABLE(4), Minimum Differential between data points that would be interpreted as a real increase by the algorithm, has similar effects than ITABLE(3). After trying several values in the range from 1-7, we chose ITABLE(4) = 1.
- ITABLE(5) is fixed at 1, so that the output is a single precision floating point. Double-precision would be obtained with ITABLE(5) = -1.
- (b) INPUT is a 1,000-element array containing input data.
- (c) INLAST specifies subscript of last data element in INPUT; it is set at 1,000 at the beginning of the program.
- (d) INPTR specifies subscript of last element in INPUT processed; it is set at 0 at the beginning of the program.
- (e) OUTPUT(10,100) is the output array, and in particular:
 - OUTPUT(2,N) crest height, Nth peak
 - OUTPUT(3,N) crest time, Nth peak
 - OUTPUT(4,N) leading minimum height, Nth peak
 - OUTPUT(5,N) leading minimum time, Nth peak
 - OUTPUT(7,N) trailing minimum height, Nth peak
 - OUTPUT(8,N) trailing minimum time, Nth peak
- (f) IDIMO=100 specifies the number of peak data sets that can be stored in OUTPUT.
- (g) NPEAKS is the number of peak data sets already stored in OUTPUT.

We split the filtered file into continuous sets of 1,000 elements each. Every set becomes successively the input of the PEAK program, and the characteristics of all the peaks found are stored in a first file (F1). We noticed that if a significant peak was located at the end of a set and the point where the decreasing trend stopped was located at the beginning of the following set, no end of peak was detected by the subroutine and, therefore, the peak was not detected. For identical reasons, a significant peak located at the beginning of one set could sometimes remain undetected. To avoid missing these peaks, an overlapping of 500 elements is done, and the peaks found are stored in a second file (F2). A third file (F3) is then created by merging F1 and F2 and deleting one of each peak common to both F1 and F2.

For each remaining peak, the following parameters are then calculated and stored:

- (a) width of the peak
 $W(N) = \text{OUTPUT}(8,N) - \text{OUTPUT}(5,N)$
- (b) relative amplitude of the peak
 $A(N) = \frac{\text{OUTPUT}(2,N) - (\text{OUTPUT}(4,N) + \text{OUTPUT}(7,8))/2}{2}$
- (c) an approximation to the area under the peak, called triangulation and being the area of the triangle: beginning of peak / peak / end of peak.

- (d) the instantaneous frequency of each peak, which is the inverse of the distance expressed in seconds between one peak and the previous one.

$$IF(N) = 1 / (\text{OUTPUT}(3,N) - \text{OUTPUT}(3,N-1))$$

We then average and calculate the standard deviation of these parameters for different windows of the filtered signal.

IV. Spikes analysis

Spikes analysis involves eliminating the slow fluctuations of a signal, detecting only its fast variations called spikes, and quantifying the activity of each burst of spikes. A data point is considered to be a spike if its amplitude is greater or smaller than the threshold defined as the mean of the filtered signal's values plus or minus 1.4 times standard deviation. This value was selected because lower values of the threshold prevented the separation of spikes from noise. Two consecutive spikes are part of the same burst if the distance separating them is less than two seconds. For each burst of spikes, we calculate and store its duration, its activity, and its instantaneous frequency. If:

ibeg(n)	time beginning of burst n
iend(n)	time end of burst n
A(i)	amplitude of point time i
RMEAN	mean of the signal's values
DUR(n)	duration of burst n
ABS(Y)	absolute value of Y
IF(n)	instantaneous frequency of burst n

then:

$$DUR(n) = iend(n) - ibeg(n)$$

$$i = iend(n)$$

$$ACT(n) = \sum_{i=ibeg(n)}^{i=iend(n)} ABS(A(i) - RMEAN)$$

$$IF(n) = 1 / (ibeg(n) - ibeg(n-1))$$

We then average and calculate the standard deviation of these parameters for different windows of the filtered signal.

RESULTS

Figure 1 illustrates an example of our results. Figure 1A represents a raw electrical signal, while Figures 1B and 1C represent the processed signals. The position of each peak is marked with a "+" on the graph. The beginning of each burst of spikes is marked with a "+", its end is marked with an "x".

CONCLUSION

Peak-processing and spike analysis of short tracings seemed at first easy to perform by "eye-balling" the tracings. However, the computer was needed to achieve objective and quantitative analysis of large amounts of data. By combining locally-developed programs, software packages and by trial and error, this method provided a precise and objective measurement of the two types of myoelectrical activity, thus allowing comprehensive analysis of the electrical activity of the stomach.

REFERENCES

1. Real-11/MNC Fortran Programmer's Reference Manual. Digital Equipment Corporation, Landover, MD, 1982.
2. Theory and Application of Digital Signal Processing. Lawrence R. Rabiner and Bernard Gold. Program written by Jim McClellan, Rice University, 1973:194-204.
3. Laboratory Subroutines Programmer's Reference Manual, Chapter 2, Digital Equipment Corporation, Landover, MD, 1982.

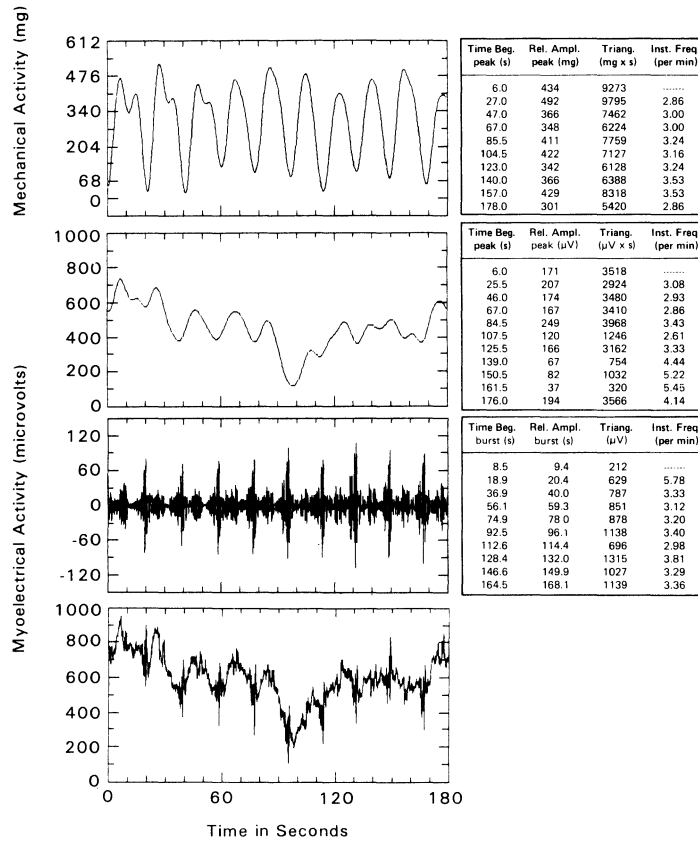


Figure 1: Example of mechanical (A) and myoelectrical (B,C,D,) gastric activities. On the left are the digitized signals: A and B are the output of the low-pass filter processing using the peak program, C is the output of the band-pass filter processed using the spikes program. The tables on the right of the tracing A, B, and C represent the parameters calculated for each event.

BASEWAY IMPLEMENTATION ISSUES

Daniel J. Drislane
Digital Equipment Corporation
Rochester, New York

ABSTRACT

Baseway is a software product set from Digital Equipment Corporation that allows manufacturers maximum flexibility and expandability in connecting industrial controllers to manufacturing applications. Application programs and plant personnel are then allowed easy access to data originating on the factory floor through industrial devices such as programmable controllers, robots and numerically controlled machine tools. Baseway software also provides a well defined information management architecture whereby many applications may share data and send messages to one another as well as to devices on the factory floor. Implementing Baseway in a factory environment represents a significant step towards Computer Integrated Manufacturing. Consequently, there are many issues that must be addressed to ensure successful implementation. The diverse family of devices that can be computer integrated, potential manufacturing applications to be used and performance considerations are discussed.

INTRODUCTION

The Baseway software product set can be considered an architecture that serves two important functions: (1) data acquisition and control of manufacturing processes and (2) implementation of manufacturing software applications. The most notable feature of the architecture is that it allows the above two functions to work together by enabling powerful high-level applications to collect data from a manufacturing process, make some decisions and/or interact with a user, and then perhaps effect control of that same (or another) manufacturing process.

The product set (there are two) addresses the needs of the manufacturer who wishes to implement a hierarchical manufacturing data acquisition and control system. The environment that Baseway addresses is usually a collection of various industrial devices such as programmable controllers, robots, numerically controlled machine tools and barcode wands. These devices are connected to a computer that "commands" the industrial devices and perhaps the same or a separate computer provides a user interface and runs application program tools that collect and use factory floor data. All of this operating in a distributed environment (factories lend to the distributed processing theory) and classically linked

into the plant MIS environment completes the scenario. Distributing data to these disparate devices, or collecting data from them to send to powerful applications, is a monumental problem for manufacturers. Implementing Baseway is a move in the right direction.

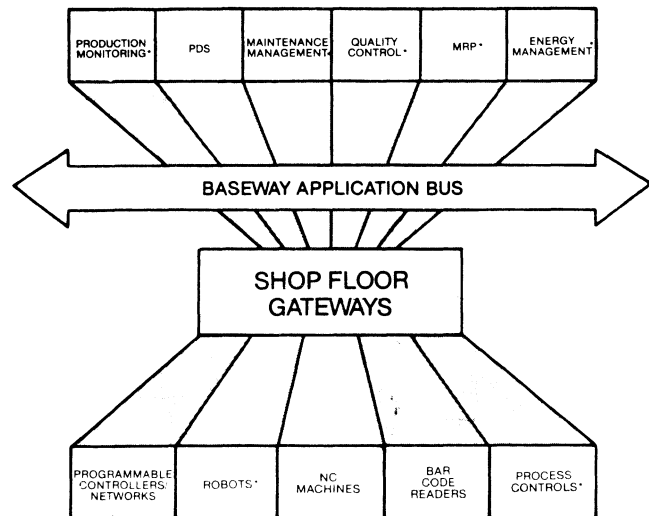


FIGURE 1, Baseway Software Architecture

The Baseway architecture, illustrated in Figure 1, makes the sets of rules in implementing a manufacturing software system known to you. However the task of implementation within such harsh, unfriendly environments demands a carefully thought-out plan. Understanding the function of the Baseway product family is necessary before such a plan is drafted.

What Is Baseway?

The Baseway software product family consists of two components [1,2]: (1) the Baseway Applications Bus is an applications processing and data management program that runs in a VAX/VMS standalone, cluster or DECnet environment; (2) the Shop Floor Gateway is a real-time data processor that acts as an industrial controller interface and is able to send and receive data to and

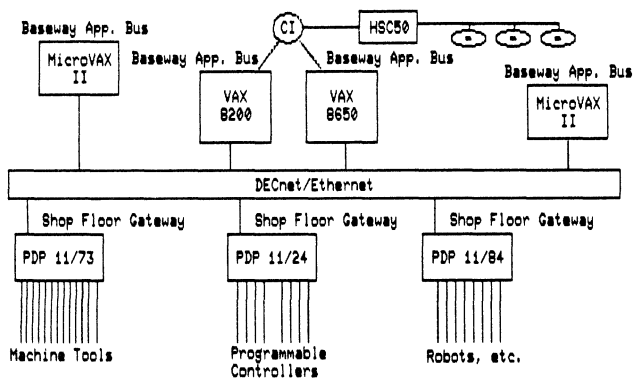


FIGURE 2, Sample Hardware Configuration

from Baseway applications as well as execute polling operations. The Shop Floor Gateway executes on any 22-bit PDP-11 and runs in an RSX-11S memory-only environment. Therefore, the Shop Floor Gateway software is built on the VAX (using the VAX/RSX Application Migration Executive) and downloaded via DECnet to the appropriate PDP-11 [2]. A typical hardware Baseway configuration is illustrated in Figure 2.

The Baseway Applications Bus software also provides a number of menu-driven, ready-to-use facilities that provide useful functions such as:

- o Programmable Device Support - a utility that allows you to upload and download programmable controller logic programs, store those programs in disk-based libraries on the VAX and even perform comparisons of running logic programs with its stored copy. There is also

logic program documentation/annotation capabilities.

- o Application Control - allows you to start applications in the Baseway environment, and more importantly, provides for an orderly shutdown of the application if it encounters run-time problems.
- o Definition Editors - allow you to define users, their operating privileges, device registers (or memory locations), Shop Floor Gateways (there may be more than one), ports or lines and polling sets.

So what are the rules of machine integration; of implementing applications; of data management? And specifically, what must you consider when installing Baseway? To start, consider the environment.

YOUR SHOP FLOOR

Manufacturers who have made steps towards automation are committed to employ advanced control devices on their factory floors. Programmable controllers are in their second decade of use and have advanced considerably in processing capabilities, speed and I/O capacity. Computer Numerical Control has risen to new heights in versatility and operation of machine tools. Robots are displaying increasingly wider applicability with enhanced capabilities in vision, programmability, communications, repeatability and payload. These industrial wonders populated about the shop floor are sinks for critical, invaluable information. They hold in their memories process data, tolerance measurement results, product tracking information. Information that often times is difficult -- even impossible -- to obtain in any meaningful way. Opposite to obtaining this desired information is the sending of equally important information. Downline-loading of logic programs to PLCs, part programs to NC machine tools, path programs to robots, and routing information to conveyor systems are key components of automated manufacturing. Employing the features of Baseway software will facilitate these operations.

PROVIDING SUPPORT FOR DEVICES

A design feature of the Baseway Applications Bus ("Applications Bus") and the Shop Floor Gateway ("Gateway") is that they provide the manufacturer with the "hooks" for connecting new devices or providing total communications support for a device. "Providing support" means developing the total communications support for an industrial device. Communications support in Baseway requires that a device is integrated at the following levels:

- o physical communication - providing a line or device driver to execute in the Shop Floor Gateway.
- o functional I/O support - defining the necessary device I/O function codes to operate the device. This is the responsibility of the Shop Floor Gateway's Device Interface Module (DIM), which will be explained later.
- o generic I/O support - providing the application programmer with a common device access interface at the Applications Bus level [1].

The Gateway's primary component for device integration is the Device Interface Module, or DIM [1]. DIMS define the functional characteristics of each device to be integrated. The Bus components consist of the Device Definition file and high level device access (i.e. read/write) and control (i.e. start/stop) programs that applications will use when communicating to each device. The key device support components are illustrated in Figure 3.

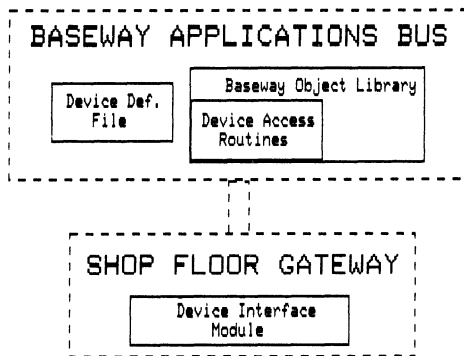


FIGURE 3, Primary System Device Support Components

The Baseway Applications Bus V1.1 and Shop Floor Gateway V1.1 currently supports Allen-Bradley Data Highway (tm) and programmable controllers (PLC-2, PLC-3).

Gould Modbus (tm) and programmable controllers (384, 484, 584) are also supported [6]. Texas Instruments Corporation provides separate Baseway/Shop Floor Gateway support for their PM programmable controller line and TIway (tm) network. These controllers are supported utilizing DIMs, device access programs and definition files.

Providing device support requires that you know the device fairly well. Essentially, two things must be considered when

integrating a device: communication and functionality. Ask two questions when considering supporting a device in Baseway.

- o How does it communicate? The specific communications interface and how it will be connected to a computer host (point-to-point, multi-drop, etc.) must be defined. Many devices come configured with various communications options supplied by the device vendor. RS232C is a popular standard for remote host communication, but don't assume that this is the supplied interface. The device could support RS422 as well, or perhaps a parallel interface. Protocols are to be considered as well. Protocols vary across the wide spectrum of industrial controllers for different reasons. In some cases, industrial controller makers might offer network hardware and software enabling their controllers to communicate with one another and possibly a host. Allen-Bradley's Data Highway (tm) is one such example that connects their PLCs (slaves) in a multi-dropped configuration to a PDP-11 host (master). Determine the most desirable method of communication configuration (if you have a choice) of the devices you wish to integrate into Baseway.

- o How functional is it? Another way of asking this question is, "How capable is the device of providing a true remote interface?" For example, devices that allow 15 local operations to be performed at the local console but allow only 5 to be performed remotely could be considered minimally functional from a remote source. It is possible that the desirable control features that you expect can be influenced from a Baseway environment are not possible.

Consider as an example the following: you plan to have a custom Baseway application "talk" with a family of robots. The application's purpose is to upload and download path programs to various robots at certain times or when certain events occur. A desirable feature is that you want the application program to query a particular robot for a directory of resident path programs. A problem arises because the robot's communications interface is incapable of reporting such information.

Reasons for this limit in remote functionality are many. An understandable one is that of safety: makers of programmable machinery don't want uncontrollable sources to turn their equipment on at the wrong time resulting in human injury. Whatever the reason, you must examine the functionality of each device and determine if it is capable of doing what

you want it to do. Baseway can only collect information that is capable of being reported.

Once you know how functional a device is, you can then design the device's Device Interface Module, develop any necessary Applications Bus device specific routines and update the device definition files to include the new device.

As previously stated, the Gateway supports two programmable devices as delivered by Digital: (1) Allen-Bradley PLCs (the PLC-2, PLC-3); (2) Gould PLCs (384,484,584); it also supports most ASCII and barcode devices. It is capable of supporting much more, including robots, additional PLCs, NC machine tools and process controls. The Gateway currently supports the DZ11 communications interface for Unibus PDP-11 systems and the DZQ11 for Q-bus PDPs. By no means are you limited to these two interfaces. For example, in supporting a device you may wish to collect data at a rate faster than that of the DZ11 (9600 bits/sec). Selecting a KMS11 for synchronous transfer of data at 56,000 bits per second might be more suitable in this instance. You do not have to limit your throughput capabilities to 9600 baud. Be aware, though, that the interface you choose is compatible with the device. Looks can be deceiving.

THE APPLICATIONS YOU'LL USE

Programmable Device Support, which is part of the standard Baseway shipped from Digital, is one example of an application that takes advantage of the Applications Bus. It utilizes various Bus facilities to read and write data to programmable controllers, to upload and download logic programs, and to compare logic programs resident in a PLC with those resident in libraries on the VAX.

Programmable Device Support is not the only application that may run in a Baseway environment. Custom applications may be written to take advantage of the data acquisition and control capabilities of the Applications Bus and the Gateway.

What Is An Application?

In a Baseway environment, an application can be considered as any set of programs and VAX/VMS processes running in a VAX/VMS environment that are (1) known to Baseway (via the Baseway application editor) and (2) utilize Baseway facilities. Putting it simply, programs that utilize some of the more than forty Baseway programming subroutines are Baseway applications. Applications might also share data gathered from shop floor devices. Most applications utilized in a Baseway environment will be

custom applications, examples of which follow:

- o QUALITY CONTROL MANAGER - monitors a automated mixing process by polling PLCs based on a predefined timing formula to read critical process values such as pressure, composition, temperature or measurements such as critical tolerance. This polled data is collected from a programmable controller's registers and either sent to a file for subsequent use by an analysis program or analyzed immediately. The application determines if the process was within specification. The application then orders the mixing process either to continue to abort.
- o ALARM MONITOR - monitors various points in a group of machines for correct operation. Each machine sets a status bit when abnormal operation occurs. The Alarm Monitor polls these status bits continuously to detect change, and once change occurs, begins a notification process. Notification is enabled via audible alarms on video terminals along with status information displayed in the shop maintenance department.
- o FLEXIBLE PATH MANAGER - part of a Flexible Manufacturing System (FMS), this application stores and manages robot path programs for all robots plant-wide. The FMS requires different robot paths programs to be downloaded to different robots at specified times or even when certain events occur. The application uses the Applications Bus to download the appropriate path to the appropriate robot.

Applications can be as simple as collecting data and populating a sequential file. But they may be complex applications that collect data, analyze it, converse with users and other applications or systems, and access large databases on remote systems. Baseway applications may run across a VAXcluster environment or even a DECnet network. In this case, the Applications Bus software would be installed on each CPU running a Baseway application.

In some cases, you might already use a manufacturing-related software package. Possibly you are presently evaluating software offered by third party software developers. Packaged software such as Material Requirements Planning (MRP), Inventory Control, Work In Process (WIP) Tracking are popular and useful tools in manufacturing management. There are many third party software vendors that sell VAX/VMS-based manufacturing software. Is it possible to integrate these packages with Baseway? It depends primarily on two factors: (1) Is source code available from the vendor and/or, (2) does the vendor offer a customizable facility or application

calling feature? There must be a means for the vendor's software to call the appropriate Baseway subroutines.

To examine this notion more closely, as an example consider using a third party statistical analysis program. You'd like to use the Applications Bus to access information required by the program. A key function that would be necessary is reading/writing to the database. Once you've collected the data from, say, your PLCs, you'll want to write this into the statistical program's database. Therefore, "middle-man" programs must be developed that will use the Baseway read subroutines to access the data and then use the statistical program's database write subroutines to enter the data into the database. This "middle-man" program is actually a Baseway application utilizing Baseway access subroutines to perform the widely varied I/O to the PLCs.

The Applications Bus library of over forty subroutines supports the VAX/VMS calling standard. The Applications Bus supports VAX languages: PASCAL, FORTRAN, COBOL, PL/I, C, Basic and Bliss-32.

How Does A Baseway Application Work?

A custom Baseway application can operate in many different ways according to the specific needs of the plant. Most Baseway applications, though, have a few similar characteristics. Almost all will be performing shop floor device access and an equal number will use the Baseway messaging and polling facilities.

Polling is accomplished by defining a "polling set" to Baseway. The polling set consists of information such as which device to poll, what memory locations, the polling interval and the destination Baseway message port that polled data will be sent to. Though polling operations with Baseway can be quite sophisticated, polling is simply a read operation that is carried out by the Shop Floor Gateway to determine if a specific memory location has changed in value.

The polled message port is usually controlled by a user written program called a Data Processor. The Data Processor is a data sink that is actually part of an application. Its purpose is to wait for polled data and when it arrives, to dispatch it to other application processes that will analyze it or display it.

It is here that the Baseway messaging facility earns its keep. The Data Processor may send and receive application-specific messages and data using Baseway subroutines. An advantage to Baseway messaging is that messages are routed to logical message ports, possibly in a VAXcluster or DECnet

environment.

Other application processes may wish to do deliberate device access such as stopping a robot, downloading a programmable controller or reading a status register in a machine tool.

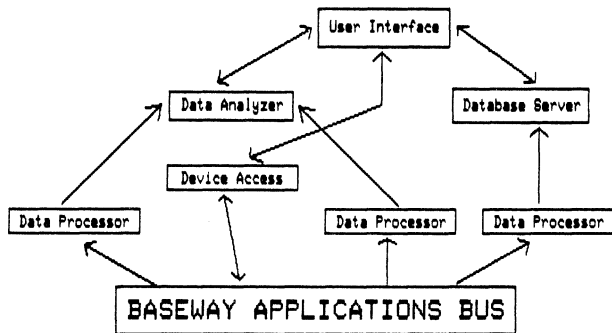


FIGURE 4, Sample Application

Baseway applications may perform a variety of functions within a flexible VMS environment. An example of a Baseway application represented in functional blocks is shown in Figure 4. The Data Processors are waiting for polled data to arrive in their message ports as defined by various polling sets. Each Processor sends its data to other areas of the application for further (perhaps more time consuming) processing. The Data Analyzer interprets the meaning of each message and interacts with a database server and a user interface process that drives terminal screens and manages a user session.

As you can see, an application doesn't have to exist as one VAX/VMS process, nor do the functions illustrated in Figure 4 have to execute on the same VAX. Baseway accounts for all application processes with its Application Control facility. Each part of an application may communicate with other parts via the Baseway messaging facility. Additionally, separate applications may communicate with each other. Figure 5 illustrates various communication scenarios.

A Baseway application running on Node 1 can communicate quite readily using Applications Bus messaging with a sister application running on Node 2 (see I). Similarly, processes belonging to the same application may communicate to each other across a DECnet network or VAXcluster (see II). As shown in Figure 4, Figure 5-III shows processes executing on the same node communicating with each other. Baseway provides the flexibility and frees you from managing many VMS mailboxes. Separate applications are often desirable for variable Baseway application control.

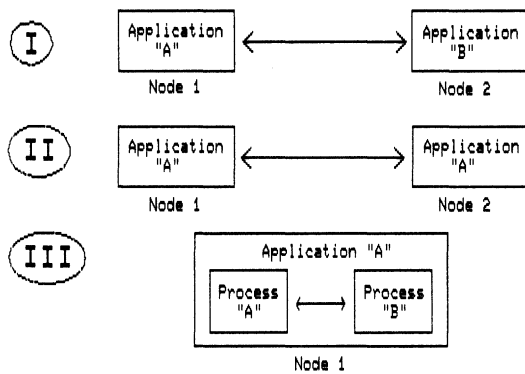


FIGURE 5, Baseway-supported Communication

What Programming Skills Do You Need?

Designing and developing Baseway applications are probably no more difficult than designing any other VAX/VMS application program of medium complexity. Good programming practice works miracles. Besides being familiar with Baseway concepts and operation, a working knowledge of VAX/VMS programming features is a must. Utilizing VMS Run Time Library routines, System Services, RMS, AST operation and the Lock Manager serve to enhance the power and functionality of any application, let alone one running in Baseway. Knowledge of one of the Baseway supported VAX-11 languages (mentioned earlier) is necessary [3,5].

Skills and knowledge that would prove to be a bonus are:

- o VAX/VMS Global Sections
- o VAXclusters
- o DECnet Concepts
- o Real-Time Programming Experience
- o VAX/VMS Products: FMS, ACMS, Rdb

The skills necessary are not extraordinary for the application programmer. A key skill that has not been mentioned is knowledge of the actual shop floor device. That's the advantage of Baseway! You, as the application programmer, are freed of that burden. Once support is provided for a device (or if it's already a Digital supported device), the application programmer need not worry about device attributes, protocol peculiarities, native operation, etc. Baseway provides high-level subroutines to access these devices.

How Valuable Is Your Data?

Data that originates on the shop floor as a result of a process or event can be insignificant or very important. It's up to the manufacturer -- specifically, the production engineers, quality engineers and plant foremen -- to determine the importance of data. In a computer-integrated manufacturing environment, this issue is addressed at the manufacturing applications level. Ask yourself the following:

- o what data must be collected?
- o when does it need to be collected?
- o where does it originate?
- o when is it created?
- o how is it to be used?
- o who is it to be used by?
- o how long is it to be kept?

This will help you establish how valuable your data is. The Applications Bus allows data collected from a single source to be sent to more than one calling application. Data Processors, previously described, can exist to receive data from polled devices via the Gateway and then dispatch that data to one or more applications.

Alternately, in the case of utilizing a database (Figure 4), Data Processors might be responsible for write accessing the database. A separate program, utilizing the Applications Bus messaging facility and the VAX/VMS Lock Manager, can notify various applications that the data has been placed in the database and is available.

Helpful Hints

Like most software systems, there are tried and true practices in Baseway. A few "steps in the right direction" follow:

- o Data Processors should be separate detached VMS processes that take advantage of: large working sets, process priority and resource locking.
- o Use the handle argument in the device access Baseway subroutines to avoid additional disk I/O when accessing devices repeatedly. Use of the defined handle will cause Baseway to reference a configuration global section instead of a disk file.
- o The Baseway messaging facility supports both temporary and permanent message ports. Use the permanent ports for unplanned, critical events and priority data. Use temporary ports for processes that hibernate or have secondary processing chores.

- o Use the Baseway allocate/deallocate routines when uploading and downloading programs to shop floor devices. This will lock the device until it has the proper program to run.
- o Take advantage of the Baseway logging facility to record significant events.
- o When creating Baseway message ports, use the message size and queue size wisely to keep VMS process BYTLM economical.

PERFORMING WITH BASEWAY

A successful implementation of Baseway requires that you examine the environment it will operate in and design the system accordingly. An often abused term used to describe the success of a computer system, no matter the size, is performance. Definition of performance is ambiguous in a computer environment. To folks with a computer engineering bent, things such as instruction time, I/O latency, and seek time are parametric to discussing performance. But to most of us, performance is a synonym for Response Time, the naked time between the release of the RETURN key and the display of the requested data on the screen.

This is perhaps a too simple definition but one that is central to a successful Baseway implementation. This paper does not intend to present a tried-and-true formula for performance in utilizing Baseway, but to raise important issues that one should consider when implementing Baseway.

Questions To Ask Yourself

There are performance considerations for both the overall VAX/VMS based load, and the load placed on the system by the Applications Bus and any Baseway applications. We also should consider the PDP-11s, acting as Gateways. Ask yourself:

- o how many programmable devices will be accessed (integrated)?

Knowing what programmable devices will be accessed and how many of them is helpful in sizing and configuring the Shop Floor Gateway PDP-11. Two important sizing parameters of concern are memory and the number of I/O interfaces. This has to do with Gateway saturation, which is further discussed in the next criterion.

- o for each device, what is the "communications profile?"

The communications profile of a programmable device can be defined as follows:

- * how many points (registers, memory locations, status bits, etc.) will be defined?
- * how often will these points be polled, read or written to?
- * are file transfers (uploads, downloads of programs, data, etc.) done often, and how big are the transfers?
- * what speed is the communications link between the device and the Shop Floor Gateway?
- * is it part of a industrial local area network (data highway, etc.) or a standalone device?

Of course, it's impossible to answer all of these questions. But giving thought to them will help you size your Gateway systems better. Perhaps the most critical factors are the first two, collectively impacting the Gateway processing saturation point. Generally, a Gateway responsible for polling 2000 points distributed among 125 PLCs every 5 seconds will be much "busier" than one who must download 1000 foot NC part programs to 5 machine tools every half hour. Study the role of each device that will be integrated, what environment it exists in, and what kind of information it will hold. These are the factors that will deem it either a busy device or a loner.

- o across the entire device population, how many known points will there be?

Consider how many known points (status registers, words, coils, trigger bits, etc.) will be defined to the Baseway system. The sum of these points represents the potential polled data load across the Baseway system. Of course, polling is only one operation that might be occurring at any one time. Data reads and writes (i.e. uploading and downloading) make up the balance of the I/O tasks performed by the Applications Bus and the Gateway.

- o what other software products will be used?

This question is posed as a reminder to those installing Baseway on a presently utilized VAX. This issue is more global in scope, addressing general performance of VAX systems in various environments. Various standard Digital software products complement the Applications Bus but also pose an additional processing load. This should be taken into consideration when designing for performance.

- o for each application, what is its "resource profile?"

An extension of the previous question, Baseway applications generally are free to allocate the same resources as most other applications that run in a VAX/VMS environment. The Applications Bus utilizes VAX/VMS, DECnet-VAX and VAX-11 FMS. It is up to the application developer, when designing a Baseway application, what resources, what services and what utilities to utilize. If the application is a third party package, it is important to obtain from the vendor the suggested minimum software/hardware resources that are used.

- o will plant floor design hurt or help me?

Consider the logistical implications of connecting the population of programmable devices to the Shop Floor Gateway PDPs. Shop floor devices and factory floor design have a way of making wiring tasks difficult. Be aware that length/speed requirements will impact the performance and noise rejection qualities of the communication links to the shop equipment.

Performance Hints

Baseway can consume resources if not properly configured and managed. The Baseway/Shop Floor Gateway environment in general is impacted by:

- o CPU power and memory utilization.
- o Baseway message port activity.
- o DECnet logical links.
- o amount of polling performed.
- o Baseway application resource load.
- o Speed and configuration of programmable device industrial local area networks that are attached to Gateways.
- o Gateway polling interval and rate of change.

Here are some helpful hints for the Applications Bus and Baseway applications:

- o critical inter-application and intra-application messaging (high volume, real-time) should reside on the same VAX.
- o in a DECnet Baseway environment, applications performing device access operations should reside on same VAX as the Baseway configuration global section.

- o perform time-consuming upload/download/compare operations at non-critical times such as between shifts. le;using the Baseway editors, define useful point group associations that applications may reference.

For the Gateway:

- o employ 1MB memory or more.
- o use the maximum Gateway message size of 8000 bytes (configured at Gateway startup time).
- o distribute heavily polled devices across many ports on one Gateway.
- o distribute total device population (total device I/O) across multiple Gateways.
- o poll only as frequently as necessary.
- o encourage device programmers (PLCs, robots) to group related data contiguously to minimize the number of discrete polling operations.

SUMMARY

There are many factors that contribute to the success of implementing Baseway in your plant. I have attempted to describe the characteristics of the two Baseway software products, the Baseway Applications Bus and the Shop Floor Gateway, how Baseway applications are written and what they can do for you, helpful hints on configuring Baseway-related hardware, supporting foreign devices and designing applications.

The Baseway environment is complex because it must solve complex problems in a seemingly always-customized, user-unfriendly factory floor. Baseway is a programmer's tool that can do wonders towards developing a comprehensive manufacturing control system for the end users, those knowledge workers in the factory: production engineers, plant management and plant equipment operators.

REFERENCES AND SUGGESTED READING

- [1] Digital Equipment Corp., BASEWAY APPLICATIONS BUS PROGRAMMER GUIDE, 1985.
- [2] Digital Equipment Corp., BASEWAY APPLICATIONS BUS USER'S AND UTILITIES GUIDE, 1985.
- [3] Digital Equipment Corp., VAX/VMS RUN TIME LIBRARY ROUTINES REF. MAN., 1985.
- [4] Digital Equipment Corp., VAX/VMS REAL-TIME USER'S GUIDE, 1982.
- [5] Digital Equipment Corp., VAX/VMS SYSTEM SERVICES REF. MAN., 1985.
- [6] Digital Equipment Corp., BASEWAY APPLICATIONS BUS SOFTWARE PRODUCT DESCRIP., 1986.

USING MICROPOWER/PASCAL TO IMPLEMENT A DATA ACQUISITION & ANALYSIS SYSTEM

Paul Brown
Sidlinger Computer Corporation
San Antonio, TX

ABSTRACT

A real-time data acquisition and control system requires interrupt-driven software to make effective use of computing resources. The MicroPower/Pascal development environment is a powerful tool for creating an event-driven, multi-tasking system on a dedicated, inexpensive microcomputer such as an LSI-11/2 or FALCON single board computer. This paper is intended to serve as an introduction in the use of MicroPower/Pascal in the development of software for concurrent data acquisition, data analysis, device control, and user interfacing.

BACKGROUND

We are supporting a customer who is interested in producing a benchtop instrument for determining the freezing point of various liquids. The existing manual testing method is both tedious and labor-intensive from an operator's point of view. It is expected that the introduction of the new, automated instrument will allow the test operators to perform other tasks while waiting for the test to complete. This paper describes how MicroPower/Pascal was used to develop the necessary software functions for controlling the instrument.

An integral part of the proposed instrument is an internal, ROM-based controller. Several alternatives for implementing the controller were investigated, including chip-level microcomputers, single board computers (SBC) and Standard (STD) bus systems. We suggested the use of a DEC FALCON SBC and software developed with MicroPower/Pascal (MPP). Initially, DEC equipment had not been considered because, although the customer used DEC minicomputers, they had no experience with non-disk-based DEC computers. Further, they weren't aware that some DEC board-level products, such as the FALCON and the AXV-11C Analog-to-Digital Converter were available at prices comparable to those of more primitive SBCs.

After an initial cost analysis in which the DEC equipment was shown to be competitively priced, the customer gave approval for the initial phases of the project. An important factor in their decision was the question of whether or not the project could be completed in a reasonable amount of time in a low-level language on a custom SBC. It was estimated that the development period would be appreciably shortened with the FALCON/MPP system. The longer period that was necessary for a "homebrew" system would have made the project economically questionable.

Software for the FALCON SBC is typically written in Pascal (and MACRO if necessary) with the MPP system or entirely in MACRO (assembly language). MicroPower/Pascal was chosen as the development environment for several reasons:

- The development team was effectively given a "head start" with the existence of the MPP system library of procedures and functions that would be required in any real-time implementation. Primitive features such as terminal I/O, real number support, and prioritized multi-tasking would require significant lead time to develop.

Additionally, Digital supplies drivers for many popular Q-BUS devices.

- Software modifications that were required in the course of development would probably be easier to handle with a high level language.
- Maintainability of the source code would be enhanced by the use of a high level language.
- The multi-tasking capability of MPP greatly simplified concurrent data acquisition and analysis.

INTRODUCTION

The instrument is planned to be a one-button device. To begin a test, all you do is insert a sample of the liquid and press the start/stop button. The instrument will perform the test and report the freeze point temperature on a display. Testing can be interrupted at any time by pressing the start/stop button again.

The liquid that is being frozen behaves in a characteristic manner as shown in figure 1. The freeze point is marked by an inflection point during the initial freezing and also when the liquid is thawed. By monitoring the change in temperature over time (the slope of the curve), the point at which the liquid freezes or unfreezes can be determined.

IMPLEMENTATION

Figure 2 is a block diagram of the hardware in the instrument. The AXV-11 is a DEC board with 8 double-ended channels of Analog to Digital conversion (A/D) and 2 channels of Digital to Analog conversion (D/A). Both the A/D and the D/A have 12 bits of resolution. One D/A channel is used to control the power supply of the cooling unit. A change in output voltage will cause a proportional change in the temperature of the liquid being cooled.

One channel of the A/D is used to take temperature measurements the liquid. These measurements are made every clock tick (60 times a second in our case).

The FALCON SBC controls the overall operation of the system. The FALCON is a 16 bit processor with up to 16K bytes of RAM

and 32K bytes of PROM. It has 2 lines for serial I/O, 24 lines of programmable parallel I/O, and 3 real-time clock rates: 50, 60, and 800 Hz. Two user interface functions will be implemented through the parallel I/O ports: a small status/result display and the start/stop switch.

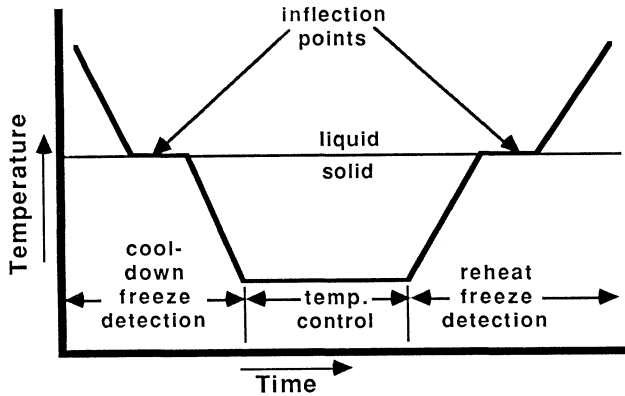


Figure 1 - Reheat Curve for a Liquid

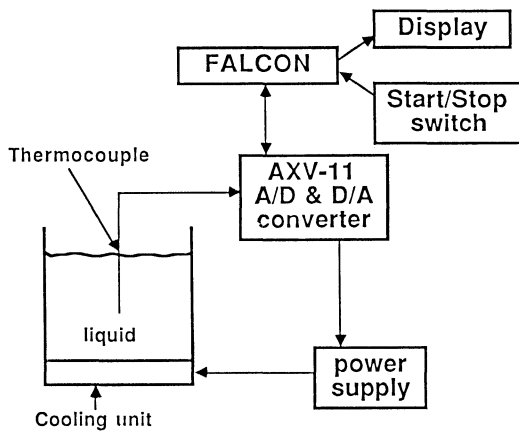


Figure 2 - Freeze Detection Hardware

The software for the system must provide several functions including temperature control, data acquisition, and freeze point detection. Figure 3 is a block diagram of the major dynamic processes in the system. (Figure 3 does not show the static "control" process which activates each of the processes shown in Figure 3 when they are needed.) Data acquisition occurs synchronously and is triggered by a semaphore that is connected (with the Digital-supplied "Connect_Semaphore" procedure) to the system clock. As previously mentioned, this interrupt will occur every 1/60th of a second. By connecting and disconnecting the interrupt to the semaphore (or vice versa) the data acquisition process can be turned "on and off" when needed.

Figure 4 shows a flow chart of the data acquisition process. Once the clock semaphore is signalled, the process commands the A/D to take data for a specific set of channels. The process then waits for the A/D to interrupt when it is done (the A/D interrupt is detected with another "Connect_Semaphore" call). When the interrupt occurs, the process retrieves the data and places it in a ring buffer. The process then loops back and waits for the next clock tick.

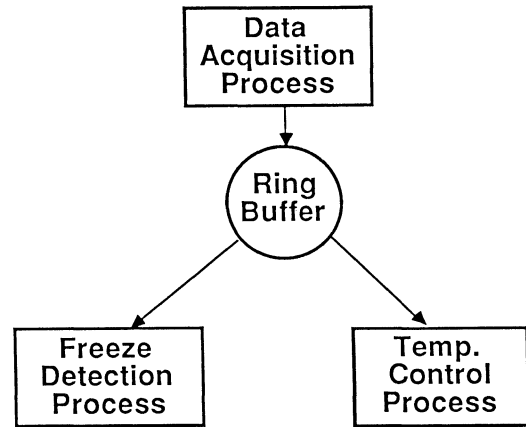


Figure 3 - Freeze Detection Processes

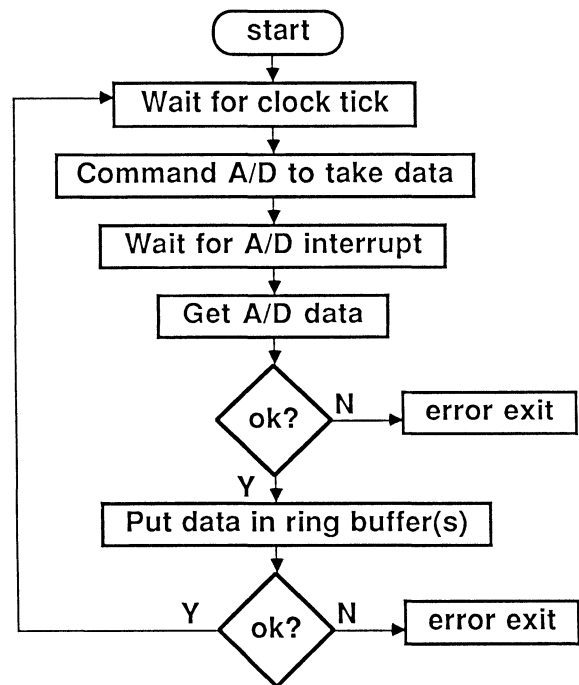


Figure 4 - Data Acquisition Process Flowchart

A ring buffer is a software mechanism that is managed by Digital-supplied procedures and functions. It provides a convenient way to transfer data between processes and does not require the processes to know about each other's operation. Data can be placed in the ring buffer synchronously (or asynchronously) and removed asynchronously, whenever the removing process finds data available.

Figure 3 shows that data is removed from the ring buffer by either the freeze detection process or the temperature control process, but the two do not compete for data at the same time. The freeze detection process uses the data to determine if and when the liquid has frozen. The temperature control process uses the temperature data to achieve a stable temperature in the test cell between periods of freeze detection.

A simple flow chart depicting the freeze detection algorithm is shown in Figure 5. The process waits for a block of data to be available in the ring buffer (a "Cond_Get_Element" call is used). Once a block is available it is removed and averaged. An estimate of the slope is then determined using several adjacent average values. The slope is compared to empirically determined parameters to determine if the freeze point has been reached. If it has, the process ends and either the temperature control phase begins or the final freeze point temperature is displayed (depending on which phase of freeze detection was just completed). Otherwise, the process loops back and waits for the next block of data.

Once the inflection point has been found during the cool-down phase, the liquid must be stabilized at a temperature just below the suspected freeze point (inflection point). Figure 6 shows a flow chart of the temperature control process. Data is retrieved from the ring buffer and is compared to empirically derived parameters. If the comparison indicates that the sample's temperature is stable, the process ends and we move on to the last (reheat) phase of freeze detection. Otherwise, we calculate proportional, integral, and derivative (PID) control values and output the appropriate digital values via the D/A converter. A complete discussion of the PID control algorithm can be found in many control texts including reference [2].

Figure 7 shows that the data acquisition process operates concurrently with either the freeze detection or the temperature control process. Although the figure indicates that two processes are active simultaneously, only one can be truly active at any given time. The A/D process has a higher priority and will be activated every clock tick. The freeze detection or temperature control process, with lower priority, will be active in the remaining free time slots to do asynchronous data processing.

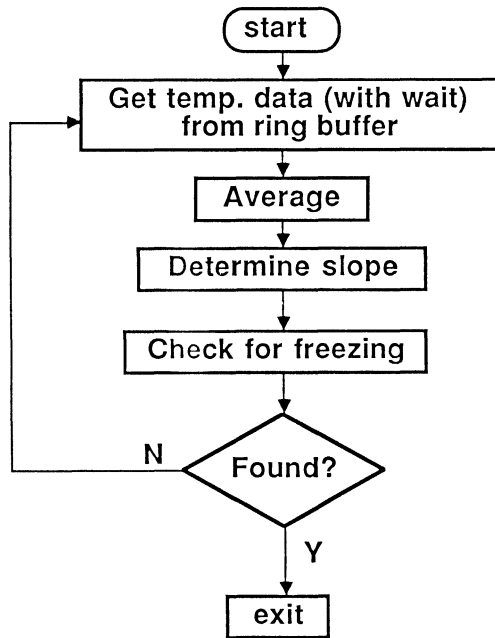


Figure 5 - Freeze Detection Process Flowchart

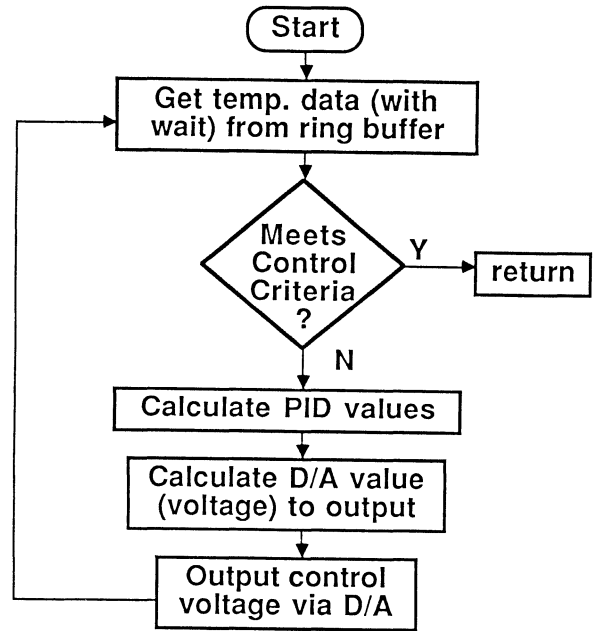


Figure 6 - Temperature Control Process Flowchart

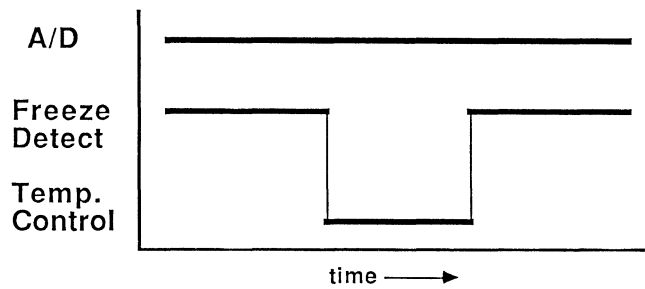


Figure 7 - Process Activity Diagram

CONCLUSIONS

The previous figures have shown the major functions of the freeze detection process. Several other features will be added to the final product, including over-temperature protection, power supply monitoring, thermocouple checkout, and switches for field calibration. These features will be implemented with the remaining A/D and parallel I/O ports. The serial ports are being reserved for future expansion capabilities including the possible connection to a host computer.

We believe that MicroPower/Pascal has reduced the amount of time required to develop a working prototype. Additionally, the FALCON/MPP combination has been found to be economically competitive with alternate solutions. Some of the major advantages of using the MPP development system are:

- The source Code is easy to understand. All the code for this project is in Pascal. There is no MACRO code.

- Since this was a development project, there were many changes in the software due to increasing understanding of the problem as time went on. We found that major modifications could be implemented easily. Again, the fact that the code was written in a high level language was a major factor.
- Because a FALCON SBC was not initially available, the initial development target processor was an 11/02. Changing the kernel build procedure enabled us to easily switch to the FALCON when it became available.
- MPP made the implementation of concurrent processes relatively simple. MACRO code to duplicate or simulate the synchronous data gathering operation and concurrent number crunching would require significantly more complex software.

REFERENCES

1. Digital Equipment Corp., MicroPower/Pascal Language Guide, (1985).
2. Phillips, Charles L. and H. Troy Nagle Jr., Digital Control System Analysis and Design, Prentice Hall, Inc. (1984).

An ultra-high speed data acquisition front end for PDP-11s

E.R. Darken, A.L. Taylor, O. Oakeley, M.S. Spach and S. Herman-Giddens
Duke University Medical Center
Durham, N.C.

Abstract

An expandable high speed data acquisition front end attachable to the Unibus was designed and built in our cardiology laboratory to augment an existing data acquisition system. The principle used in developing the system was to dedicate A/D converters and controlling microprocessors to each data channel. The front end is capable of collecting samples at rates up to 62,500 samples per second per channel for brief durations (about half a second). We currently have a six-channel system which can be expanded on a channel-by-channel basis, with no degradation in the maximum sample rates of the existing channels.

1. Introduction

This paper describes the design and construction of a high-speed data acquisition front-end for PDP-11s. We built, at relatively inexpensive cost, a six-channel system capable of aggregate data acquisition rates as high as 375,000 samples per second. Also, in constructing the system ourselves we were able to tailor it to our needs.

Research in our cardiological laboratory is directed toward the analysis of electrical activity in the heart. Experimenters use an interactive real-time data acquisition system [1] to control conditions in a tissue bath and collect data from electrodes in contact with heart tissue which in turn is stimulated by a pacing electrode.

This system currently is controlled by a PDP-11/44 running a large standalone program which responds to user commands while it controls the tissue bath, collects data and moves selected sample runs to permanent storage [2]. The original equipment could collect data over a maximum of 24 channels at aggregate sample rates up to 40,000 per second, the maximum rate for a single channel being limited to 20,000 samples per second. Significant features of the system include facilities for automatic control of experiments [3] and immediate display of recorded data [4]. Both are critical for most of our experiments. The original design goal was to facilitate the recording of isopotential body surface maps. These experiments tended to use all 24 channels at approximately 1,500 samples per second per channel [5].

In recent years, our researchers have been moving more and more from study of macroscopic events in the heart to study of microscopic events, typically involving one cell or a small bundle of cells. As a result, they have been tending more and more toward experiments involving one to six channels using high sample rates. It became apparent that significant research results could be attained at sample rates greater than those allowed by the existing system, constraints being felt in both the aggregate and single channel limitations.

The research goals demanded a system that could collect data over each of six channels at a minimum of 40,000 samples per second, for an aggregate rate of 240,000 samples per second. This specification greatly exceeded the limits of the laboratory system and it became necessary to consider purchasing an entire new system or modifying the existing system substantially. Economic constraints and our needs ruled out the former possibility. Systems which met our requirements cost more than we could afford, and systems we could afford failed to meet one or more of our requirements. On the other hand, we discovered the cost of the individual components for such a system was within our means, and the presence of a well-equipped technical staff made it feasible to design and implement an in-house data acquisition subsystem to augment the existing resources.

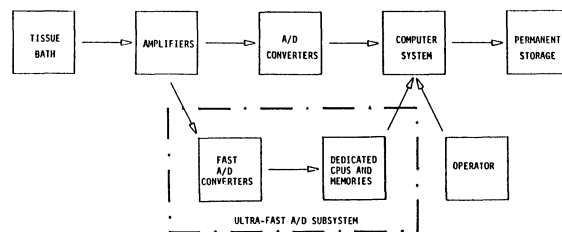


FIGURE 1. Block diagram of the Biophysipedic data acquisition system and how it was augmented by the ultra-fast front end.

2. Functional description

The new data acquisition subsystem is based on A/D converter boards designed and fabricated in-house and Z80-based slave microprocessor boards bought off the shelf. Figure 1 shows how the original laboratory system was augmented by the new data acquisition equipment. One of both board types is dedicated to each data channel, and currently we have a six-channel system. Figure 2 shows the components and functional operation of this system. Data collection is coordinated by a special A/D timing

control board which ensures data is acquired simultaneously on all active channels. The PDP-11/44 initiates sampling by first signaling each slave processor to take X samples and then priming and triggering the A/D timing control board, which starts the A/D conversions at a selectable rate. As soon as a slave processor is started, it collects a sample on each conversion completion signal from its associated A/D converter, which in turn is triggered by the timing control board. Upon collection of X samples, the slave sets its done bit, and the PDP-11/44 transfers the data from the slave's memory to its own memory. The data then may be displayed and stored permanently, and the sampling cycle begins again.

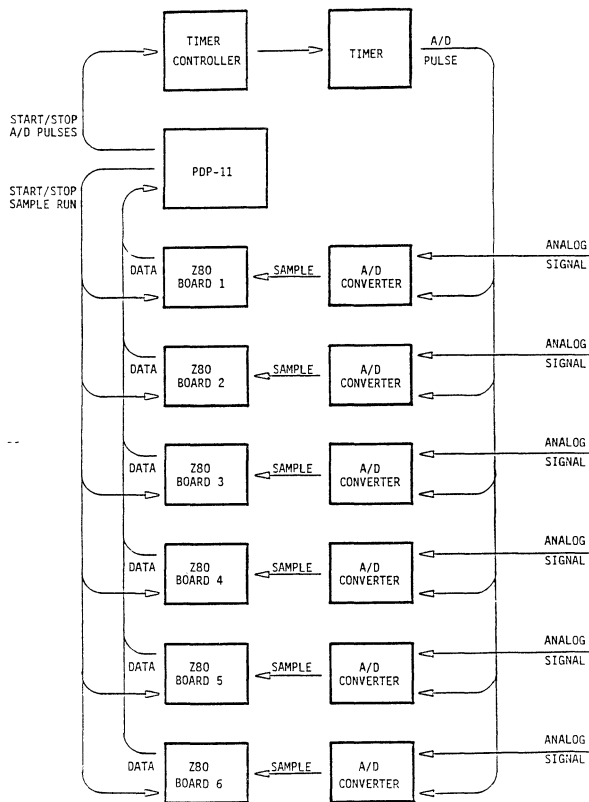


FIGURE 2. The components and data paths of the six-channel subsystem and their relation to the controlling PDP-11 computer.

3. Hardware

There are three basic components to the data acquisition subsystem: the slave microprocessor board, the A/D converter board and the A/D timing control board. Figure 3 shows the signal and data paths, on and off the Unibus, for a single channel set-up.

The slave microprocessor board is an off-the-shelf unit which plugs directly into the Unibus. Each board contains a Z80 microprocessor driven by a 10-megahertz clock, 64 kilobytes of memory and a parallel I/O port. The host PDP-11 controls each board through four eight-bit registers in the I/O page, two for slave memory address and one each for slave memory contents and control/status.

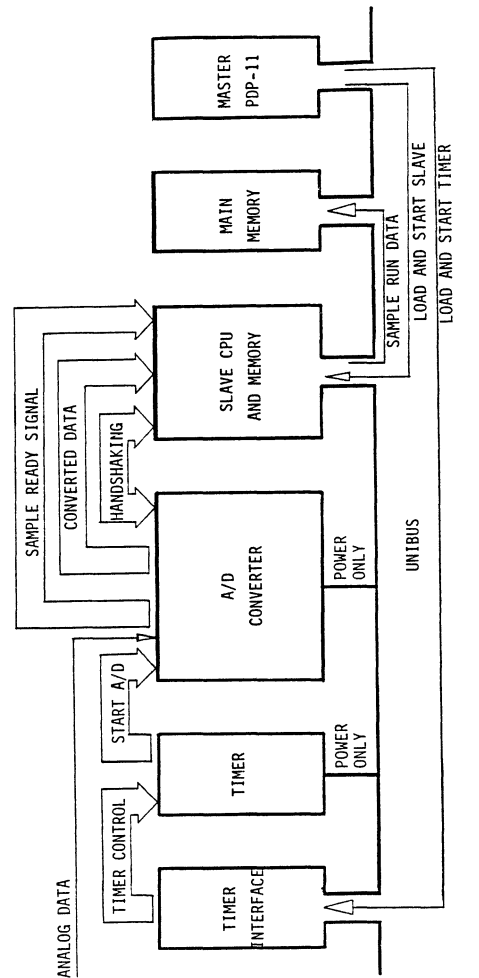


FIGURE 3. Control and communication signals involving a single channel of the subsystem.

The A/D converter board was designed and built in-house. Each board consists of three Analogic IC packages: an MP2712D very high speed 12-bit A/D converter, an MP270 high speed sample and hold and an MP3025 DC power converter. The board draws power through its Unibus backplane connection. It does not have any other Unibus connections; all of its operations are carried out through external connections. Those connections include an 11-wire link with the Z80 microprocessor board, a single line from the timing control clock and a single data line carrying the signal to be converted. Extra hardware on the board synchronizes the operations of the sample and hold and A/D converter, along with presentation of data on the Z80 data path. Amplified data signals coming into the board generally have peaks lying within a 5-volt range around zero. An on-board differential input amplifier with unity gain and a high common mode rejection ratio (CMRR) reduces input signal noise to the .5-millivolt level. When corrected for signal amplification, this introduces noise on the order of 10-microvolt into our data. The reduction of signal noise was the single most time-consuming element in production of the subsystem. The speed at which the A/D conversions take place resulted in an unexpected amount of on-board noise which was largely eliminated only after an intensive search involving

a variety of attempted fixes. The most important alteration to the board to remove this noise was the routing of all component grounds through a single ground (which we call "Mother Ground"). By comparison, the reduction of noise from incoming signals was easier, since methods for designing noise filters were more well-known locally. We settled on the differential input amplifier with high CMRR as the filter which least affected the data signal while reducing noise. Figure 4 shows the level of noise seen on a very slow (4 Hz) sine-wave. The bar at the left of the baseline represents 10 microvolts, and about 50 milliseconds of data is displayed. The signal has been blown up to the point that the 5-microvolt step limit of the A/D converters is clearly visible. We believe the rapid oscillation visible throughout the wave is due to on-board noise.

The third component is the timing control unit. This element is composed of two boards: a DR11-A general purpose interface; and an in-house timer based on an Intel 8254 programmable timer. The in-house board also includes a 10-MHz crystal whose signal is routed through two modulo-n dividers cascaded together to allow a switch selectable pulse rate supplied to the Intel timer. This timer, after counting a preselected number of pulses, in turn sends a pulse to a port which may be connected to multiple A/D converters for synchronization of data acquisition among two or more channels. The master processor loads and starts the timer through the DR11-A.

4. Software

Software control of this data acquisition front-end is conducted through our laboratory's PDP-11/44. The controlling software must deal with two devices in the front-end: the slave microprocessor and the timing control card. Each slave microprocessor must have a program loaded into its memory and started. In this case, the program consists of the equivalent of about 40 lines of Z80 assembly code to store a designated number of A/D converter results in a slave memory buffer. The controlling software also must prime and start the timing control board for a particular sample rate. The clock pulses from this board trigger each A/D converter board and therefore control the rate at which the entire system operates. When the slave microprocessors have collected the designated number of samples, they signal done to the master processor, in this case the PDP-11/44. The controlling software has access to all of slave memory through the slave microprocessor's Unibus registers, so upon reception of the done signal the PDP-11/44 can index through the slave memory and move the data to its own memory.

5. Conclusion

Figure 5 shows data taken over a single channel at 50,000 samples per second. This in-house data acquisition front-end can collect data at rates up to 62,500 samples per second per channel, with no degradation for increasing the number of channels. The modular design of the system and the dedication of equipment to individual channels accounts for this. Currently, our system consists of six channels over which we can collect an aggregate 375,000 samples per second for periods up to half a second. These limits

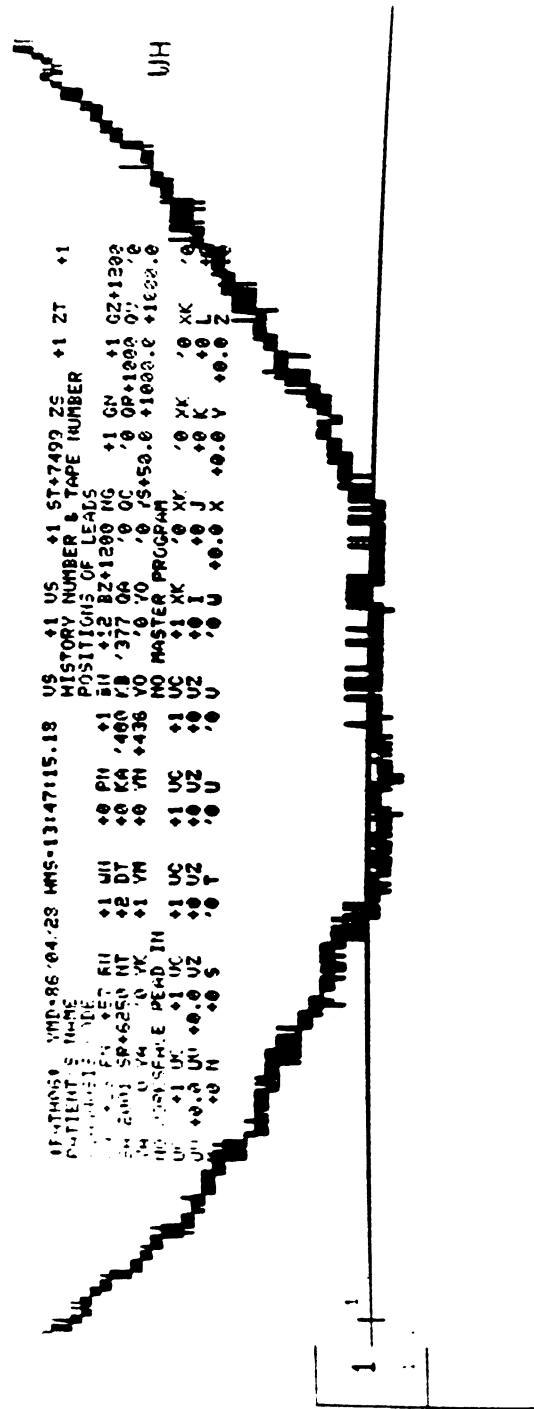


FIGURE 4. An exploded 4-Hz sine wave showing noise added to signals by the subsystem. The bar intersecting the left end of the baseline represents 10 microvolts. The 5 to 10 microvolt resolution of the A/D converters is observable.

are well beyond the requirements stipulated by our researchers. The sample rate limitation is imposed by the speed with which a slave Z80, running at a 10-MHz clock rate, can execute the code for gathering a single sample. The time limitation is imposed by the size of the slave microprocessor's memory, in this case a little less than 64 kilobytes. Both limitations could be raised to significantly higher levels through use of a faster processor and larger slave memory. The trade-off for these speed improvements would be increased cost. In our case, cost was a significant factor, and we found we could meet our specifications with the Z80 board.

6. References

- [1] Barr, Herman-Giddens, Spach, Warren and Gallie, "The design of a real-time computer system for examining the electrical activity of the heart," *Computers and Biomedical Research* 9 (1976), 445-469.
- [2] BATH11 USER MANUAL, Biophysipedic Laboratory, 320 Old Chemistry Building, Duke University, Durham, N.C. 27706, Aug. 1, 1979.
- [3] Herman-Giddens, Warren, Spach and Barr, "Two-level control of a real-time data acquisition and control system for studying electrical activity of the heart," Proc. of the 16th Annual Southeast Regional ACM Conference, pp. 288-294, Atlanta, Ga., April 13-15, 1978.
- [4] Warren, Barr, Herman-Giddens and Spach, "Display of electrical wave forms from the heart," *Computer Graphics*, Vol. 9, No. 3, pp. 17-30, Fall 1975.
- [5] Barr and Spach, "Sampling rates required for digital recording of intracellular and extracellular cardiac potentials," *Circulation* 55:40-48, 1977.

```

#BATH061 YMD=85/05/22 HRS=11:31:58.12 US. +0 US +0 ST+5999 ZS +1 ZT +1
LORI HINOP, P58905, 24 YO, TBATH RUN 1
CH 2101 SR+5000 HT +1 UM +8 BN +17 BZ+1200 MG +1 GM +1 GZ+1200
YA -100 YA -100 VK +1 VM +8 DT +0 KA '400 KB '377 GA '0 QC '1 OR+1000 OU '0
SIMPLIFIED PREAMPURE BEAT MASTERPROG NI 8 REG PAGES MI 8 PAGE START SAK
UIREG INTX XIPREN INT +1 UC +1 UC +1 UC +1 UC +1 UC +1 UC +1 UC +1 UC
UC +1 UC +1 UC +1 UC +1 UC +1 UC +1 UC +1 UC +1 UC +1 UC +1 UC +1 UC
UU +0.0 UU +0.0 UZ +0.0 UZ +0.0 UZ +0.0 UZ +0.0 UZ +0.0 UZ +0.0 UZ +0.0
M +13 N +12 $ +0.0 Y +0.0 V +0.0 W +0.0 X +0.0 X +0.0 X +0.0 X +0.0 X +0.0

```

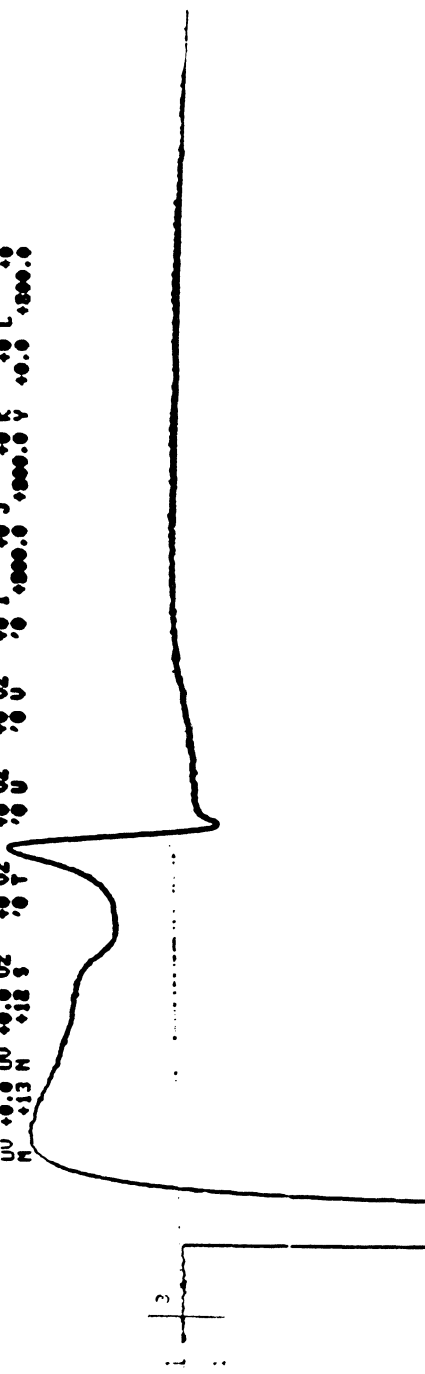


FIGURE 5. Data collected over a single channel of the subsystem. The bar intersecting the left end of the baseline represents 1 millivolt, and the duration of the displayed sample is about 35 milliseconds.

DATA ACQUISITION WITH THE MICROVAX II

W.M. Foreman, J.F. Amann, T. Kozlowski, M.A. Oothoudt
Los Alamos National Laboratory
Box 1663
Los Alamos, NM 87545

This paper summarizes the effort required to install a Digital Equipment Corporation Micro Vax II computer in the data acquisition environment at the Los Alamos National Laboratory Medium Energy Physics Division.

The standard data acquisition systems at the Los Alamos Meson Physics Facility (LAMPF) have used either a PDP-11 and more recently a VAX-11/750 as the host computer. The obvious cost/performance ratio of the MicroVAX II, made it highly desirable to determine the feasibility of using it as replacement for the older PDP-11 based systems. The main concerns in using the MicroVAX were in the ability to maintain compatibility with our existing data acquisition software system known as Q (1,2), and to utilize existing PDP-11 peripherals wherever possible. In particular, it was necessary to continue use of CAMAC and the LAMPF standard micro-programmed CAMAC branch driver (MBD) (3) as well as existing industry standard 1/2" magnetic tapes on which the experimental data is recorded.

The MBD is an 18-bit Unibus device which operates as an independent list processor performing sequences of CAMAC commands associated with the specific experimental triggers. Data acquired through CAMAC is temporarily buffered by the MBD and then moved via a DMA transfer to the host memory. Under VMS, the Q system interfaces to the MBD through the "connect to interrupt" system service and a driver which handles the completed data transfers. Low speed CAMAC operations, i.e. experimental setup and control, are handled by a VMS driver, CC:, written by a collaborator at U.C. Berkeley, R.P. Singh.

The tape drives are after-market drives which interfaced to the Unibus via a Unibus coupler (Dilog DU132) which emulates the Digital Equipment Corp. (DEC) TS11 tape controller. On the VAX-11/750's we have used Kennedy 9100/9220 and 9400 drives, the Dilog coupler, and the standard DEC VMS driver for TS11s. On the MicroVAX, these drives were to be interfaced directly to the Q-bus using a Q-bus version (DQ132) of the same coupler. The older 9100 series drives provide dual density (800/1600 BPI) recording and the newer 9400 drives provide tri-density (800/1600/6250 BPI). This does not affect TS11 emulation, except that the density must be selected manually, since the TS11 is a single density drive.

In order to attach the MBD to the Q-bus, we purchased an after-market 22-bit Q-bus to Unibus converter, the Able Microverter. Our first attempts to use this device failed. The MicroVAX II could not address the MBD. Investigation of the addressing revealed an incompatibility between the Microverter and the MicroVAX II Q-bus. Unibus I/O page addresses begin at 760000 octal and the MBD

decodes all 18 bits of the address on the Unibus, requiring that bits DA13-DA17 be set. The Q-bus protocol allows devices to use only the low order 13 bits for addressing. A Q-bus device must decode RBS7 to determine I/O requests. In order to continue with our test, we modified the Able hardware to provide the high order bits for the MBD address. It is our understanding and experience that Able has corrected this incompatibility in later versions of their product which will be installed on a MicroVAX II. The Unibus VMS device drivers for MBD operation were copied directly from a VAX-11/730 and worked without modification on the MicroVAX.

A Dilog DQ132 was installed on the MicroVAX II and the standard DEC supplied TS11 device driver was copied from the 11/730. All that was required to make the tape operational was to run a VMS SYSGEN in order to define the TS11 to the system. While DEC does not explicitly support TS11 operation with version 4.1 of MicroVMS, we have had no problems.

The Q system software was copied to the MicroVAX II and a simple data acquisition system was tested. There were no apparent problems in the system so plans were made to install a MicroVAX II in an actual experiment. That system was used around the clock from mid August 1985 until the end of our operational cycle in mid December. While there were no major problems associated with its use, a few minor difficulties were encountered. The first MicroVAX we ordered was housed in a BA23 enclosure with a single RD-52 disk and 2 MB of memory. Our experience indicates that the BA23 enclosure is inadequate for our needs with respect to power for peripherals and in ease of service. The small disk and limited memory also resulted in some problems. In particular, the default SYSGEN parameters for swap and page files were not sufficient for our needs. Both parameters were changed to match those in use on the VAX-11/750 systems. All future systems will be housed in a BA-123 enclosure and will be equipped with 5 MB of memory and an RD-53 disk.

One final area of concern was that the use of the Q-bus, as opposed to the Unibus, might somehow limit the rate at which data could be acquired in the MBD and passed through the VAX and onto a data tape. In order to answer this question two comparisons were made of a VAX-11/750 and a MicroVAX II. The first test involved writing data to tape using a Kennedy 9400 tape drive at 6250

BPI/45 ips with 12.8 KB records. Data throughputs of 196 KB/sec and 199 KB/sec were achieved on the 11/750 and MicroVAX II respectively. The second test involved using the MBD to transfer blocks of data from the MBD memory to VAX memory. Two versions of the MBD code were tested, one which released the bus after every word transferred, the second released the bus only after every fourth word was transferred. In the latter mode average transfer rates in excess of 600 KB/sec were measured for both the 11/750 and the MicroVAX. In addition, by comparing the difference in average time per word transferred in two modes, one can separately infer a bus arbitration time and a transfer time per word. The results are presented in the table below.

	11/750	MicroVAX II
Arbitration time (microseconds)	1.4	1.2
Transfer time/word (microseconds)	1.6	1.8

For both these tests of bus speed, the MicroVAX II was essentially equivalent in speed to the 11/750. In an actual data acquisition environment we expect the MicroVAX II to be superior to the 11/750 because of its higher intrinsic speed in data handling.

- (1) M.A. Oothoudt et.al., "DESIGN OF THE RSX-11M Q SYSTEM", IEEE Transactions on Nuclear Science, Vol. NS-28 No. 5, October 1981.
- (2) M.M. Minor et.al., "A SOFTWARE SYSTEM FOR DATA ACQUISITION IN NUCLEAR PHYSICS EXPERIMENTS USING CAMAC", IEEE Transactions on Nuclear Science, Vol. NS-23, 459 (1976).
- (3) L.R. Biswell, R.E. Rajala, "DESIGN AND OPERATION OF A MICROPROGRAMMED BRANCH DRIVER FOR A PDP-11 COMPUTER", LA-5144 (Los Alamos), May 1973.

CONVERSION FROM RT-11 TO MICRO-RSX FOR REAL-TIME DATA ACQUISITION AND ANALYSIS

Mitchell E. King
Northrop Services, Inc. – Environmental Sciences
P.O. Box 12313
Research Triangle Park, NC 27709

DISCLAIMER

The research described in this paper has been reviewed by the Health Effects Research Laboratory, U.S. Environmental Protection Agency and approved for publication. Approval does not signify that the contents necessarily reflect the views and policies of the Agency nor does mention of trade names or commercial products constitute endorsement or recommendation for use.

ABSTRACT

Many scientists who use Digital Equipment Corporation microcomputers utilize the RT-11 operating system for the acquisition of real-time data in the laboratory. For these researchers, the work and time required to learn a new operating system and reprogram software prevents them from upgrading their laboratory computer resources. However, there are several advantages in upgrading to Micro-RSX, a multiuser, multitasking system. Some advantages include sharing of hardware resources by many experimental setups, multiterminal support for concurrent data analysis, and reduced costs per experimental setup.

This paper presents simple techniques for converting FORTRAN/Macro-11 single-user programs that run under RT-11 to run concurrently under Micro-RSX. The concept of using a shared common region that maps the I/O page will be introduced along with a simple method to use Macro-11 routines from RT-11 with Micro-RSX. Techniques for overlaying programs and setting up group global event flags for dual-task control will be illustrated. Programs rewritten for Micro-RSX to acquire and analyze pulmonary physiology data will be used as examples.

INTRODUCTION

For many years, scientists have used microcomputers to collect and analyze data in the laboratory. Digital Equipment Corporation (DEC) markets the PDP-11 series that utilizes the RT-11 operating system for this purpose. This system, although very fast in processing speed, has limited memory addressing capabilities and limits the laboratory to collect data from one experiment at a time. Previously, the only solution, which proved too costly for most researchers, was to purchase a series of computer systems. With the vast improvements in computer technology over the past few years, multiuser-multitasking real-time systems have become cost effective in the laboratory.

This paper outlines a method to convert real-time data acquisition programs, using the RT-11 single-job operating system, to the Micro-RSX multiuser operating system. This technique is based on the concept of shared common regions and shows the RT-11 programmer how to map the I/O page in RSX (the top four Kbytes in memory designated for control of I/O devices) so that user programs can access device registers to control I/O operations. Overlaying techniques and uses of group global event flags for dual task synchronization will also be discussed. Lastly, the experimental setups at the Northrop Services, Inc. – Environmental Sciences, Pulmonary Functions Laboratory will be presented as an example of real-time data collection using this technique.

METHODS

Users of the RT-11 operating system for real-time data collection either employ previously written subroutines or use their own custom-written subroutines to drive their I/O interfaces. These subroutines generally manipulate the control and data registers of specific boards to implement the desired operations. The programs often are written in Macro-11 assembly language. Digital has, for many years, assumed users are proficient Macro-11 programmers; however, in the scientific community, a laboratory scientist may only be familiar with FORTRAN. Assuming the scientist has FORTRAN programs that use Macro-11 subroutines to collect data under the RT-11 operating system, the method described below will allow for easy conversion to the Micro-RSX operating system.

Shared Common Region

Steps for creating a shared common region that maps the I/O page in Micro-RSX (e.g., for an A/D converter controlled by a real-time clock and a D/A converter) are as follows:

Step 1: Determine and write down the control status and data registers (CSR & BUF) of your I/O boards. Except for a specific design reason or conflict of addresses, these will be set for the standard LSI-11 bus floating address assignments (See PDP-11 Microcomputer Interfaces Handbook, Appendix A).

Step 2: Rearrange these numbers in an increasing incremental list.

For example:

Device 1

```
A/D CSR    = 170400 (ADCSR)
A/D Buffer  = 170402 (ADBUF)
```

Device 2

```
Clock CSR  = 170420 (CKCSR)
Clock Buffer = 170422 (CKBUF)
```

Device 3

```
D/A CSR Ch 0 = 170440 (DAC0)
```

Step 3: Subtract (octal subtraction) 160000 from the lowest address (ADCSR in Device 1).

```
i.e., 170400
      -160000
      -----
          10400 (initial offset)
```

The offsets are used to designate the number of bytes from the start of the shared common region to the first register location (or between registers) that are then mapped to the task. This ensures that each label (e.g., ADCSR) will be located at the proper address (160000 is the base starting address of the I/O page).

Step 4: Now subtract (in octal) the next address above the previous device (e.g., Device 1, Buffer address A/D Buf = 170402 + 2 = 170404) from the next device (e.g., Device 2) CSR.

```
i.e., 170420
      -170404
      -----
           14 (next offset)
```

Step 5: Continue this procedure until all the offsets have been defined.

```
i.e., 170440
      -170424
      -----
           14 (next offset)
```

Step 6: Create the shared common region by writing the following simple Macro-11 program on the RSX system.

```
i.e.,
.TITLE DEVCOM
;
.PSECT DEVCOM,GBL,D,RW,OVR
;
. = . + 10400 ;Initial offset from I/O page base
;
ADCSR:: .WORD 0
ADBUF:: .WORD 0
;
. = . + 14 ;Next offset
;
CKCSR:: .WORD 0
CKBPR:: .WORD 0
;
. = . + 14 ;Next offset ...
;
DAC0:: .WORD 0
.END
```

The PSECT directive establishes a program section named DEVCOM.

The double colon (::) defines the label as a global symbol.

Each .WORD 0 designates the word space that is mapped over the CSR or buffer register for each device.

The . = . advances the location counter the specified amount (. = . + 14 is equivalent to .BLKB 14 or .BLKW 7).

Step 7: Compile the source listing using the Macro Assembler.

i.e., MAC DEVCOM

Step 8: Link the object code with the following command file

i.e.,

```
LINK @DEVCOM

LB:[1,1]DEVCOM/CO/PI/-HD, LB:[1,1]DEVCOM/-WI/-SP, -
[1,1] DEVCOM = DEVCOM
/
STACK = 0
PAR = DEVCOM:17760000:20000
//
```

The syntax to link a task is TASK-IMAGE,MAP-FILE,SYMBOL-DEFINITION-FILE = INPUT-FILE with the appropriate switches.

The symbol-definition-file (STB) contains required linkage information for the operating system about the mapped region.

Switch /CO causes the task builder to build a shared common region.

Switch /PI causes the task builder to build a position-independent region. This PI switch allows the task builder to place the shared common region in any position within the user-task virtual address space.

Switch /-HD directs the task builder to exclude a header from the task image because headers are only required in executable tasks.

Note that switch /IP (task maps the I/O page) is omitted because this is the common that maps the I/O page; tasks using this method will map to this region. Also, switch /IP is the default.

The option STACK = 0 suppresses the stack area in the task image. Stack areas are only required with executable tasks.

The option 'PAR = ...' identifies the partition for which the task is built. This option defines the base of the I/O page at memory address 17760000, and the size of the partition (the octal number of bytes in the partition).

Step 9: Examine the memory allocation map in [1,1]DEVCOM.MAP to verify that the global symbols are offset to correspond exactly to the actual address registers.

i.e., Global symbols:

```
ADCSR 170400-R CKCSR 170420-R
ADBUF 170402-R CKBPR 170422-R
DAC0 170440-R
```

If these addresses do not correspond to the device registers, the offsets are incorrectly defined in DEVCOM.MAC

Step 10: The final step needed to correctly set up a shared common region is to inform the RSX executive that the region exists. This is done with the SET PARTITION command and is most efficiently executed if placed in a startup command file (in Micro-RSX place command in [1,2]FASTART.CMD).

i.e.,

```
SET PAR = DEVCOM:177600:200/DEVICE
```

This command uses values that are in 64. (decimal) byte (100 octal byte) increments. Therefore, the two trailing zeros are stripped from the 22-bit base address and the size of the partition (17760000 → 177600, and 20000 → 200).

The /DEVICE qualifier identifies the partition as a common partition for mapping device registers.

When the SET PAR command is implemented in Micro-RSX, the partition DEVCOM is displayed (using the SHOW PAR command) as a named common (!DEVCOM!) subpartition to the partition named IO PAR. This partition is preassigned by DEC in the Micro-RSX distributed code, but it is not documented.

Sampling Routine

Now that the shared common region has been created and implemented, it is very simple to change the RT-11 sampling routine to use the shared common region in RSX. All that is required is removal of the statements in the assembled subroutine that define the CSR and Buffer registers for that device (place a semicolon in front of the definitions), since the device assignments have already been established as global variables in the shared common region. The RSX task builder takes care of all global references in the built task. The following subroutine is given as an example. This is a FORTRAN callable subroutine designed to sample up to eight channels of analog data using the AAV11-C A/D converter. The A/D converter timing is controlled by a KVV11-C programmable clock.

```
.TITLE SAMPLE
;SAMPLE.MAC SAMPLE UP TO 8 CHANNELS
;(CONTROLLED BY KVV11-C PROGRAMMABLE CLOCK)
;
;FORTRAN CALL:
;CALL SAMPLE (ICOUNT,IFREQ,NUMPTS,CHCODE,IData)
;
;WHERE:
;ICOUNT - NUMBER OF CLOCK COUNTS BETWEEN SAMPLES
;  FREQ - FREQUENCY OF CLOCK (MUST BE 8, 16, 24, 32, OR 40)
;        VALUE: 8 16 24 32 40
;        FREQ: 1 MHz 100 KHz 10 KHz 1 KHz 100 Hz
;NUMPTS - NUMBER OF DATA POINTS PER CHANNEL
;CHCODE - TO SAMPLE A CHANNEL, SET THE CORRESPONDING BITS
;        (EXAMPLES: CH 0,3 = 9, CH 1,7 = 130, CH 0,1,2 = 7)
;IDATA - THE RAW DATA STORAGE ARRAY
;
;THIS PROGRAM SETS THE GAIN AT 4 (+/- 2.5 VOLTS) AND THE
;CLOCK MODE AT 1 (REPEAT INTERVAL)
;
;M.E.KING NORTHROP SERVICES, INC. AUG-84
;
;FOR RSX, THE FOLLOWING ADDRESS ASSIGNMENTS ARE
;COMMENTED OUT FOR USE WITH THE SHARED COMMON
;REGION, DEVCOM, MAPPED OVER THE I/O PAGE.
;FOR RT, TAKE OUT THE SEMICOLONS AND
;LINK AS USUAL.
;
;ADCSR = 170400 ; A/D CSR ADDRESS
;ADBUF = 170402 ; A/D DATA BUFFER ADDRESS
;CKCSR = 170420 ; CLOCK CSR ADDRESS
;CKBPR = 170422 ; CLOCK COUNT BUFFER ADDRESS
;
;.CSECT
.GLOBL SAMPLE
;
SAMPLE:ADD #2,R5 ;R5 POINTS TO 1ST ARGUMENT
        MOV @(R5) + ,@#CKBPR ;ICOUNT TO CKBPR
        COM @#CKBPR
        INC @#CKBPR ;TWO'S COMPLEMENT
        MOV @(R5) + ,R0 ;IFREQ TO R0
        MOV @(R5) + ,R1 ;NUMPTS TO R1
        MOV @(R5) + ,R4 ;CHCODE TO R4
        MOV (R5) + ,R2 ;IDATA STARTING ADDRESS TO R2
        MOV R4,R5 ;STORE CHCODE IN R5
        ADD #3,R0 ;INITIALIZE CKCSR
        MOV R0,@#CKCSR ;MOVE R0 TO CKCSR & GO
LOOP1:  TSTB @#CKCSR ;TEST FOR DONE BIT SET
        BPL LOOP1 ;BRANCH IF NOT SET
        BIC #200,@#CKCSR ;CLEAR DONE BIT
        CLR @#ADCSR ;CLEAR A/D CSR
        MOV #10,R3 ;R3 COUNTS NO. OF CHANNELS
LOOP3:  MOVB #11,@#ADCSR ;SET GAIN AND CONVERT
LOOP4:  TSTB @#ADCSR ;TEST FOR DONE BIT SET
        BPL LOOP4 ;BRANCH IF NOT SET
        MOV @#ADBUF,(R2) ;STORE CONVERSION
        BIT #1,R4 ;IS BIT IN CHCODE SET
        BEQ SKIP ;NO - DON'T STORE SAMPLE
        TST (R2) + ;INCREMENT ARRAY POINTER
```

```
SKIP:  INCB @#ADCSR + 1 ;INCREMENT CHANNEL NO
        ROR R4 ;ROTATE CHCODE
        SOB R3,LOOP3 ;DEC CH COUNTER & BRANCH
        MOV R5,R4 ;LOAD CHCODE INTO R4
        SOB R1,LOOP1 ;DEC NUMPTS & BRANCH
        CLR @#CKCSR ;CLEAR CLOCK
        CLR @#ADCSR ;CLEAR ADCSR
        RTS PC
        END
```

Overlay File

In our laboratory, one data collection program uses different array sizes, channels, and timing parameters. For these cases, it is more efficient to overlay (share the same address space) all of the various FORTRAN subroutines that call one Macro-11 sampling routine, rather than to have copies of the same sampling code inserted in each FORTRAN routine. An overlay descriptor file (ODL) can be created to accomplish this approach. The ODL file names the subroutines and library references that make up the task.

For example:

```
.ROOT MAIN1-MAIN2-LIB1-LIB2-★(PART1,PART2)
MAIN1: .FCTR ONLANA-COMAND-FILRDY-NEWFLE
MAIN2: .FCTR SAMPLE-IADC-MBELL-DACOUT-XYPLOT-CODE
LIB1:  .FCTR LB:[1,1]F77RMS/LB
LIB2:  .FCTR LB:[1,1]RMSLIB/LB:R0AUL:R0IMPA:R0EXSY

PART1: .FCTR ONLBOY,ONLN2,ONLPV,ONLRST,OLCHEK,ONLVL
PART2: .FCTR SETSEQ,SETGAN
.END
```

This file has two separate segments: the root segment (MAIN1, MAIN2, LIB1, and LIB2), composed of programs or library references that always reside in memory, and the overlay segment (PART1 & PART2), the components of which share virtual address space. The root comprises three sections: the program backbone (MAIN1), the various I/O control routines (MAIN2), and the library references (LIB1 and LIB2). The first line of the ODL file uses the ROOT directive to define the structure of the overlayed task. The hyphen (-) operator designates the concatenation of virtual address space (each subprogram has its own space). As seen, the subroutine SAMPLE is located within the root, which means it can be referenced from any of the overlayed subroutines. The FCTR directive allows you to build a complex tree-structured program in a clear format. The second segment of the ODL file contains the overlayed subprograms. These are designated by commas (,) that represent the overlaying of virtual address space. In the root, parts 1 & 2 are overlayed because they perform different functions within the task at different times. Part 1 represents the individual subroutines that call the A/D sampling routine with different arguments. The asterisk (★) is the autoload indicator. This is used to ensure that a subprogram will be properly loaded when called, regardless of the flow of control within the task.

Task Build

Once the overlay descriptor file has been created, task building is a simple procedure. The ODL file has to be referenced and any task builder options must be included. The following example demonstrates the task build command file for a FORTRAN program named ONLANA.

```
ONLANA/CP,ONLANA.MAP/CR = ONLANA/MP
(Main program with ODL reference)

CLSTR = F77RMS,RMSRES:RO
(Library reference)

ACTFIL = 3
(Number of active files)

MAXBUF = 7000
(FORTRAN maximum buffer size)
```

```

UNITS = 10
  (Number of FORTRAN logical units)
ASG = TT1:6
  (Assign logical unit 6 to TT1)
COMMON = DEVCOM:RW
  (Shared common reference)
TASK = ...ONL
  (Assigned task name when installed)
//

```

The /MP switch informs the task builder that the input file contains an overlay descriptor. This switch automatically prompts for the options input; thus, a single slash is not needed to inform the task builder to switch to optional input.

The option COMMON = DEVCOM:RW declares that the task being built intends to access the shared common region DEVCOM with Read/Write attributes.

Event Flags

RSX event flags are means by which tasks can recognize specific events. In all single central processing unit systems, only one operation can take place at any one time. With RSX and similar operating systems, several tasks can reside in memory simultaneously and compete for CPU time. For system tasks (EDT, TKB, etc.), RSX takes care of scheduling and setting priorities for these operations. User tasks that must have access to the CPU at a specific time, however, must be synchronized. Simultaneously running tasks that use the same device (i.e., an A/D) must also be synchronized so they will not request A/D conversion data at the same time. Event flags are one way to prevent conflicting requests to the same device.

In RSX, three groups of event flags exist. The first set (#1-#32) is unique within one task. The second set (#33-#64) is common flags that can be manipulated by any task in the system. The third set (#65-#96) is known as group global event flags. Group global event flags apply only to tasks running under the User Identification Code (UIC) containing the group number specified when the flags were created (The UIC is the group and member numbers that designate accounts, e.g., 100,1).

Five programming directives (FORTRAN or Macro callable subroutines that perform a specific operating system function) are commonly used to control synchronization using event flags. The directive, CRGF (create group global event flags), needs to be called only once from any of the tasks being synchronized. Once called, all tasks running under the same UIC group can access this particular group of global event flags. Two directives, SETEF (set event flag), and CLREF (clear event flag), manipulate individual event flags. The directive, WAITFR (wait for single event flag) is used to suspend a task's operation until a specified event flag is set. The final directive is READEF (read event flag), which tests the specified event flag and reports its status (set or not set).

A strategy for programming event flags for dual-task synchronization is as follows. Each of two cooperating tasks should be assigned a different group global event flag. One task (Task A) checks to see if the other task's (Task B) event flag is set using the READEF directive. If Task B's event flag is clear, Task A clears its event flag (CLREF), performs its operation, then resets its flag (SETEF). Meanwhile, Task B works in an opposite fashion, calling the directive WAITFR to see if Task A's event flag is set. If Task A's flag is not set, Task B is suspended until the flag is set. If Task A's flag is set, Task B proceeds to set its event flag, performs its operation, then clears the flag. This method prevents both tasks from attempting to use the same device at the same time. If the criterion for each program requires both tasks to sample data for 10 s every minute, the tasks should be synchronized in such a manner that one task samples data at the minute mark, and the other at the 30 s mark. The Mark Time (MARK) and WAITFR directives can be used in conjunction with group global event flags to accomplish this.

The following example shows the event flag programming strategy.

Task A (EF 66)	Task B (EF 65)
CALL CRGF(,IDSW)!Create flags	..
CALL SETEF(66) !Set flag	..
!so Task B	..
!is not	..
!blocked	..
(USER INPUT)	(USER INPUT)
..	..
CALL READEF(65) !See if Task B	CALL WAITFR(66) !See if Task A
!is sampling	!is sampling
!data Yes-write	!Yes-Delay
!message	!until free
!& loop	..
CALL CLREF(66) !No-clear flag	CALL SETEF(65) !No-set flag
!to block	CALL SAMPLE() !Get data
!Task B	..
CALL SAMPLE() !Get data	CALL CLREF(65) !Clear flag
CALL SETEF(66) !Reset flag	..

Physiological Techniques

The Pulmonary Functions Laboratory at Northrop conducts scientific experiments that measure the effects of airborne pollutants on the respiratory system of small mammals. Generally, there are two categories of experimental setups. The first setup (Task A - see Figure 1) is used to perform a battery of different techniques to assess the physiological function of the lung after exposure. Pressures created from breathing efforts are sensed by four pressure transducers and nitrogen concentration in the exhaled air is monitored. The data are collected at a sampling rate of 250 Hz. Using this task, data collection is requested at random time intervals, thus requiring event flag synchronization. The second setup (Task B - see Figure 1) is used to measure the amount of pollutant retained in an animal's respiratory system during exposure. The experimental protocol requires sampling the pressure transducer to obtain the volume of air moved through the animal's lungs and various monitors to analyze gas concentrations before and after it passes the animal's face. These data are also collected at a rate of 250 Hz for 4 s approximately once per minute.

Event flag synchronization allows both tasks to run simultaneously, without interfering with one another. Suppose, for example, Task B's minute had elapsed and it was time to obtain the next sample point. However, 5 s before the minute mark, the technician using Task A requested a test that required 10 s of sampling. Event flag synchronization could delay Task B from executing its I/O request until Task A was completed. Although Task B would then be 5 s behind schedule, Task B could use the GETTIM directive to calculate the time lost and resume data collection at the minute mark. Another example of conflict would arise if Task B had just initiated a sample at the minute mark and Task A requested a sample. Task A event flag checking would notify the technician that Task B was using a device and to try again in a few seconds. In this manner, both tasks could not attempt data collection at the same time.

DISCUSSION

While implementing these techniques, two problems arose. Using Micro-RSX (a pregenerated, e.g., built by DEC, RSX-11 M+ operating system), the memory required by the operating system and the two simultaneously running tasks is substantial. We discovered that running the two programs with only 128 KWords (KW) of memory occasionally caused one or both programs to hang. This was due to checkpointing (one task is removed from memory by the operating system to make room for the other task), and in our case, a simple solution was the addition of another 128 KW memory

board. The other problem occurred because Micro-RSX automatically installs system accounting during its start-up procedures. Accounting requests several system statistics every 5 min. We found that if the two tasks were being heavily used when an accounting request occurred, the programs would malfunction. The solution was to disable accounting from the system when using these two programs.

One precaution when implementing shared common regions is that the proper global symbols must correspond exactly to the device address registers (see Step 9 in Methods). Writing or reading device registers that are incorrectly assigned could cause loss of data on disks or improper operation of system or user tasks.

More complicated experiments requiring increased synchronization and timing protocols can be performed using the connect-to-interrupt (CINT) facility in RSX, or by using a device driver to schedule and handle I/O requests. Tasks using shared commons tend to tie up the CPU while waiting for control status bits to change in the control status register of a device, whereas these methods free the CPU until the status changes. These techniques do promote efficient handling of asynchronous events; however, they are difficult to program and beyond the scope of this paper.

CONCLUSIONS

The techniques described in this paper allow one computer system running the Micro-RSX operating system to perform the operations that normally would require two processors under RT-11. The

inherent benefits of RSX over RT include account and directory structures, security, and memory management, which improve a laboratory's overall computer processing capabilities. Naturally, using a more complex operating system requires more intricate and technical programming schemes. Upward migration of existing programming techniques is, however, accomplished with relative ease and minimal effort.

Shared common regions provide the most straightforward approach to upgrade from RT to an RSX-based system. As demonstrated, previously written Macro-11 programs can be converted without much change. The shared common region is simply a list of device addresses. Addition of new devices requires only modifying and rebuilding the common region. For larger laboratories, using different programs at various times, individual shared commons can be built to use specific devices within each task.

Laboratories currently using RT-11 should seriously consider upgrading their computer capabilities by converting to an RSX-based system. As shown, reprogramming should not be considered a drawback when weighing the benefits of switching from RT to RSX.

ACKNOWLEDGMENTS

This work was supported by the U.S. Environmental Protection Agency, Research Triangle Park, NC, under contract # 68-02-4032.

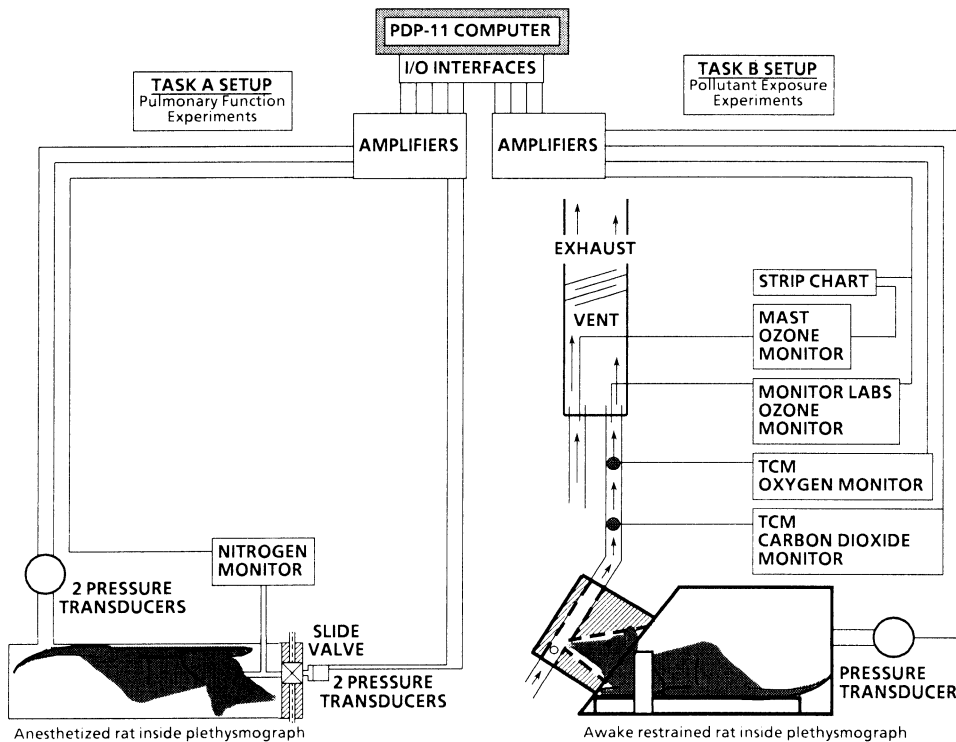


Figure 1. Block Diagram of the Experimental Setups in the Northrop Services, Inc. - Environmental Pulmonary Functions Laboratory.



DATA ACQUISITION AND VALVE CONTROL
FOR ONBOARD OXYGEN GENERATING SYSTEM (OBOGS)
CHEMICAL CONTAMINATION STUDIES

Paul A. Lozano
George W. Miller

Crew Technology Division
USAF School of Aerospace Medicine
Brooks AFB, San Antonio, TX 78235-5301

Kent S. Knaebel
Department of Chemical Engineering
The Ohio State University
Columbus, Ohio 43210

ABSTRACT

At the USAF School of Aerospace Medicine a study is being conducted to evaluate molecular sieve adsorbents based on their ability to remove chemical contaminants from an OBOGS inlet air stream. These adsorbents are used in the molecular sieve oxygen concentrators, the primary component of the OBOGS, to separate oxygen gas from the inlet air stream. This paper describes a Fortran computer program written under the RT-11 operating system which is used by the study to simultaneously control the experiment and collect data.

INTRODUCTION

An Onboard Oxygen Generating System (OBOGS) is a breathing system designed to automatically provide aircrews with an inexhaustible supply of oxygen-enriched gas for the prevention of hypoxia. It has many advantages over conventional liquid oxygen systems, such as: reduced cost, extended flight-time capability, minimum ground servicing, and elimination of fire/explosion hazards.

The OBOGS system produces oxygen aboard the aircraft by application of Pressure Swing Adsorption (PSA) technology. In applying this technique, the aircraft's engine delivers bleed air to an OBOGS concentrator comprising several beds of molecular sieve adsorbent. When air pressure is cycled within the adsorbent beds, the nitrogen component of the air is preferentially adsorbed and vented overboard, thus yielding a nearly pure oxygen product gas suitable for hypoxia protection.

Because these systems use ambient air as their source, the possibility of chemical contamination exists. The OBOGS system must protect crewmen from any substance which, if inhaled, could adversely affect their health. Thus, the selection of a molecular sieve which minimizes the degree of contaminant penetration of the oxygen concentrator's adsorbent beds is of critical importance.

An integral component of the experiment is a PDP 11/03 minicomputer. It uses a Fortran computer program written for data collection and simultaneous experiment control. The program was originally written by K. S. Knaebel and has since been upgraded to meet new requirements imposed on the system.

EXPERIMENT

OBOGS oxygen concentrators are typically comprised of two or three beds of molecular sieve adsorbent, valving, piping, and an electronic timer for automatic valve cycling. In the PSA process the adsorbent beds are alternately cycled through steps of adsorption and desorption (Fig. 1). During the adsorption step, air at high pressure enters the adsorbent bed, whereupon nitrogen is preferentially adsorbed and enriched oxygen is recovered from the bed outlet or product port. The adsorption step is followed by desorption, or venting of the previously adsorbed gases to a lower pressure; usually these gases are discharged to the ambient surroundings. During the desorption step, countercurrent purging of the adsorbent beds with a portion of the enriched product sweeps nitrogen from the interstitial void volume and reduces the amount of nitrogen remaining adsorbed on the sieve. These steps of adsorption and desorption are repeated in each bed, and ultimately result in the production of a continuous stream of enriched oxygen at the unit's outlet.

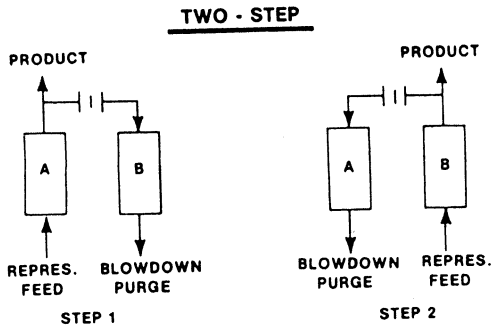
Input File

Figure 1

PSA Process

The PDP 11/03 used to control the experiment sequentially: actuates sampling valves to permit measurement of gas contaminant concentrations; actuates the solenoid valves of the OBOGS concentrator to precisely control system cycle time; and collects up to 14 channels of data at a specified sampling rate (Fig. 2). It is an LSI-11 processor with dual floppy 8-inch disk drives. It has a KVV11-A (Programmable Real-Time Clock), ADV11-A (Analog Input Board), AAV11-A (Analog Output Board), and a DRV11 (Parallel Line Unit). Although not the latest in equipment, this setup has been suitable for our requirements, because it is transportable between experiment sites and has sufficient speed for the current sampling rates used for this experiment.

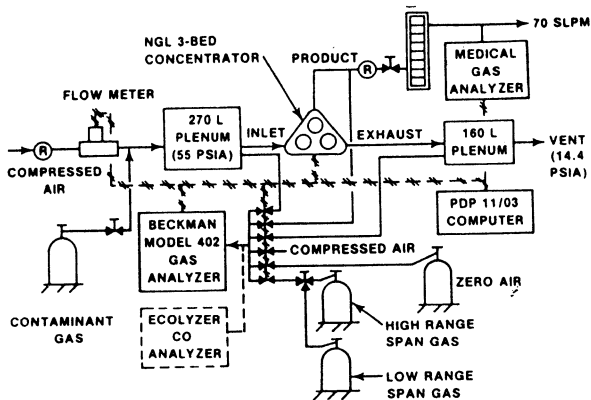


Figure 2

Experimental Apparatus

The first phase of the Fortran Data Acquisition and Control program is to read an input file created by the experimenter using an editor (Appendix A). The input file specifies the experiment operating conditions, data acquisition requirements, and the calibration equation parameters. The experimental operating conditions are specified in the input file remarks section for identification when the results are analyzed.

The alternation of adsorption and desorption is described in the experiment controls and is referred to as steps within cycles. There may be several steps within each cycle depending on the conditions to be tested. The experiment controls define the number of steps within a cycle and the number of cycles in which to collect data. Each step has a step length in seconds, the number of valves to be opened, and the codes of the open valves.

The data acquisition requirements specify the sampling rate, the number of channels to digitize (A/D) and the respective 8-character channel labels. The calibration equations for each channel and their respective parameters are precalculated by the experimenter for converting the data into the appropriate engineering units.

Process

Once the input file is read, it is transferred to an output file to be used for future analysis. The program then starts cycling through the different steps, opening and closing the appropriate valves, sampling the different channels, and displaying on the screen the calibrated values for each channel. The display is in 1-second intervals with channel names displayed every 10 seconds. The program runs continuously through the different steps, displaying data on the screen, but does not collect any data until the line feed key is pressed. This is done to allow the experiment to reach the proper operating conditions the experimenter has selected to analyze. Once the line feed key is pressed the program collects data for the specified number of cycles, outputs it to a file, and terminates.

The PDP 11/03 has only one KVV11 real-time clock and the A/D process requires the clock to sample the data channels at the proper time intervals. The problem then is to simultaneously digitize the data and monitor the elapsed time of a step. The solution is to set the clock frequency to a factor of both the sampling rate interval and the smallest timing unit (one, 1/10, or 1/100 of a second) of the step time. A simple example is a digitizing rate of 10 samples per second and a step time unit of 1 second; the sampling rate interval would be 0.1 second. However, a digitizing rate of 20 samples per second and a step time unit at a tenth of a second would require a time interval of 0.05 second.

At each time interval two counters are incremented; one for the step time and one for digitizing. When the step counter equals the current elapsed step time, the program proceeds to the next step and opens the appropriate valves. When the digitizing counter reaches the sampling rate interval, the required A/D converters are sampled and the data is stored. When one second of data is collected, the last sample of each channel is displayed on the screen. The collection process is double buffered with one second of data in each buffer.

Upon termination of the program two files have been created. One is the configuration file with the experimental conditions. The other is the data file with the step number, elapsed time, and raw data in the form of integer values corresponding to the analog (voltage) input. These files are then transferred to a VAX-11/780 for further processing.

History

The program has been through several phases of modification. It was originally written using the Fortran/RT-11 Extension library and the step times were in 1-second multiples. Also this version was only capable of a maximum of ten steps per cycle. The second version was written with the capability for defining step times in tenths of a second, and a third version was written to accommodate up to 60 steps per cycle.

The latest version uses a Macro subroutine to control the time intervals (KWV11 clock) and to sample from the A/D converters. It is more efficient than using the Extension subroutines because it allows much faster sampling rates and requires less computer space. This program contains all the enhancements of previous versions and is capable of resolving step times to the one hundredth of a second which will be a future requirement.

The PDP 11/03 is being replaced with a Micro PDP-11/73 and will be connected to a LCP01 ink-jet color printer. The new system will allow the experimenter to analyze and plot the results immediately after a run. This will eliminate the transfer of data to the VAX-11/780 and thereby, improve the experiment turn-around time. Also there will not be the limitation on the amount of data that can be collected as there is now using the PDP 11/03.

SUMMARY

We have developed a Fortran A/D program in the RT-11 operating system which is capable of performing two timing functions with only one KWV11 clock. It can sample A/D converters while simultaneously monitoring the step time of a function. It has proven itself to be very useful in OBOGS experimentation and could be easily converted to other uses.

APPENDIX A

Input File Layout

```
1 OXYGEN BREAKTHROUGH EXPERIMENT
2 USING 100% OXYGEN BED SATURATED WITH AIR.
3 OXYGEN FLOW = 20. SLPM (FLOWMETER)
```

```
4 86/04/06
5 3 Number of cycles in the experiment
6 1 Number of sample per average sample
7 20 Sampling rate (sweeps per second)
8 2 Step sequence
9 Feed to bed no. 1 (STEP SEQUENCE 1)
10 2.0,4,01,03,05,07
11 Feed to bed no. 2 (STEP SEQUENCE 2)
12 2.0,4,02,04,06,07
13 6 A/D channel sequence
14 FLOW IN BECKMAN FLOW OUT NITROG% OXYGEN% ARGON%
15 01 CHAN 1 FLOW:INPUT (0-150 LPM) = (0 TO 5 VDC)
16 2,-158.483,7.712E-02
17 01 CHAN 2 BECKMAN (0-10 PPMV) = (0 TO 1 VDC)
18 2,-181.888,8.8764E-02
19 01 CHAN 3 FLOW:OUT (0-85 LPM) = (0 TO 5 VDC)
20 2,-17.483,8.6207E-03
21 01 CHAN 4 N2 PER CENT (0-100%) = (-5 TO 5 VDC)
22 2,-3.2266,2.5208E-02
23 01 CHAN 5 O2 PER CENT (0-100%) = (-5 TO 5 VDC)
24 2,-2.7777,2.523341E-02
25 01 CHAN 6 AR PER CENT (0-100%) = (-5 TO 5 VDC)
26 2,-0.61676,5.13966E-02
```

Lines 1-4 Remarks section for identifying the experimental operating conditions.

Lines 5-8 Cycles, sampling rate and step sequence.

Lines 9,11 Remarks for the step sequence used for screen display.

Lines 10,12 Step time (sec.), Number of valves to be opened and codes of those valves.

Line 13 Number of A/D channels.

Line 14 8-character channel labels. If more labels are needed they can be continued on the next line.

Lines 15,17,19,21,23,25 Type of equation and remarks. (01=linear, 02=quadric, 03=cubic)

Lines 16,18,20,22,24,26 Number of constants for the calibration equations and the values of the constants. (as depicted: intercept, slope)



REAL TIME PERFORMANCE OF MICROVAX II AND MICROVMS

By Richard K. Somes
Digital Equipment Corporation
Marlboro, Massachusetts

ABSTRACT

In many data driven applications the data can be acquired in bursts and will fit in available physical memory. In these applications polling loop techniques can be used. In many event driven applications interrupt latency is the real time attribute of interest. Measurement of these performance attributes characterize real time computers so as to permit the user to make informed cost/performance comparisons.

This paper presents the following real time performance data for MicroVAX/VMS systems:

- o Polled I/O to memory
- o Interrupt latency on various system configurations

A discussion of test methodology will be included in the paper, as well.

INTRODUCTION

Laboratory Data Products is an applications marketing group within Digital Equipment which concerns itself with the computing needs of the scientific end user. MicroVAX puts single user 32 bit computation, and the MicroVMS computing environment within the reach of the scientific user. Today it is economically feasible to dedicate a MicroVAX to a single user application. Although real time computation was possible on the VAX 11/780, it couldn't be done effectively with large numbers of users logged on because of the terminal interrupt loading. The cost of a 11/780 made it useable for real time only in limited applications. Today, it is feasible to commit a MicroVAX to a dedicated application, and it will exhibit good realtime performance under certain situations. The laboratory end user needs performance information in order to select the right system, and to successfully design his application.

Third party developers have an increasing interest in using 32 bit computers, VAX computers in particular, in embedded applications and those who want to do that sort of thing need to know about performance. There is a demand for more sophisticated single-user development environments, and there aren't many more sophisticated development environments other than what's available under VMS.

The performance data our users need is being developed by a consortium of marketing and engineering groups within Digital. LDP, Laboratory Data Products, CMG, (that's the new name for what you know and love as TOEM) Channels Marketing Group and CAEM, Computer Aided Engineering and Manufacturing, are marketing groups. The next two groups are central engineering organizations. MSD, Micro Systems Development is the engineering group responsible for PDP11's, and MicroVAXs will provide performance data for MicroPower PASCAL and VAXELN. There is also an ongoing performance testing effort within the VMS development group.

There has been performance data reported at previous DECUS Symposia. What's new, however, is a will and a commitment to coordinate the activities of these groups to merge the data and start reporting them in way that's useful. This cooperative effort began last fall and a couple of the groups that reported at the last DECUS symposium are now involved. The efforts of the ELN group in benchmarking the interrupt performance of ELN which was reported in Fall DECUS last year now fall under this general heading of performance testing and the PDP11 data which was provided by the Channels marketing group at the last DECUS is also included, so all of these groups are working together. It's already begun and we're continuing here at this DECUS

with my presentation today on a couple of aspects, a couple of attributes, RT attributes of MicroVMS and MicroVAX systems.

The ultimate goal of all this performance testing is to characterize the attributes of computer systems which affect realtime performance. We want to characterize them in such a way that we can understand how applications work. We begin by measuring attributes like interrupt latency or the rate at which we can acquire data using polled IO methods. These are the things I'm going to characterize in this paper. But from here, we need to characterize additional attributes which will be indicated later.

Ultimately we want to be able to say, for instance, whether data acquisition at 100 khz from an A/D converter to disk is going to work on a particular system. We want to be able to define the realtime attributes of the system that decide if that application is going to work. We want to identify the primitives that measurements can be made to determine if a system will work, and which parameters can be adjusted to make a system work. We want to know how to enhance the operation of existing systems and perhaps more important, we want to know how to design better systems. This isn't just performance testing. This is the beginning of an on-going effort in understanding and providing systems with advanced realtime architectures.

TEST METHODOLOGY

Figure 1 is a functional block diagram of the system on which the performance data was acquired. The test philosophy is to start with the minimum configuration that will run, characterize it and then incrementally add loads to show their effects. In the past VMS performance data has been acquired on heavily loaded multi-user systems. It is not unusual in such systems to observe interrupt latencies out into the hundreds of micro-seconds and even in the millisecond range. Lightly loaded systems don't exhibit that kind of behavior at all, as the data will illustrate.

The test system always includes a KA 630 CPU, and 4 megabytes of expansion memory for a total memory load of 5, and the interrupt test module. A DEQNA NI controller, and a VCB01 video subsystem are optional depending on the loading condition being tested. The data describes four system configurations; a standalone MicroVAX II isolated from the Ethernet, a workstation configuration likewise isolated, and both MicroVAXs and workstations with network interfaces.

The test board is a parallel interface board, not the DRV11-WA, but a a DEC product sold in Europe called the DRQ11C.

It is the Q Bus version of the DREll-C and DRUll-C for the Unibus. It is a 22 bit DMA board - 16 bits in, 16 bits out - and it's claim to fame is that it does double buffered DMA. The DMA functionality doesn't really have any bearing on what we're doing here. For the purposes of this testing it is a parallel interface capable of generating interrupts.

External to the parallel interface is connected the test counter shown in Figure 2. It is a simple 16 bit test counter, using four 74LS193 pre-settable up/down counters and a holding register made up of two 8 bit transparent latches. The counter can be turned on or off, and counting can be enabled or disabled. It can be pre-set by writing a 16 bit word to the output register of the parallel interface. The counter always counts up and the outputs are always connected to the holding register. Depending on whether the holding register is in hold mode or transparent mode, these lines are either running with the counter or frozen.

The carry output of the last counter stage is connected to one of the function bits so that the counter can generate an interrupt on overflow.

Handshake lines between the input interface of the parallel IO board and the holding register allow a software generated trigger to freeze the holding register until handshake completes when a polled I/O read is done to the holding register. This triggering of the holding register is synchronized with the counter clock to insure that there is always a valid code stored in the register.

POLLED I/O TEST

The first test done with this arrangement was polled I/O. The MACRO32 code for the test is given in Figure 3. This code is called from a C program, but it could be called from any higher level language. One of the things that distinguishes this routine is that it is called using the SYS\$CMKRNL (Change Mode to Kernel) system service. This was necessary because the polling code does one of a couple of things which require kernel mode access. One of the things is to disable interrupts by raising the processor IPL to 30 for the execution of the polling loop and re-enable them before returning to the calling program. The other alternative is to turn off the processor interval timer and then turn it on again at the end. Both of these are commented out in the listing shown, but the test was run for both cases as well as for the case in which neither was done.

There are three sets of data here. In the first set, polling is done at elevated IPL so that the system is "asleep" for anybody else. In the second set of data the interval timer is off and the system is otherwise idle. In the third set of data the interval timer is on and the system is otherwise idle.

There are two graphs of this data, shown in Figures 4 and 5, one which shows all the data and one which shows the detail of the major peak in the distribution. The distribution of polled I/O times for IPL30, and that for IPL0 with the interval timer off are essentially the same. With the interval timer on, there is a major peak similar to that in the two previous cases, but there are also a couple of discrete groupings of longer sampling times.

Every time the polling loop is traversed, the holding register is triggered and its contents are read into a data buffer in memory. The data is processed by taking the first difference producing an array of times between each successive sample. Anytime there's an interrupt on the system, the instruction stream is pre-empted and a delay is injected between samples. There are three discrete groupings, one at around 60 microseconds, another one around 300, and a third around 600. In summary the minimum polling time for the code shown is 10 microseconds, the maximum polling time is 14, and the most frequent polling time is 10 1/2 microseconds. That translates into a polling rate of 70 - 100 KHZ, 70 KHZ worse case.

In Figure 5 it is clear that the first peak of all three distributions are essentially identical. The spread in polling times is due to the fact that there are other things happening even on an idle system besides the process it is executing. Among them, memory has to be refreshed. In a MicroVAX II the memory is connected to the MicroVAX CPU through a private memory interconnect and is managed by a memory controller on the MicroVAX CPU board. Among other things, that memory controller has to refresh memory periodically and that refresh interval can delay the reply times on a READ thereby extending the length of an I/O cycle.

Let's take a closer look at the outliers in the third test case. They are obviously the result of interval timer interrupt loading on the system. There are several different code paths which can be traversed in response to an interval timer interrupt. One possibility is that there is a timer queue entry. Another possibility is that the quantum allocation for the process at hand is expired. A third possibility is that neither of these conditions exists and the interrupted code stream can be resumed. Each one of those code paths is associated with one of the outlying peaks. My guess is that the shortest interruption occurs when there is neither timer queue entry nor quantum timeout. I am reasonably certain that the second one out is associated with quantum expiration. When the polling loop is run at realtime process priority, the outlier moves in from about 200 to about

125 microseconds. The third peak is probably due to the posting of an AST resulting from a timer queue entry.

INTERRUPT LATENCY TEST RESULTS

The connect interrupt driver, which is supplied as a part of VMS on most VAX systems including the MicroVAX II, is a generic driver which allows the user to write his own start I/O code, interrupt code, cancel I/O code and initialize code and to call it using the QIO system service without having to write an entire device driver. The device driver code provides JSB, RSB linkages to the user defined code. The user defined code and any data locations required for its execution is known as the CIN buffer. Its location and size are defined in the QIO call which initiates I/O from the device. The CIN buffer must be less than or equal to 64K bytes in length. It must, above all, be clean code, because it runs in system context at elevated IPL. If it generates a machine check, it will cause a system crash.

Figures 6 and 7 are fragments of the code used for interrupt testing. In the test 1024 interrupts are accepted for each QIO, and the time data for each interrupt is stored in a data buffer located at the top of the CIN buffer.

In the START I/O routine two temporary locations within the CIN buffer are initialized, one containing the number of interrupts to be processed (1024) and the other containing a pointer to the temporary data buffer. The base address of the device, a DRQ11 in this case, is loaded into another register. Before manipulating device registers, IPL is raised to power fail level to synchronize access with other processes which might be "touching" the device. The test counter is stopped and read to obtain a time measurement from the beginning of the QIO call to the start I/O. It is then preset at -20,000 clock ticks, which amounts to 10 milliseconds and started. Interrupts are re-enabled, and control is returned to the driver and from there to the process which issued the QIO.

Since there is a 10 millisecond delay loaded in the test counter, it is virtually guaranteed that control will return to the user code before an interrupt occurs. When that counter overflows, that code stream is interrupted. The counter continues to run, however, until the interrupt service routine is entered. The ISR fetches the base address of the DRQ11 and immediately transmits a trigger to the test counter. The counter is triggered in the second instruction of the ISR. The effect of the trigger is freeze the contents of the counter. Because the triggering of the counter is synchronized with the test counter clock, it is necessary to test the DONE flag to make sure that the synchronization has occurred before reading

the register. That's accomplished by the two line polling loop at 3\$. With a 2 MHz this loop is never traversed more than once. If we were using an A/D converter doing 20 microsecond conversions, it would be necessary to traverse this loop a number of times before reading the device.

Once the data has been stored in the temporary data buffer, the counter is pre-set to -20,000 again and control is returned to the user. If 1024 interrupts have been acquired a request to execute user defined AST code is posted and the I/O operation is terminated. There are two versions of this code, by the way, one with a fixed pre-set and one with a random pre-set. I've seen one test result where it appeared that there was some sort of synchronization going on between the test source and another interrupt source on the system. In most cases, I didn't see anything like that, but in one case I did and I wrote the random code in order to deal with that specific situation.

The interrupt data in Figures 8, 9, and 10 were gathered on three different system configurations. For the tests of Figure 8 the system includes only a KA630 CPU, 4 megabytes of expansion memory, a disk controller, and the interrupt test board. The terminal is connected to the console terminal port on the KA630 CPU board. For Figure 9 an Ethernet controller is added to this configuration, and for Figure 10 a VCB01 video subsystem is added making the system a VS II with an Ethernet port. These three sets of data show the basic interrupt latency of MicroVMS on the MicroVAX II CPU, and the cumulative effects of additional interrupt loading.

In a minimum MicroVAX II configuration such as was used for the tests shown in Figure 8 the only interrupt sources are the interval timer, the console port, the disk controller, and the test board. In all tests the disk controller was inactive, so that only the other three sources were considered. In the first test the Interval Timer was disabled. The resulting distribution of interrupt latencies is indicative of the raw performance of the processor and the operating system. All interrupts were processed in between 36 and 42.5 microseconds, with a most frequent (almost 80%) time 37.5 microseconds. The variation in times is due to bus arbitration time which is influenced by the instruction being interrupted and by memory refresh.

When the Interval Timer is enabled there are two interrupt sources, and it is possible for the test interrupt to be blocked by Interval Timer ISR code. In this case the maximum interrupt latency stretches out to 75 microseconds, but the minimum and the most frequent values are not significantly effected.

The scrolling load injects a third source

of interrupts. These are also capable of blocking the test interrupt. Apparently the ISR for the console terminal is capable of blocking for extended periods, for the worst case latency is extended to 740 microseconds in this case.

In Figure 9 the effects of adding DECnet loads is shown. In the first data set the DECnet link is quiescent, and the results are similar to those shown in the second data set in Figure 8. The Ethernet controller adds an interrupt load and the worst case latency is extended to 91.5 microseconds. In the second and third data sets, the effects of network activity are shown. In the second set a single remote login/logout transaction was performed over the Ethernet from a LAT server. In the third set a continuous scrolling process was executing from the remote terminal. It was the same scrolling load as used to test console interrupts, but when executed over the Ethernet the worst case interrupt latency was only 122.5 microseconds.

In Figure 10 the effects of a video subsystem are shown. Even the quiescent VCB01 adds interrupt loading to the system. It generates an interrupt for every frame, or 60 times per second. The worst case interrupt latency in the presence of a video subsystem is measured at 266 microseconds. When the scrolling load used in the two previous tests is executed in a VS II window interrupt latencies of about 2 milliseconds are observed.

A word or two about how interrupts are handled on the MicroVAX II are in order. In the MicroVAXII the interval timer interrupts at IPL22, which corresponds to BR6 on the Q Bus. It is serviced at that IPL as well. The console terminal interrupts and is serviced at IPL 20, corresponding to BR4 on the bus. In the tests run for this paper the test device was requesting at BR4, so that it could be blocked by either the clock or the console terminal. It is reasonable to assume that if a BR7 device were the only interrupt source, it would have an interrupt response distribution essentially identical to that observed on an idle MicroVAX II with Interval Timer disabled.

Because interrupts are acknowledged on only one line on the Q Bus, the processor's IPL is raised to 23 regardless of the BR level of the request. Unless the device driver drops to the correct device IPL or forks very quickly in the interrupt service code, it is possible for other higher priority devices to be blocked. In order to get optimum interrupt response for a critical device, it is necessary that interrupting devices be arranged in strict accordance with DEC Std 160 and that all devices are serviced at the appropriate device IPL. At this time compliance with the latter restriction is the responsibility of the device driver writer.

FUTURE PLANS

The next major area of performance characterization is DMA throughput, because DMA is the fastest way to get burst data and the only way to get sustained data flow for buffers larger than 64 KB. It is possible to acquire bursts of data using polled I/O at 70 KHz. With that kind of capability one can grab frames of speech data, for instance, analyze it off-line, and reproduce it, using very simple peripherals like the AXV11-C.

If you want to move data at hardware rate of the bus, you've got to have DMA. The Q Bus as it's operated by most I/O peripherals today exhibits a peak bandwidth of about 1 2/3 megabytes per second in single transfer mode. It's capable of much higher rates (about double that rate) in block mode and that is one of the things we want to characterize for you.

Channels Marketing Group is going to complete the job of baseline characterization of the PDP11. Most DEC realtime users come to where they are today knowing how PDP11's work under RT and RSX and wanting to understand how MicroVAX under MicroVMS and under ELN compares with it. CMG is interested in doing that job and a lot more. The ELN group is committed to characterizing the performance of new VAX systems as they are supported by ELN. All of us, - LDP, CMG and ELN are interested in providing this kind of data. We have a pretty good picture of interrupt response, interrupt latency. We want to characterize interrupt rate; and DMA performance, and the major inter-process and synchronization mechanisms real time users are interested in.

ACKNOWLEDGEMENTS

I am indebted to a number of people who contributed to this paper.

Frank Jones is responsible for the isometric plots of polled I/O and interrupt latency performance. Frank wrote the programs to extract the data from the log files produced by the tests, and to plot it. The plots were produced on an LVP16 plotter connected to a MicroVAX II.

Ivan Goddard built the the test timer circuitry.

Amr Hafez helped with the running of the tests and gathering of the data.

Jane Whitney transcribed the audio tape of my DECUS presentation for use as a first draft of this paper. She also proofed the final draft, and did the paste up work.

MICROVAX II BASED PERFORMANCE TEST SYSTEM

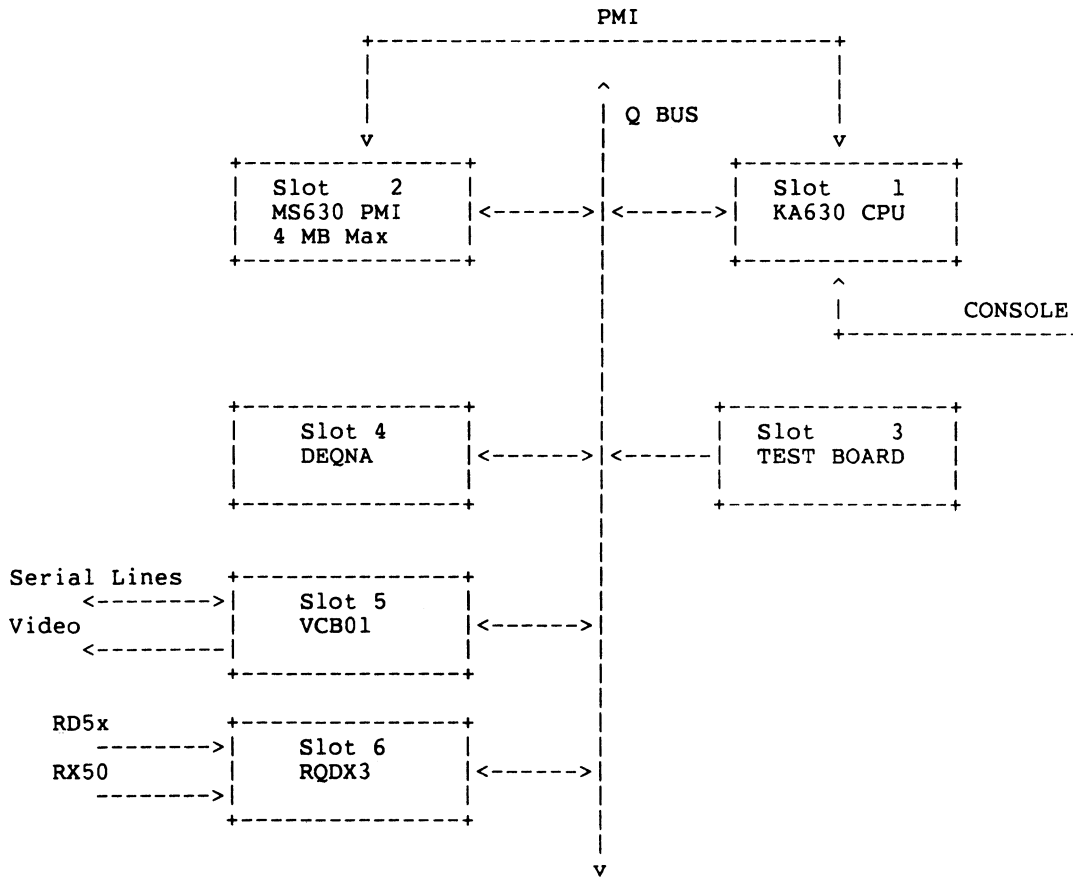


Figure 1

DRQ11 TEST COUNTER

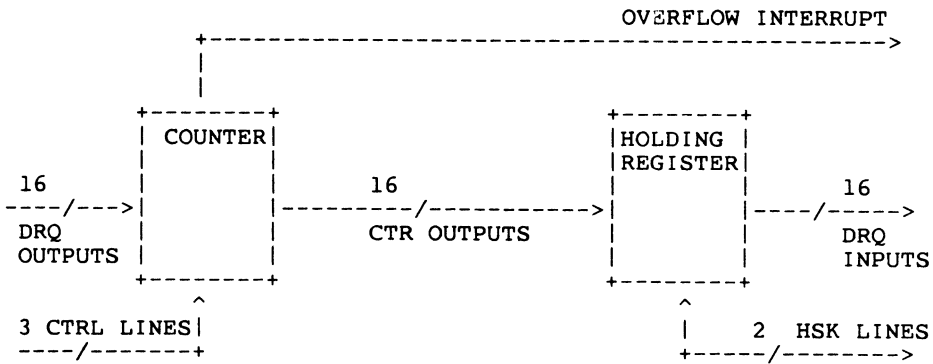


Figure 2

```
;MODULE TO ACQUIRE N (FIRST ARG OF CALL) POINTS FROM A DRQ11-C BY POLLED I/O
;AND STORE THEM IN A DATA ARRAY DEFINED BY THE SECOND ARG OF THE CALL
;THIS VERSION PERFORMS THE ACQUISITION AT IPL 30
```

```
.LIBRARY /SYS$LIBRARY:LIB.MLB/
```

```
$IDBDEF          ; Definition for I/O drivers
$UCBDEF          ; Data structures
$IODEF           ; I/O function codes
$CINDEF          ; Connect-to-interrupt
$CRBDEF          ; CRB stuff
$VECDEF          ; more
$PRDEF           ; Processor register defs
```

```
DRQ_SCR          = 0
DRQ_COR          = 2
DRQ_ADR          = 4
DRQ_DATA         = 6
```

```
CLEAR           = 8
PRESET          = 14
START           = 4
STOP            = 12
TRIGGER         = 5
```

```
.PSECT          DRQ_PIO,PAGE
.ENTRY          DRQ_PIO, ^M<R2,R3,R4,R5,R6>

    MOVL        4(AP),R3          ; GET THE VALUE OF N AND SET LOOP CTR
    MOVL        8(AP),R4          ; GET THE ADDRESS OF THE DATA ARRAY
    MOVL        12(AP),R2         ; GET THE BASE ADDRESS OF THE DRQ11-C

;    DSBINT     #30                ; ELEVATE IPL TO 30
;    MTPR      #0, #PR$_ICCS       ; TURN THE TIMER OFF
$1:  MOVW      #TRIGGER,DRQ_SCR(R2) ; TRIGGER THE HOLDING REGISTER
$2:  MOVZWL    DRQ_SCR(R2),R5       ; TEST THE DATA AVAILABLE FLAG
    BBC       #11,R5,$2           ; AND WAIT HERE UNTIL IT SETS
    MOVW      DRQ_DATA(R2),(R4)+   ; STORE THE DATA
    SOBGTR    R3,$1               ; TEST THE LOOP COUNTER
;    ENBINT     ; RESTORE IPL
;    MTPR      #^X40, #PR$_ICCS    ; TURN THE TIMER ON
    MOVZWL    R2,R0
    RET                          ; RETURN

LOOP_END::      .LONG    0

;
.END
```

Figure 3

MicroVAX II (Polled I/O)

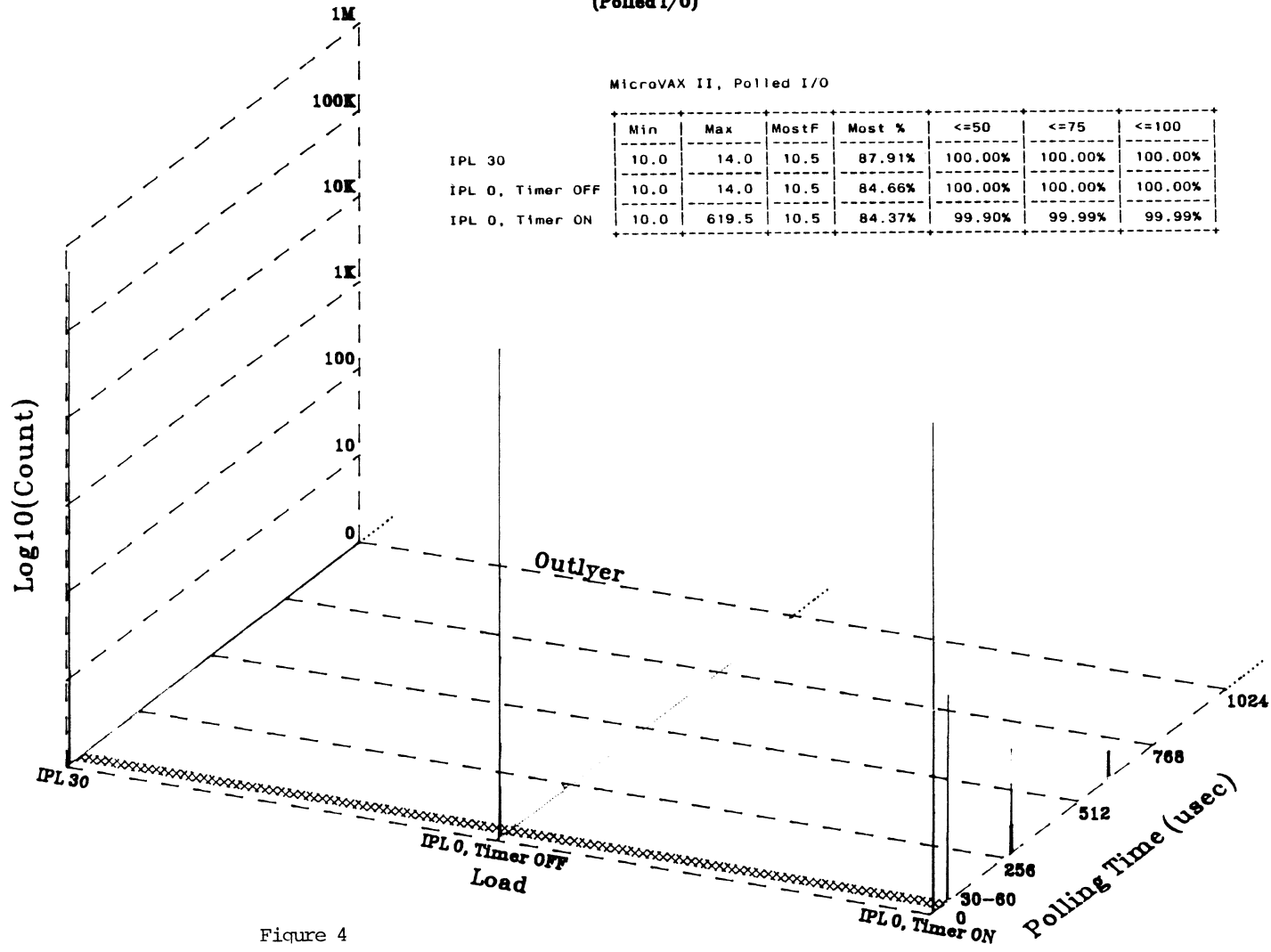


Figure 4

MicroVAX II
(Polled I/O)

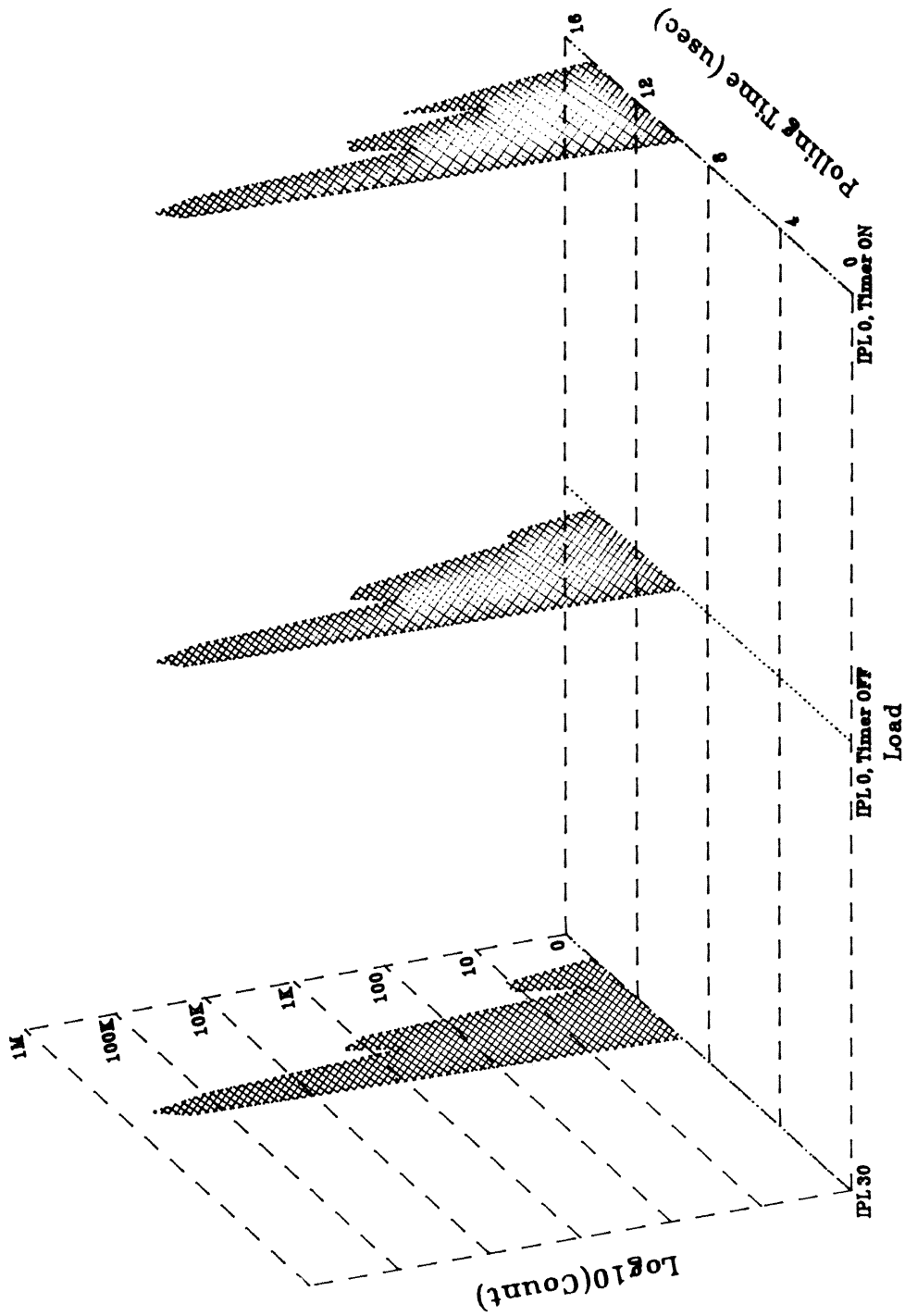


Figure 5

```

.SBTTL LDPIO_CIN_START, Start I/O routine

; ++
; LDPIO_CIN_START - Starts the DRQ
;
; Functional description:
;
; Inputs:
;   R2   - Addr of count arg list
;   R3   - Addr of IRP
;   R5   - Addr of UCB
;
;   0(R2) - arg count of 4
;   4(R2) - Address of the process buffer (system mapped)
;   8(R2) - Address of the IRP (I/O request packet)
;   12(R2) - Address of the device's base I/O address
;   16(R2) - Address of the UCB (Unit control block)
;
; Outputs:
;   none
;
;   The routine must preserve all registers except R0-R4.
;
; --

LDPIO_CIN_START::
    MOVW    #1024,POINT_COUNT      ; Set point count 1024 points
    MOVL   4(R2),POINTER          ; Set up pointer into data buffer
    MOVL   12(R2),R0              ; Get value of the DRQ I/O address
    DSBINT                                ; Raise IPL to IPL$ POWER
    MOVW   #STOP_CTR, DRQ_SCR(R0)  ; STOP CTR AND READ_QIO TIME
    MOVW   #TRIGGER_CTR, DRQ_SCR(R0)
    MOVW   DRQ_DBR(R0), QIO_TIME
    MOVW   #PRESET_CTR, DRQ_SCR(R0) ; PRESET DRQ TEST COUNTER
    MOVW   #45356, DRQ_DBR(R0)
    MOVW   #START_CTR, DRQ_SCR(R0) ; START CTR AND ENABLE DRQ INTERRUPTS
    ENBINT                                ; Restore IPL
    MOVZWL #SS$_NORMAL, R0         ; Load a success code into R0.
    RSB                                     ; Return

```

Figure 6

```

.SBTTL LDPIO_CIN_INTERRUPT, Interrupt service routine

; ++
; LDPIO_CIN_INTERRUPT
; Functional description:
;
;
; Inputs:
;   R2   - Addr of counted agr list
;   R4   - Addr of IDB
;   R5   - Addr of UCB
;
;   0(R2) - arg count of 5
;   4(R2) - Address of the process buffer
;   8(R2) - Address of the AST parameter
;   12(R2) - Address of the device REGISTERS
;   16(R2) - Address of the IDB (interrupt dispatch block)
;   20(R2) - Address of the UCB (Unit control block)
;
;
; Outputs:
;
;   The routine must preserve all registers except R0-R4
;
; --
CIN_BUF_ADD = 4           ; Address of CIN buffer
AST_PARM = 8             ; Offset to AST parameter address
CIN_CSR_ADD = 12        ; Address of BASE I/O ADDRESS

LDPIO_CIN_INT::
  MOVL   CIN_CSR_ADD(R2),R0 ; Get BASE I/O address
  MOVW   #TRIGGER_CTR, DRQ_SCR(R0) ; TRIGGER TEST COUNTER
3$:     MOVZWL DRQ_SCR(R0), R4
        BBC   #11, R4, 3$
        MOVL  POINTER, R3
        MOVW  DRQ_DBR(R0), (R3) ; Read data (assume no error)
        ADDL  #2, R3
        MOVL  R3, POINTER
        DECL  POINT_COUNT ; Dec point count
        BNEQU 10$ ; Done ?
        MOVW  #STOP_CTR, DRQ_SCR(R0) ;
        MOVW  #CLEAR_CTR, DRQ_SCR(R0) ;
        MOVW  #STOP_CTR, DRQ_SCR(R0) ;
        MOVW  #START_TIM, DRQ_SCR(R0) ;
        MOVZWL #1, @8(R2) ; RETURN #1 to AST PARAMETER
        MOVL  #1, R0 ; 1 means queue the AST, 0 means don't
        RSB

10$:    DSBINT ; Not done; IPL$ POWER
        MOVW  #PRESET_CTR, DRQ_SCR(R0) ; PRESET DRQ TEST COUNTER
        MOVW  #45356, DRQ_DBR(R0)
        MOVW  #START_CTR, DRQ_SCR(R0) ; START CTR AND ENABLE DRQ INTERRUPTS
        ENBINT ; Restore IPL
        MOVL  #0, R0 ; 1 means queue the AST, 0 means don't
        RSB

```

Figure 7

**MicroVAX II
(Interrupt Latency)**

MicroVAX II

	Min	Max	MostF	Most %	<=50	<=75	<=100
Timer OFF	36.0	42.5	37.5	79.61%	100.00%	100.00%	100.00%
No Load	38.0	75.0	40.0	83.49%	99.95%	100.00%	100.00%
Scrolling	36.0	740.0	40.0	68.68%	97.59%	98.89%	99.53%

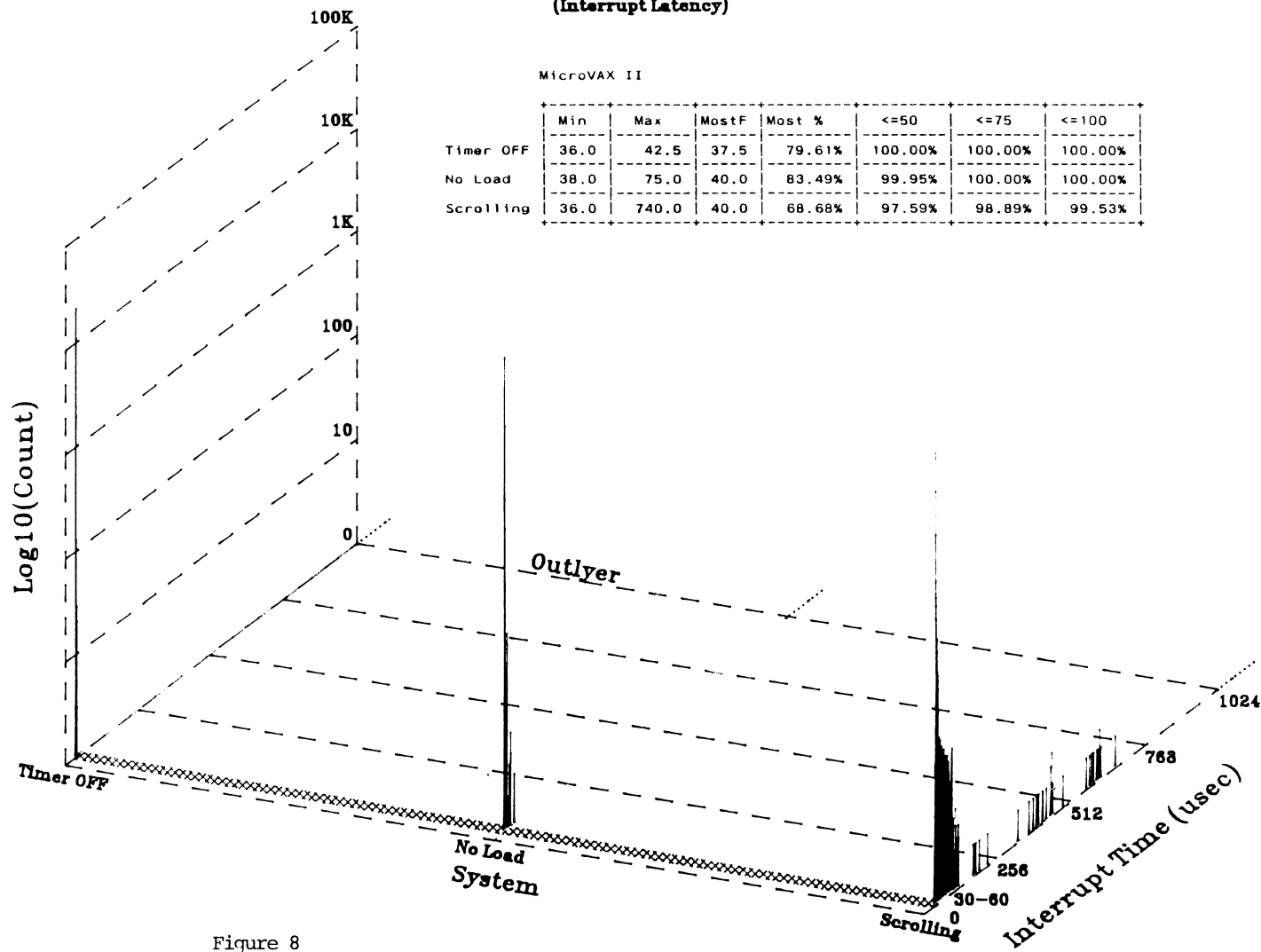
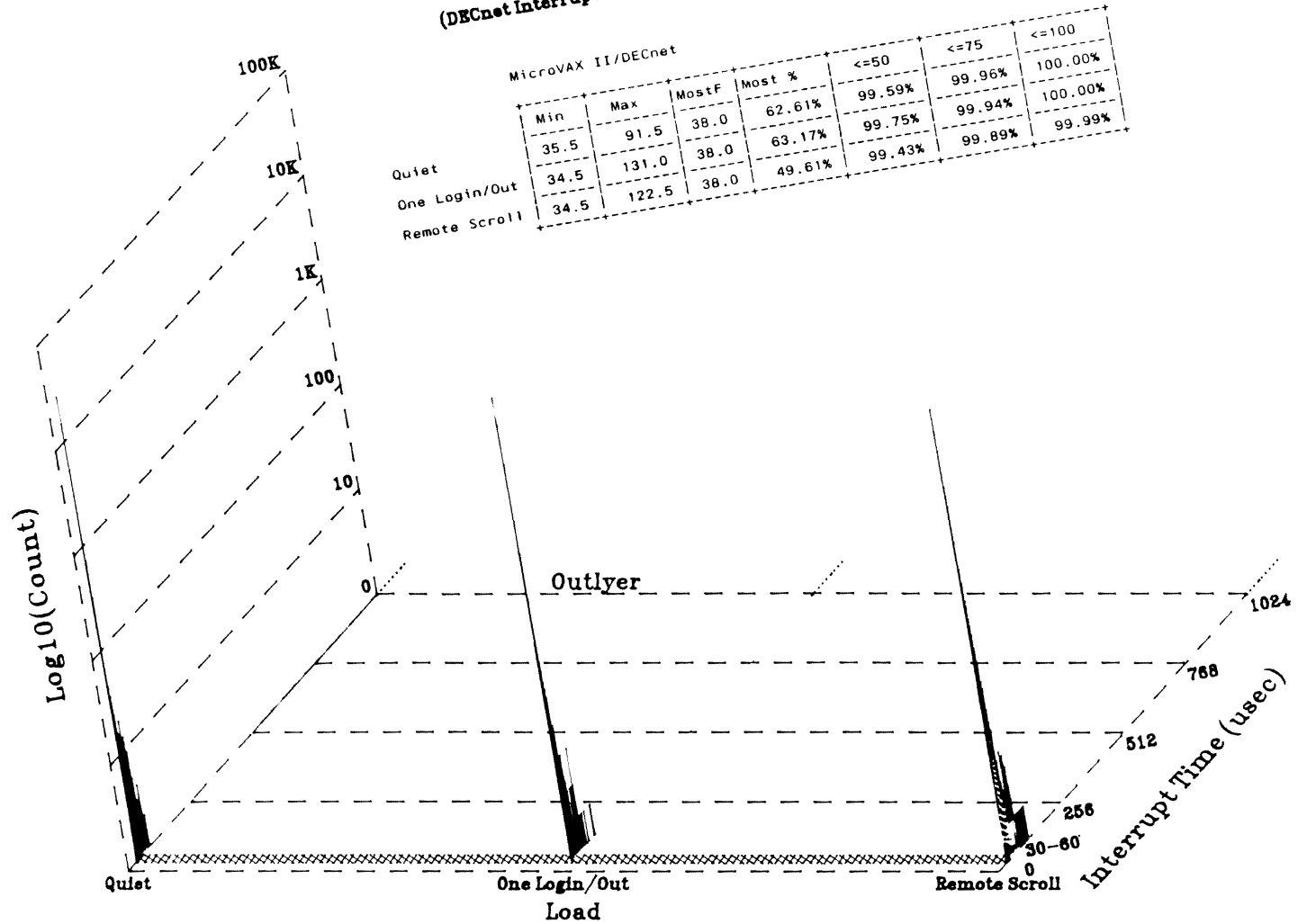


Figure 8

MicroVAX II (DECnet Interrupt Latency)



MicroVAX II/DECnet							
	Min	Max	MostF	Most %	<=50	<=75	<=100
Quiet	35.5	91.5	38.0	62.61%	99.59%	99.96%	100.00%
One Login/Out	34.5	131.0	38.0	63.17%	99.75%	99.94%	100.00%
Remote Scroll	34.5	122.5	38.0	49.61%	99.43%	99.89%	99.99%

Figure 9

VAXstation II/DECnet
(Interrupt Latency)

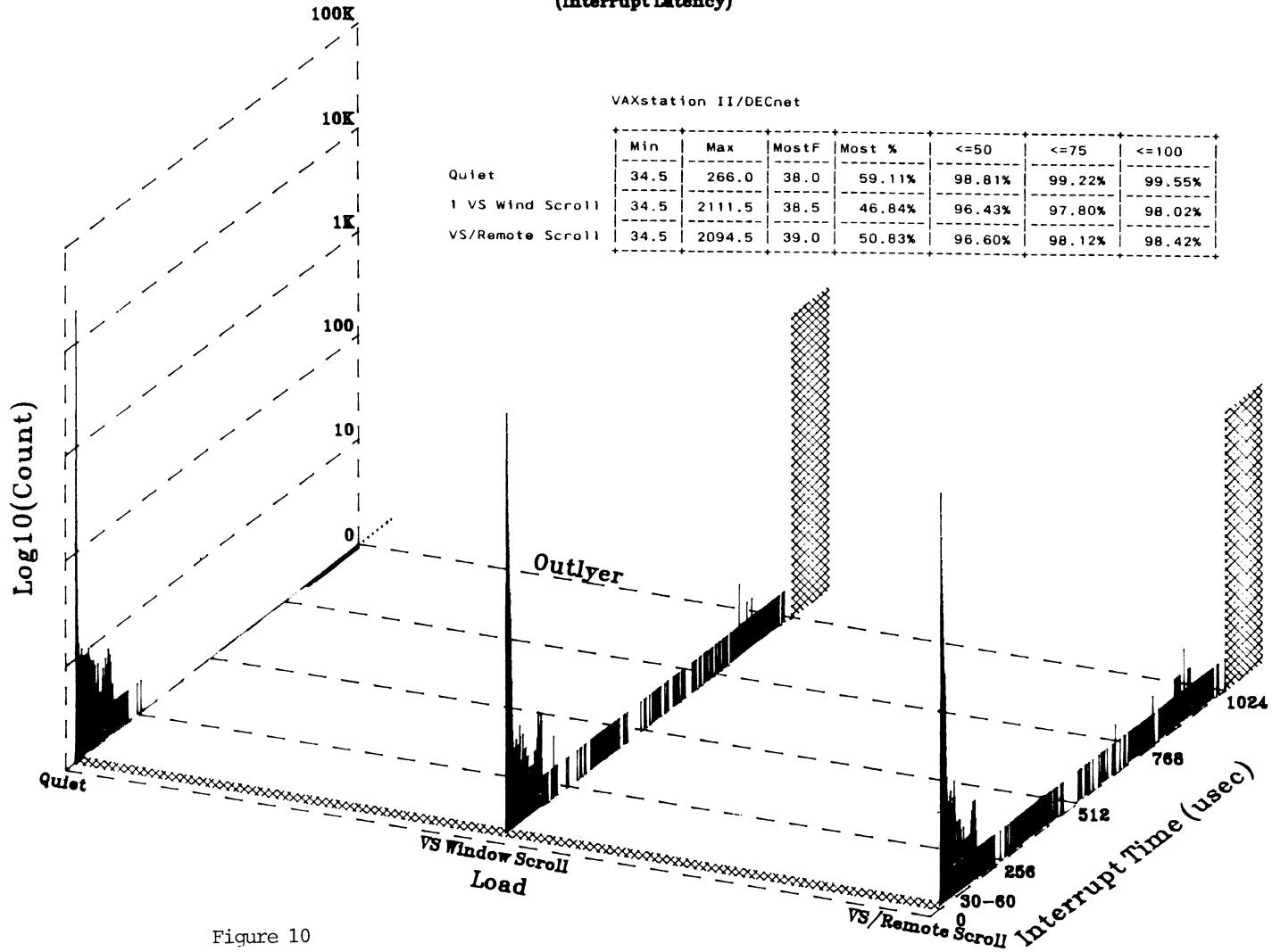


Figure 10

INTERFACING TO
DATA ACQUISITION SYSTEMS

George F. Sirois
Digital Equipment Corporation
Marlborough, Massachusetts

ABSTRACT

Some of the more common problems encountered by users when interfacing to Data Acquisition Systems (DAS's) - their symptoms and proposed solutions. Includes: static and dynamic problems due to MUX's and source resistance; ground loops; single-ended vs. differential input schemes; non-isolated vs. isolated differential inputs; and, driving cables.

INTRODUCTION

There are several problems commonly encountered by users when interfacing to a data acquisition system (DAS) that can significantly reduce the system accuracy or even cause permanent damage to the front-end.

MULTIPLEXER &
SOURCE RESISTANCE

Many problems are associated with the analog input multiplexer. Aside from the static scaling and offset errors that occur, the source output resistance also results in a dynamic error when the source is connected directly to an analog multiplexer (MUX).

SOURCE RESISTANCE

The static scaling error is due to the voltage divider effect between the source output resistance and the DAS's input resistance; the offset error is due to DAS input bias currents flowing through the source output resistance.

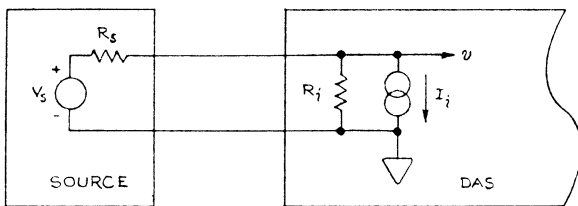


Fig. 1: Static input errors.

From fig. 1, the signal, v , that is presented to the DAS becomes:

$$v = V_s [R_i / (R_i + R_s)] - I_i [R_i R_s / (R_i + R_s)]$$

Thus, the scaling error is

$$\% \text{ Error} = -100 / (1 + R_i / R_s),$$

and, for $R_i \gg R_s$, the offset becomes

$$V_{os} = -I_i R_s.$$

As shown in table 1, R_s must be very small compared to R_i to maintain 12 bit accuracy.

Table 1: Scale Error

R_i/R_s	% Error
99	- 1.0 %
499	- 0.2 %
4999	- 0.02 %

The dynamic error results from MUX switch charge injection and MUX node charge transfer when a MUX channel is switched on, thus causing the input to become unsettled. The

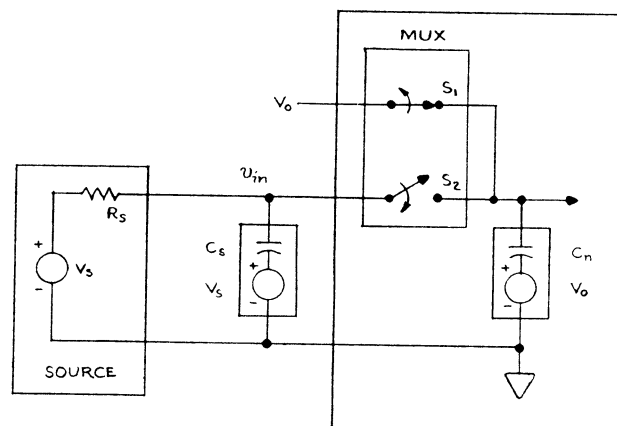


Fig. 2: MUX node charge transfer model.

MUX node charge transfer model is shown in fig. 2. The MUX node capacitance C_n has been charged through S_1 to V_o from the previously addressed channel. When the next channel is addressed its MUX switch, S_2 , will close (after S_1 has opened). This will cause the current input, which has been charged to the source voltage, V_s , and the MUX node to share a portion of their charges, thus unsettling the input. The input must now re-settle, through the source resistance R_s , to the source voltage, V_s . The resulting input transient waveform, given by

$$v_{in} = V_s + \frac{(V_o - V_s) C_n}{C_s + C_n} \text{Exp} \left[- \frac{(t - T)}{R_s (C_s + C_n)} \right] u(t - T)$$

is pictured in fig. 3. To simplify the analysis, it has been assumed that the switch on resistance is zero.

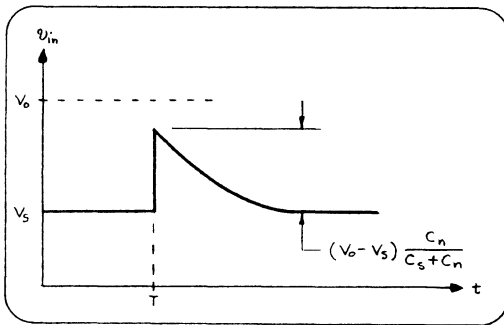


Fig. 3: MUX node charge transfer transient waveform.

A similar situation occurs due to the stray coupling capacitance between the gate of the MUX switch and its source and drain nodes. When the gate control signal turns the switch on, a charge is injected on to both the MUX input and its output node (fig.4), again unsettling the input. Many modern

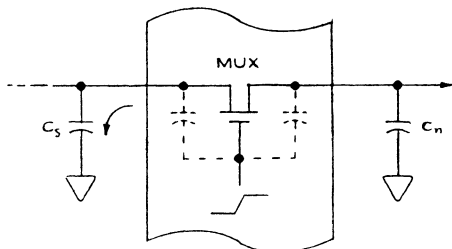


Fig. 4: MUX switch charge injection.

MUX's, however, are designed using complementary MOS switches to significantly reduce this injected charge.

A general solution to these problems is to buffer the source with a voltage follower configuration to present a lower source resistance. If, however, the static errors are not significant, an added delay to allow the input to re-settle (at the cost of a lower sample rate) or increasing the source capacitance to reduce the transient amplitude to less than an LSB (at a cost of lower signal bandwidth) will solve the dynamic problems.

OPEN CHANNELS

All unused inputs to the MUX should be grounded since addressing an open channel will cause the input bias currents to flow through an indeterminately large leakage resistance resulting in an input voltage that could saturate the front-end amplifiers. This, in turn, will cause a settling problem on the next addressed channel since op-amps typically require 10's of microseconds to recover from saturation.

OVER-VOLTAGE

Two related problems that can at the very least cause trouble and at the most cause damage to the input MUX are: power-off loading of the source and input over-voltage.

Power-off loading occurs when the power to the DAS is removed while the source is still active, causing junctions within the MUX to become forward biased. This can result in a large current being drawn from the source which may damage either the MUX or the source (or both).

A similar situation occurs when the input exceeds the MUX power supply levels during normal operation. However, even if there is an over voltage specification this often only indicates a "no damage" limit and may result in severe channel interaction when any input voltage is beyond the FSR limits, but within the maximum specified limits.

In the case of power-off loading, the user must either take steps to limit the source's output current or disconnect the sources before power is removed from the DAS. To prevent over-voltage from occurring during normal operation it may be necessary to also limit the source output voltage swing. Of course, if the DAS has been designed and specified to tolerate one or both of these conditions, then the life of the user is greatly simplified.

NOISE PROBLEMS

Noise is perhaps one of the most common and stubborn problems encountered in the use of analog systems. However, there are a few precautions that can minimize noise.

GROUND LOOPS

Ground loops can induce high noise levels as a result of current divider action of digital circuit return currents. From fig. 5

$$I_2 = (I + \Delta I) Z_1 / (Z_1 + Z_2),$$

where I is the digital circuit quiescent current, ΔI represents the digital switching currents, and where Z1 and Z2 are the wiring

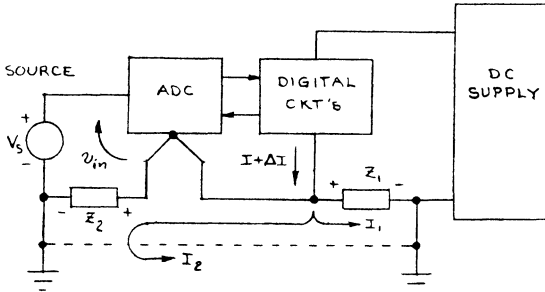


Fig. 5: Ground loop.

impedances. Thus, the front-end "sees":

$$v_{in} = V_s - (I + \Delta I) Z_1 Z_2 / (Z_1 + Z_2)$$

or,

$$v_{in} = V_s - \Delta I Z_1 Z_2 / (Z_1 + Z_2)$$

because $Z_1 || Z_2$ is significantly larger for ΔI due to the inductive components of both Z1 and Z2.

SHIELDING

Shielding the input wires can reduce externally coupled electrical noise. However, improper shield connections can result in common-mode noise pick-up due primarily to the stray coupling capacitance, C, from each input signal line to the cable shield, as shown in fig. 6. If the shield were to be connected to the analog ground return of the DAS, as in fig. 7, then the high frequency components of the common-mode voltage, V_{cm} , would have a closed circuit and

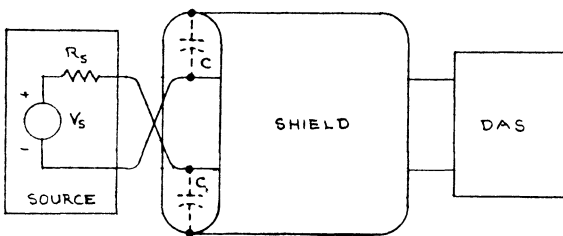


Fig. 6: Shielded twisted pair.

therefore create noise currents that would flow through both the source output resistance and the distributed cable series resistance and inductance.

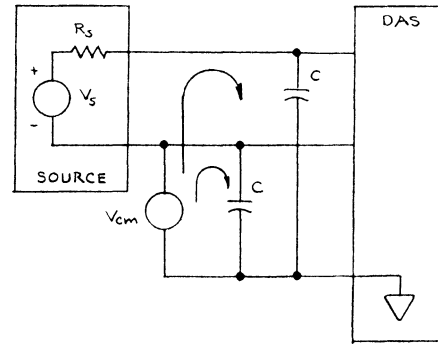


Fig. 7: Shield grounded at DAS.

Fig. 8, with the shield connected to the source's "low" side, is the preferred connection for the shield. In this configuration the common-mode voltage drives shield and, as a result, there is no longer a high frequency common-mode current path; the "low" side-to-shield capacitance has been shorted out, and the "high" side-to-shield capacitance now appears across the signal source. It should also be noted here that the currents required to drive the stray shield-to-ground capacitance from the common-mode voltage are shunted directly through the shield in this case, thus by-passing the source and the DAS input lines.

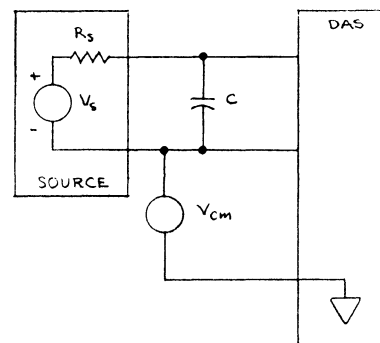


Fig. 8: Shield connected to "low" source end.

SINGLE-ENDED vs DIFFERENTIAL

The choice of single-ended vs differential is often critical, since a differential input will generally reduce common-mode noise. Because of cost, a pseudo-differential connection is often a reasonable compromise.

SINGLE-ENDED INPUTS

As shown in the ground loop example (fig. 5) a grounded single-ended source will result in higher noise. Not connecting the source "low" to the analog return isn't much better since the ADC must now "look" back through the digital return to "see" the analog input. The only "safe" source for single ended operation is a "floating" source.

PSEUDO-DIFFERENTIAL INPUTS

Pseudo-differential connections are essentially single-ended inputs with the AMP LOW input connected to the source ground, or "low" side, as in fig. 9. The input instrumentation amplifier now "sees" only the source signal and does not include noise source generated by digital return currents flowing through Z as it would if the amplifier were to be connected to the DAS analog return. Also, this scheme is general enough to accept a mix of grounded and floating sources. In such a case the floating sources would be grounded at one of the grounded sources.

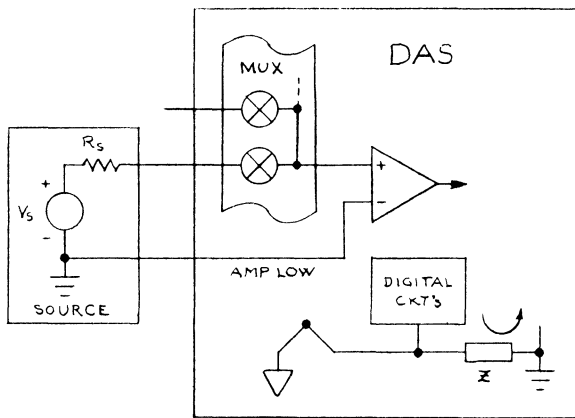


Fig. 9: Pseudo-differential input connection.

NON-ISOLATED DIFFERENTIAL INPUTS

Connections to non-isolated differential inputs are often not properly understood by users of data acquisition systems. A floating source cannot just be connected across this type of differential input. The input bias currents to the DAS require that such inputs be referenced to the DAS's analog return through a finite, and relatively low valued, resistance (R shown in fig. 10) to prevent large common-mode voltages from being generated that generally would saturate the front end. Since

$$V_{cm} = -R(I_1 + I_2),$$

V_{cm} will become very large if R is the leakage resistance to ground.

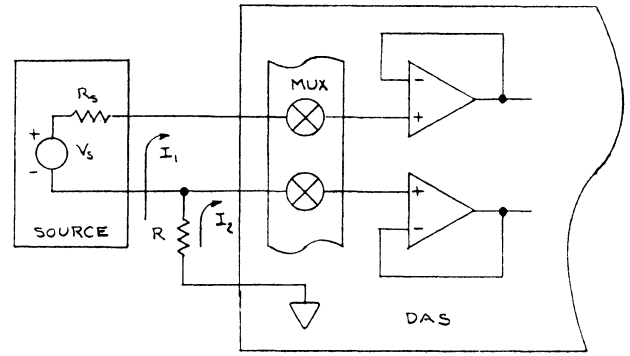


Fig. 10: Non-Isolated differential inputs.

ISOLATED DIFFERENTIAL INPUTS

Isolated differential inputs (fig. 11) are the most general purpose, but also the most expensive, connections. They will accept floating input sources without the need to reference to the system analog ground, as with the non-isolated differential case, and they will accept grounded sources without creating ground loops. They are, however, usually used where large common-mode voltage signals are present.

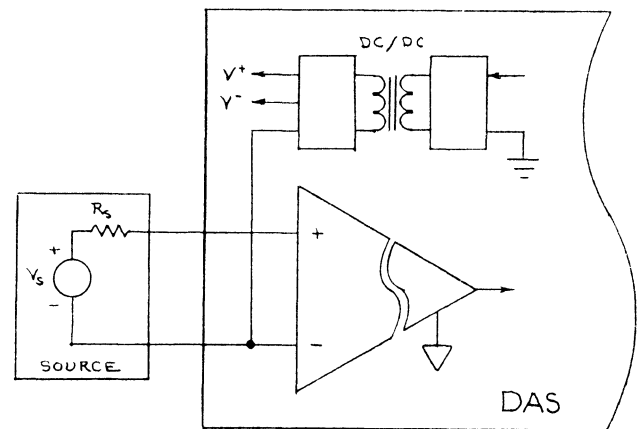


Fig. 11: Isolated Differential

DRIVING CABLES

Driving long cables with amplifiers requires some care on the part of the user to avoid instability and to maintain the expected dynamic performance. Since most cables can be expected to exhibit between 20pf to 40pf per foot of capacitance, a long cable can present a rather large reactive load to the driving amplifier. Many op-amps, especially the wide bandwidth types, are specified to drive only a few hundred pF. Even a low bandwidth type is limited to 1000 pF.

STABILITY

Cable capacitance can cause the driving amplifier to become unstable because of the pole created by the amplifier's output resistance, r , and the cable load capacitance, C (see fig. 12).

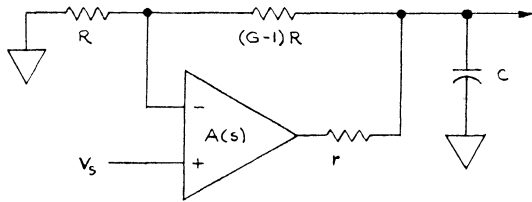


Fig. 12: Capacitive loading.

If $A(s)$ is assumed to be:

$$A(s) = A(0) / [1 + s A(0) / \omega_c]$$

and

$$G R \gg r$$

then the loop gain response will be as in fig. 13. The loop gain will roll off at a

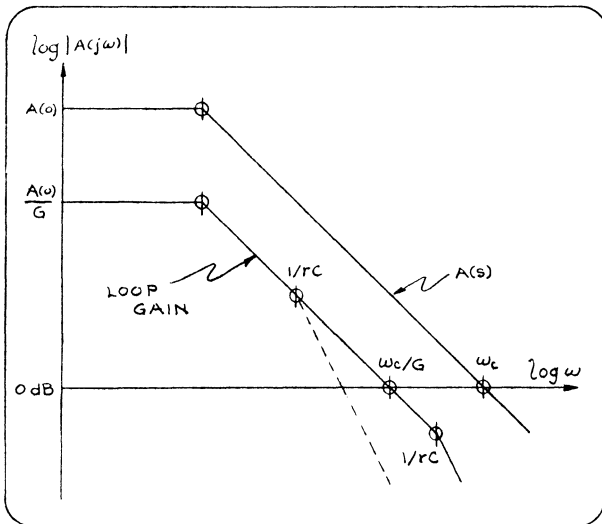


Fig. 13: Loop gain response.

20 dB/decade rate passing through unity when $1/rC > \omega_c/G$ and is, therefore, stable. However, when $1/rC < \omega_c/G$ the loop gain rolls off at 40 dB/decade through unity which results in the loop becoming unstable.

The load capacitance in fig. 14 has been decoupled from the amplifier output by R_o thus allowing the phase correction capacitor C_o to stabilize the loop. The requirement for stability is:

$$(G-1)R C_o > 2 (r + R_o) C$$

provided that $C_o \ll C$.

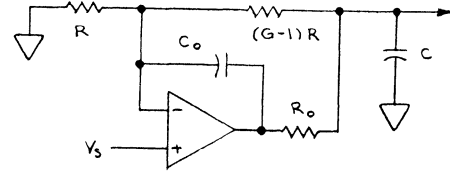


Fig. 14: Decoupled output.

SLEW RATE

Also, the apparent slew rate can be significantly lower than the amplifier's rated slew rate as a result of amplifier output current limiting. If the maximum amplifier output current is I_m , then the load capacitance charging rate is

$$de/dt = I_m/C.$$

When the amplifier slew rate SR is greater than this charging rate, the capacitance would require a current greater than I_m to maintain that slew rate. Therefore, the amplifier goes into current limiting and the apparent slew rate, SR' , becomes

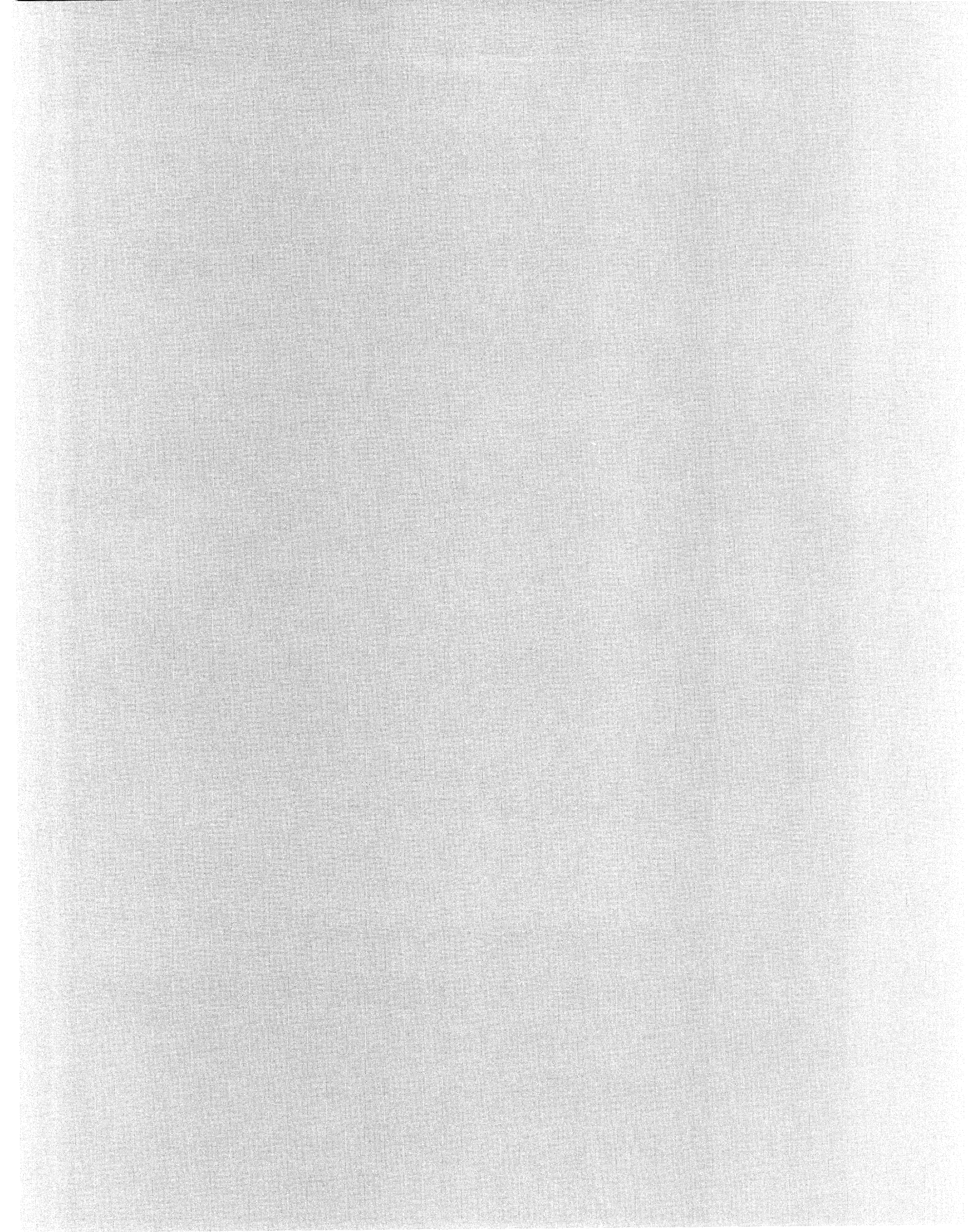
$$SR' = I_m/C$$

This effect can be eliminated by buffering the amplifier's output with a current buffer amplifier, connected within the feedback loop, to increase the maximum output current.

CONCLUSION

Although there are many potential problems for users to be aware of when interfacing to data acquisition systems, with care, most of them can be avoided or their effects reduced to acceptable levels.

DATA MANAGEMENT SIG



Encryption for Beginners

B. Z. Lederman
2572 E. 22nd St.
Brooklyn, N.Y. 11235-2504

Abstract

The purpose of this paper is to make people aware of what data encryption is, how it is used, who needs it, and why it is needed. It is intended as an introduction to the subject, so it will not go deeply into the mathematical internals of ciphers.

For many subjects, what something is and how it is used is often so intertwined that one needs to understand one before the other can be explained; so some very simple definitions will be given at first, and they will be elaborated upon later.

What is it?

Cryptography covers the general field of transmission of information which is protected from unauthorized access, and includes secret writing, codes, ciphers, and their use and defeat. Lately, encryption and decryption have come to be used in place of encipher and decipher to refer specifically to the use of ciphers to protect data, and will generally be used as such here.

Stated more simply, data encryption is a method of protecting data so that it can be accessed only by the people who are supposed to be able to get to it. This definition, while correct, is rather vague (it could apply equally well to the physical protection of data such as locking it up in a safe, or translating it into an obscure language): it does, however, explain the purpose of encryption, which is to limit the accessibility of selected items of information. This will be explained first, as it is desirable to understand why access should be limited to understand how it is to be done

Why is it used?

If you are working on a computer system which can be accessed by one or a very limited number of users, and which has no outside lines (no modems or dial-in lines), and which stores all information on easily removable media (floppy disks or tape cartridges), and you *always* remove this media and lock it in a safe when you are not using it, then you may not need encryption. If you can eliminate all access to your data other than by having the key or combination to the safe, and if nobody can look over your shoulder or otherwise tap into your computer or terminal lines while you are examining your data, then access to your information has been made about as secure as possible through physical means and encryption is probably not necessary. Unfortunately, this ideal state of affairs does not often exist. Sometimes your storage media cannot be kept in a safe, or you must store your information on a fixed disk which cannot be removed from your system, or you must share the system with many other users at the same time, or you must have dial-in lines so that people outside your physical location can access the same machine, or you must send information to other locations: in any of these cases, you may need to limit access to your information, and encryption is one method of doing this.

The immediate reaction many people have to this is: "Our computer is used only by people within our company. We don't have dial-in lines, [for our dial-in lines are secured by other methods, such as passwords or dialback], and all of our terminals are within our company area. Why do I have to protect my data?" Even in this situation, there may still be good reasons for using encryption.

Confidentiality.

First, you may have information which you are obliged to keep confidential. If you use your system to administer company medical benefits, for example, you may be obliged to keep medical records confidential. Without some sort of encryption or other protection scheme, it may be possible for many people in your company to peruse the medical records of other employees at will. Even if you are certain nobody will do this, increasing demand for rights to privacy of personnel records may set a legal requirement that you protect information from indiscriminate access. (Note that encryption will *not* protect against the persons who must still have access to the data: other checks are needed to insure that persons who must have the data will not misuse it.)

Next, there may be information you want to keep confidential. If you use your system to keep track of employee performance records, or calculate salaries as part of your budget planning, you might not want the employees involved to read or modify that data. It is all well and good to say you trust your employees, and probably most people can be trusted: but locks were invented to keep out the (small?) percentage of society which cannot be trusted. I rather imagine that most people reading this paper have locked their houses and cars before leaving them, even if they trust most of their neighbors: if you would do that, then you probably have information which should also be "locked up". Similarly, you might be preparing information for contracts, order placements, payroll records, competitive bids, and similar information which could represent a significant portion of your company's assets, and might be several times the annual salary of many of the people who have access to it (and they are not always only the people whom you think have access to it). The more important an item of information is, the more likely it is that someone could benefit by getting it, and therefore the need to protect it increases directly with its importance.

Unauthorized access.

The case where a "hacker" or other unauthorized person calls into a computer system and proceeds to cause various type of mischief and/or damage is one that probably most people fear. You may have a system where it is necessary to have dial-in access for your own personnel, and it then becomes necessary to guard the system as much as possible. There are various methods of limiting access to a system through passwords, or through hardware, which are outside the scope of this paper. Data encryption can act as a second line of defense, however, and should also be considered. In many cases, "hackers" are simply looking for files they can read, or programs they can run: encryption can put data in a form where it cannot be read, and programs into a state where they cannot be run, and thus defeat two of the hackers main goals. Encryption will *not* prevent the random modification of data (where the modifier doesn't care what the change actually does) or deletion of files: other methods of protection are required to guard against that type of damage

Protection on "outside" systems.

The situation may also be reversed, as many computer users do not own their own systems and have to use time-sharing or other outside computer processing to store data and provide other computer services. In this case, you may have little control over who in the world has access to your data. An encryption scheme that can be implemented on your own data on the outside machine could be one way of protecting your information. Similarly, many companies store copies of their records in outside warehouses or other storage facilities to protect against fire or earthquake damage at their main location, and while such facilities usually offer guarantees against unauthorized access, some extra protection might be desirable.

Protecting data during transmission.

One last situation which occurs to many people is when data has to be transmitted from one location to another, usually over some public facility (telephone, telex, leased communication line, air freight, or mail). It is actually more likely that the data will be accessed from within your company than from without (intercepting telephone channels from microwave links is possible, but rather difficult), but the more valuable the information is,

the more likely it is that someone will try, and it wouldn't hurt to take some reasonable precautions. If you are engaged in any type of electronic funds transfer, such as depositing your employees payroll directly to their bank accounts, or transfer of company assets to your bank or to other companies, the sums of money involved may be so great that not encrypting the data in some way is courting disaster.

Always consider the worst case situation.

In deciding if encryption is needed, you should consider what would happen if someone were to change your records just once: if damaging or losing an important piece of information would seriously hamper your business, or cost you a significant amount of money (either by direct loss, or delay, or the effort to replace the missing information, or loss of goodwill of the person/company at the other end), then you should consider encrypting your data. Remember that the true cost of data might not be just what it cost you to obtain it, but also what it will cost if you lose it, or what it will cost to replace it.

Other protection methods.

It can be seen, therefore, that many users will have some use for a data protection scheme of some kind, as nearly everyone has some type of information which is not to be accessed by everyone else. This leads to the methods which can be used to protect information. Various computer operating systems are in use today, some of which include access protection through requiring users to log into accounts, or various methods of verifying that persons accessing dial-in lines are properly authorized, or through protection codes within the storage system (such as the file protection codes used in the RSX, RSTS, and VMS operating systems). These are outside the range of this paper, but it will be mentioned that they don't always provide the limit of protection needed, either because there has to be at least one privileged user of the system who can bypass the checks, or because backup copies of the data must be stored off of the machine, or from other limitations of the system. Even when such schemes work well, they may not be enough, and they don't work at all if the information has to be sent outside (by wire or mail, etc.). This leads us back to data encryption, which will allow the information to be protected by a method which is independent of any protection which may be provided by the operating system. This does not mean that other protection schemes should not be

used, or that encryption is the answer to everything, either: different protection schemes cover different areas, and usually complement rather than substitute for each other.

Once the need for some type of data protection is recognized, a protection scheme must be selected. As previously mentioned, cryptography covers, in general, secret writings, codes, and ciphers.

Secret Writing.

Secret writing covers such things as invisible inks, micro-photographs, and concealing messages within other messages. This is a highly specialized field, and one which is not likely to have much general application: it is usually too cumbersome for easy use, and is not applicable to storage of large amounts of information on computer media. Just to show what it is like, consider the message:

"Inspect details for Trigleth, acknowledge the bonds from Fewell."

which doesn't seem to mean anything. If you take the third letter of each word, however, you get the message "Strike Now". This is an example of secret writing, (a method which follows a fixed formula like this may also be called a concealment cipher), and it can be seen that it would not be easy to use: if it had no other faults, the concealed message has become over 6 times the length of the original ("clear") message, and if you have to pay for disk storage space or transmission costs, you can see a big disadvantage to this type of protection. Invisible inks can be used on paper messages, but obviously won't work at all on data stored on disk or magnetic tape. (There was one fictional story where a message was written on a reel of tape with a grease pencil, but this tends to gum up the tape drive, and isn't very practical.) They can be useful to authenticate documents, as they cannot be duplicated by photocopying machines, but again, this is a field where expert assistance from a printing company or ink manufacturer is required. The one and only advantage to secret writing is that many countries are implementing restrictions on trans-border data transmission: even though they encrypt their data, they won't let you encrypt your data, so they can monitor your transmissions: a good method of secret writing might evade this restriction because the essential feature of secret writing is that it does

not appear to be conveying anything but the obvious innocent message, but most methods are too cumbersome to be practical. We will not give any more attention to this subject.

Codes.

A code is the arbitrary mapping of one set of symbols to another set: it is usually one to one, but can be one to many or many to one. One example of a code which is in very common use every day is ASCII, the American Standard Code for Information Interchange, used by most computer terminals to map binary signals to numbers, letters, and other characters; a portion of which is shown here.

040 SPA	060 0	100 @	120 P
041 !	061 1	101 A	121 Q
042 "	062 2	102 B	122 R
043 #	063 3	103 C	123 S
044 \$	064 4	104 D	124 T
045 %	065 5	105 E	125 U
046 &	066 6	106 F	126 V
047 ^	067 7	107 G	127 W
050 (070 8	110 H	130 X
051)	071 9	111 I	131 Y
052 *	072 :	112 J	132 Z
053 +	073 ;	113 K	133 [
054 ,	074 <	114 L	134 \
055 -	075 =	115 M	135]
056 .	076 >	116 N	136 ^
057 /	077 ?	117 O	137 _

This isn't usually thought of as a code, and it certainly isn't a secret, but it is a code: it transforms one type of data into another through an arbitrary mapping. Note that the mapping is indeed arbitrary, even though the letters follow the alphabet for convenience: there is no reason why they would have to do so for the code to work.

Another code which better fits the general public's perception of a code is the type of code which has been used for telegrams, a portion of which is reproduced here:

MUWUB Improving rapidly
 MUXAW Improving slowly
 MUXEX Is not improving as I wish
 MUXIZ Is there any change

MUXNO Is there any improvement
 MUXPU Progressing satisfactorily
 MUXRY Sorry to hear you are ill

MYGEL How would
 MYGIM HURRY (See Haste)
 MYGON HYPOTHECATE-D
 MYHAL IF
 MYHCI And if
 NYHDO And if not

and so on. It can be seen that the mapping between the original phrase (the "clear" or "plain" text) on the right and the code word on the left is completely arbitrary, and that the book is the only way to go from one to the other. This particular code had the advantage that in most cases the coded text was much shorter than the original message: two groups of five letters could be pushed together to make one 10 letter group, which was counted as only one word in the cost of sending the telegram. Since the mapping is arbitrary, codes can be very secure. Generally, you have to have the arbitrary mapping in order to defeat (or "break") the code, though if the code is re-used often enough, the mapping can sometimes be deduced. They are also vulnerable if one can obtain a copy of the plain text and the coded text which goes with it, and of course are defeated if the wrong person obtains a copy of the code book. Some authorities consider book codes like this that are used once only to be completely unbreakable, and it would be easy to use a computer to generate lists of arbitrary code words to use.

Codes do have many disadvantages in the computer environment, however. A computer program to automatically code a message with a scheme like the example would be very complex, as the context of the entire message (or a fair portion of it) is needed to search through the list of phrases and find the appropriate code word: decoding the message by looking up the letter group would be easier. Encoding large strings of numbers through code words is tedious and likely to increase the size of the message, and there is always the problem of what to do if you need a phrase which is not already defined in the code book. Binary data cannot be coded at all with this particular scheme, and would be difficult to encode with most schemes. Codes are most likely to be of use where a set of messages or phrases are going to be sent, and where that set is not too large and can be well defined before they will be used. Since we would like a method which would work on a computer and accommodate a wide variety of data with a minimum of human intervention, we will not consider codes further.

Ciphers.

A cipher is a method of transforming data from one form to another through a logical process, usually with a geometric or mathematical basis. Since a cipher is a method or system rather than a group of arbitrary mappings, it should be possible to transform any "plain" or "clear" text, regardless of length or content, into a single enciphered message. This is more easily understood with an example, such as a simple geometrical cipher. I will take the familiar phrase,

"THE QUICK BROWN FOX JUMPS OVER
THE LAZY DOGS BACK"

and write it out in a square in the usual fashion, left to right, top to bottom.

```
T H E   Q U I  
C K   B R O W  
N   F O X   J  
U M P S   O V  
E R   T H E  
L A Z Y   D O  
G S   B A C K
```

To encipher this message, I can take the letters out by some sequence other than the way they went in: for example, top to bottom, right to left. (This is an example of a transposition cipher, as it works by transposing or changing the order of the letters in the message, but not the letters themselves.) This will give me:

"IWJV OKUO OEDCQRX H A BOSTYBE
FP Z HK MRASTCNUELG"

which doesn't look anything like the original. The underlying principle here is that there is a definite method of transformation between the original text and the enciphered text without considering the actual content (even if it is not obvious on a cursory inspection), whereas in a code the transformation was completely arbitrary and very sensitive to content. Because ciphers work on a method of translating data from one form to another, they are generally much easier to implement on a computer, and they are generally much less data sensitive than codes would be. In this example, each

character could easily be a byte or word of binary data, and the scheme would work just as well: this makes it suitable for use on a computer, and transmission of data without having to know what the data will be before use. (Though outside the scope of this paper, it may be noted that if a voice signal is converted to binary data, it can also be enciphered to protect against unauthorized reception. This is the basis for most modern voice "scramblers", though there are other methods available.)

There are a great many types of ciphers, some more secure than others, and some easier to use than others. One which is very common, and even occurs in some daily newspapers, is a simple letter substitution, where one letter is replaced by another. For example,

ABCDEFGHIJKLMNOPQRSTUVWXYZ

can be replaced with

EFGHIJKLMNOPQRSTUVWXYZABCD

This is a substitution cipher, which changes the letters in the message, but not their order in the message. This would make the sample phrase "THE QUICK BROWN ..." come out to be:

"ZLI UYMG O FVSAR JSB NYQ TW
SZIV ZLI PEDC HSKW FEGO"

Since this is a one to one mapping, I am going to leave it to the purists to determine if it is a code or a cipher, though it is content insensitive (there is obviously some overlap between some codes and ciphers). The drawback to a simple cipher like this is that it is too easy to break with just a pencil and paper, and with even the least expensive home computer it is literally child's play. (You can read *The Gold Bug* by Edgar Allan Poe or *The Adventure of the Dancing Men* by Sir Arthur Conan Doyle to find out how.) Many other, more sophisticated, transposition and substitution ciphers than the ones demonstrated here have been invented and used in the past few centuries, but since they were all implemented by hand they are all too easy to break by modern methods. You can purchase a number of books that will tell you exactly how to do it with

no more equipment than pencil and paper (and patience), and the proliferation of home computers makes most of them very simple to break indeed. They may still be adequate for some purposes however, but considering how good a cipher needs to be will be discussed later.

Modern Ciphers

If secret writing is too cumbersome, codes are too data sensitive or limited, and existing ciphers are too easy to defeat with computers, then what is left? The answer is that most modern encryption schemes are based on the same principles as older ciphers, but use the power of the computer to expand the magnitude of the encryption scheme. For example, in the transposition cipher shown, the box was 7 letters on a side: it could be made larger, but when encryption is done by hand a box much larger than 15 or so on a side becomes too cumbersome to use. With a computer, however, there is no limit to the size of the box: simply increasing the box to 100 per side makes it too large to "break" the cipher by hand. This scheme of using the computer to expand on a good encryption method can be used to create ciphers that are difficult to defeat, even with another computer (the box cipher would still be too easy to break by computer and is given only to illustrate the idea). One which I have used is a variation on the periodic number substitution (also known as an addition or Vigenere) cipher. In this scheme, a number sequence is added to the text: a simple example would be to add the sequence

13571357135713571357135713571357

to the numeric value of the ASCII characters in the message

THE QUICK BROWN FOX JUMPS OVER
THE LAZY DOGS BACK

to get this:

UKJ`RXNJL#GYPZS`GR]`KXRWT#T]FU%[
IH%SB]^`ERLZ!EFJL

With a number sequence this short, the cipher would not be too secure (you can see even in this short message that a SPACE becomes a ` four times, and the

sequence "SPACE-something-U" has twice been changed to "-something-X") though it is more secure than the simple substitution cipher shown before. Various methods of obtaining a less repetitive sequence have been tried in the past, but usually produce no real increase in security. Using the computer, however, a number sequence can be generated that appears to be random, and is thousands of digits long. Most computer languages have a random number generator (or more accurately, a pseudo-random number generator, as the sequence can be repeated exactly when desired), such as:

LET A = RND(B) in BASIC, and

A = RAN(B) in FORTRAN,

and similarly for other languages. There are theoretically an infinite number of such pseudo-random sequences, and even for a specific generator there are a very large number of specific sequences: in DEC's FORTRAN-77, the number that starts the sequence (the variable B in the above example) can have at least two billion possible values. This particular cipher is sometimes called the Fast "Infinite-Key" method, and has been widely used with good results. We could repeat the above procedure by generating a pseudo-random number sequence such as:

198683392515385726581534169734718

and adding it to

THE QUICK BROWN FOX JUMPS OVER
THE LAZY DOGS BACK

to obtain

UQM_YXLLM%CWR_S`HU](KZPTT&X]HV`U
PH`UJ_a`JULZ)JHGM

At first glance, this doesn't appear significantly different from the first example, but if someone were to attempt to defeat the cipher by the usual method of looking for repetitive patterns and common adjacent letters, they wouldn't find any, and would not be able to defeat the cipher. This cipher has

the additional advantage over the "box" cipher in that the characters can be processed in the order they are read: in the box cipher, a large portion of the message has to be read in and stored before any of it can be processed. In most computer ciphers, it is an advantage to be able to process the message serially, and to not have the length of the message have any effect on the encryption scheme itself, especially when the messages being processed are being transmitted from one place to another (over a communications line or to a disk or tape drive are two examples).

It can be seen, therefore, that even though the computer has made it easier to defeat some encryption schemes, the power of the computer can also be used to raise the complexity of a cipher to the point where it is very difficult to defeat, even with another computer. This is the basic principle behind most good modern computer ciphers: the use of the computer to make the cipher so complex that it is (hopefully) beyond the ability to defeat by any practical means.

Data Compression?

It was mentioned that the telegraph code example shown earlier also compressed the information into a more compact form. There are a number of data compression schemes in use to minimize the amount of space data occupies when stored, or to reduce the amount of time needed to transmit information from one location to another (and hence reduce the cost of transmission). Some of these compression schemes could also be thought of as ciphers, as they transform data from one form to another. While they have the obvious advantage of compressing the data, generally the compression algorithms are too well known to provide any security.

With some understanding of what encryption is, we can perhaps present a better definition. One such definition could be:

"Encryption is a method of transforming data into a state where it is not easily available to persons other than those for whom it is intended (using ciphers)."

This is a very general definition, and it does appear to be somewhat cumbersome, but it is worded in this way deliberately. Note especially the emphasis of the phrase, "not easily available". Generally, no

encryption scheme is absolutely secure from ever being defeated, and a decision has to be made as to how good a scheme is needed, or how much security has to be obtained and at what cost. From a practical standpoint, the real purpose of encryption can be defined as this:

"The goal of encryption is to make obtaining the data more expensive than the data itself is worth."

(Where expense is counted in time, effort expended, cost of labor, cost of computer services, etc.)

While this definition may not precisely define a cipher, it does clearly define the goal encryption should achieve.

To evaluate a potential encryption scheme, one must consider from whom the data is being protected. Some possibilities are:

1. Curious employees
2. "Hackers"
3. Outside visitors
4. Service personnel and/or vendors
5. Competitors
6. The Criminal Element (internal or external)
7. The IRS
8. The "spooks" (CIA, NSA, KGB, MI5, etc.)

among others. The first four can probably be discouraged with even a very simple cipher: as mentioned before, most "hackers" and other idle curious are simply looking for files that can be read or run. If they were to see a file such as this:

```
RTP $%& &.2H8I ] ).4HHQPPJ8IKNUIOQPP
RUP $%& &.3H8I ],%.H342DH).4H8III
RVP 2%-
SQP 02).4 B#/-054!4)/. /& -/2'!'% 0!9-%.43B
SRP 02).4
SUP 02).4 BO,%!3% ).054 4(&% 02).#)0!, H7)4(
SVP ).054 0
SWP 02).4 B).054 4(% !..5!, ).4%2%34 2!4% H). EIB[
SXP ).054 )
SYP 02).4 B).054 4(% 4%2- H). 9%!23IB[
TPP ).054 4
TQP 02).4
TSP 4]4JQR
TUP 1])
TVP )])OQREP
UPP -]&.2HOJ)OHQMQOHQK)I>4II
UTP 02).4 BO2).#)0!,B[4!"HSUI[BDB[0
UUP 02).4 B).4%2%34 2!4%B[4!"HSTI[1[BEB
UVP 02).4 B4%2-B[4!"HSTI[4[4!"HTPI[B-/.4(3B
UWP 02).4 B-/.4(,9 0!9-%.4B[4!"HSUI[BDB[4!"HSXM&.3
```

they might well pass it by, or maybe make a few simple attempts to read the file as if it was binary data. But if anyone should happen to figure out or guess that it is really a BASIC program (and at every previous presentation of this paper there have been attendees who have immediately recognized this), then it would not take long to decipher it as it happens to be encrypted with a simple letter substitution cipher. Since a computer is going to do the work, it would be just as easy to use a more secure cipher, and one which will transform the data into something which will not look like obviously encrypted data when examined. For example, the "Infinite-Key" method takes no more computer time or disk space than simple substitution, is very much more secure, and the resulting data doesn't look at all like text, so there is no reason to use the simple substitution when superior methods are easily available.

If interception of data by a competitor, or by a dishonest employee (which is really the greatest threat) is a serious consideration, then you will probably want the most secure cipher that can be reasonably implemented (one which protects the data well, but will not use up so great an amount of computer resource that it becomes more expensive than the data it is protecting).

If you intend to protect your data from categories 2, 3 and 4, then other protection schemes should be your first choice, such as not allowing outside visitors to wander un-escorted about your plant, removing your data from the system before allowing it to be serviced by outside personnel, and using various protection schemes to prevent unauthorized dial-in access. Encryption of data can act as a second line of defense in these cases, however, and should still be considered: it must be stated again, however, that encryption is not necessarily the best solution to every situation, and that all methods of protecting data need to be evaluated to determine what best suits a given need.

Against the last two categories: you have to be realistic, and understand that any government agency that can put the gross national product of a major world power into it's efforts is going to be able to break any cipher you could use. That doesn't mean you have to make things easy for them, and there are ciphers available now which are very difficult for anyone to defeat, but you must remember that no cipher is absolutely unbreakable.

How Good is Good Enough?

It was stated that a good encryption scheme costs more to defeat than the information is worth. This means that the cost of the labor expended and computer resource dedicated to the task are more than the ultimate value received from the information which may be obtained. For example, breaking the Infinite-Key and DES ciphers is generally expected to require a "brute force" approach: trying every possible key, and looking at the result to see if it makes sense, and even this may not work. Even if someone is willing to dedicate a computer to the task, it could take months or even years of effort to break one message, by which time the information may be useless. In addition, the time a computer spends on breaking the code cannot be used for anything else, like doing payroll or inventory or other normal business functions. If you are preparing bids on a contract which will yield, say, \$10,000 and a competitor tries to steal your information and under-bid you, then your encryption scheme is successful if it either takes so long to break cipher that the competitor can't meet the deadline for submitting bids, or if it costs the competitor more in computer resources than the \$10,000 or so that the contract would yield: even though the cipher is broken, the person who broke it comes out with a net loss. Few "hackers" are going to have the patience to let their home computer run for several months or years to decrypt one message and not use the computer for anything else, and not much information is so valuable that it would be worth while renting a Cyber or Cray super-computer for several months to break the message relatively quickly (unless you are a government agency, and can do whatever you like). The situation is similar to the physical protection of property: you can't really make any building completely burglar proof, but you can make it difficult enough not to be worth the effort to break in, or to make it easier to break in somewhere else.

It is possible that someone within a company might use the company computer to try to break a cipher by brute force, reasoning that the computer time doesn't cost them anything. Since defeating a good encryption scheme would use up relatively large amounts of computer time over an extended period, it should be possible to detect if anyone within a company is using the computer system in this manner, and deal with the problem directly.

Other necessary precautions.

A consideration which is equally important as the selection of an encryption scheme is keeping the keys themselves secure. Just as it would do no good to buy the most expensive lock and lock your house if you then put the key under the door mat, it does little good to encrypt your data if anyone can get the key. In terms of internal security, this often means correct selection of a key to use: since most modern ciphers use a number as the key, there is a great temptation to use an easily remembered number such as your telephone number, birth date, social security number, wedding anniversary, or some such number as a key. When a word or phrase is used, as is often used for computer passwords, then there is a great temptation to use names of friends, spouses, pets, children, hobby interests, etc. Unfortunately, any number or phrase that you can remember easily will also be easy to guess for anyone who knows you. If you are trying to protect data internally in your company, using such a number would defeat the best cipher: rather than having to try several billion possible keys to break a cipher by brute force, the number of attempts are reduced to a few dozen or so and trying them all becomes quite practical. This leads to the paradox that you must choose a number you can remember (or you may never get your data back if you forget the key), but one which no one else is likely to guess; or else you have to write the number down, but in a place where no one is likely to get it. The latter scheme is probably better than trusting to memory, but you should not keep important numbers laying about: keep them in your wallet (and keep your wallet with you), or in some other secure place. Similarly, don't put them in the telephone directory or card file that sits on top of your desk, or in other easily accessed places. It is also a good idea to change the keys periodically, especially if it is being used for data transmitted externally. (Internally, the threat is greater that someone will figure out your key, or may see you type in the key, or be able to compare the encrypted data with the "clear" data, and deduce the key that way). Basically, you must use at least as much caution in dealing with cipher keys as you would use in handing out door keys to your plant, or electronic lock keys to your personnel: they all protect your assets, and have to be treated with the same respect. While you can hire guards for physical security in a plant, you cannot do the same for information in a file or transmitted over a wire, and information is easier to move than equipment; so if anything, the cipher keys must be kept even more secure than other kinds of keys.

Public Keys.

When data has to be transferred from one location to another, then the risk is doubled, as the key has to be kept in two places. One absolute rule is that you never, ever, transmit the key with the data it protects (you might just as well not bother encrypting at all). It is usually a good idea to use an encrypted transmission to send the next key to be used at one time, and the data at some other time, and that both parties must exercise the same caution in protecting the keys. Otherwise, you must use some secure method of transmitting the keys to the locations where they will be used (such as sending someone you can trust to carry them by hand), and storing them in a safe or other secure location. One partial solution to the problem is the Public Key method of selecting keys. This is not an encryption scheme, but is a method where two people can create a large numeric key by each selecting a number which forms half of the key, and where each party knows only half of each key. (Please refer to figure 1.) The advantage of this method is that one half of the key

can be made public, and anyone can use it to encipher a message intended for you, but only you can decipher the message using the other half of the key which was kept secret. This can also be used for source verification if both halves are kept secret: for you to be able to decipher the message, it will have to have been enciphered using the matching half of the key, and this is "proof" that the message came from the correct source. The method is based on the fact that it is difficult to factor a very large number which is the product of two very large prime numbers (each party picks one of the large primes): lately, there have been some announcements that it might not be as difficult to factor large prime numbers as was formally thought, but it may still be useful to many people. If you are transferring data within an organization and can keep the key secret at both ends, then Public Key isn't necessary: it's primary use is where the security of the key at one end isn't known, or must be made public.

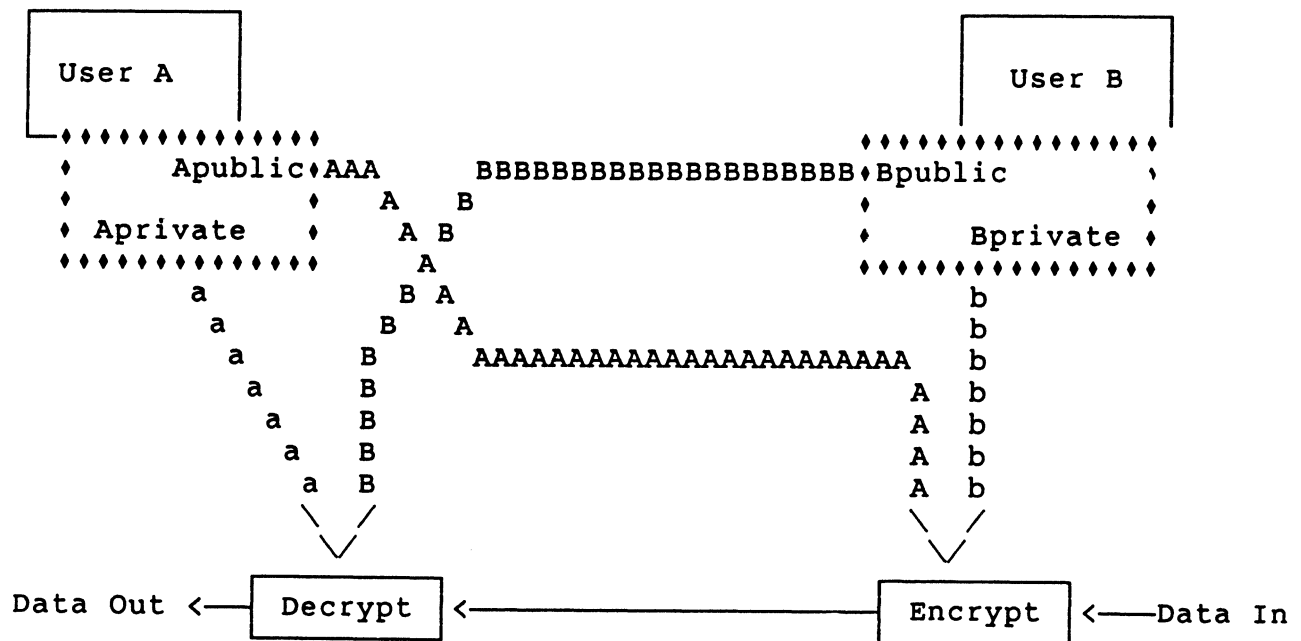


Figure 1 : Public Key Encryption

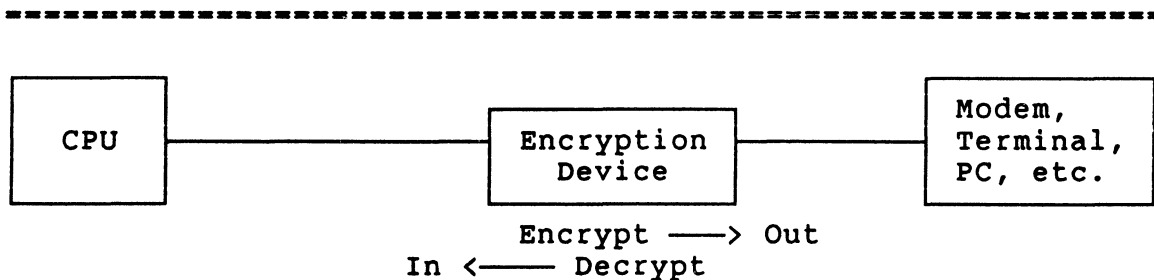
Hardware protection and DES.

So far, we have considered encrypting data while it is in the computer system, and before it is stored or transmitted. This is not the only way it can be done: it is also possible to attach a device to a communications line so that information passing through it is encrypted in one direction and decrypted in the other direction. (Please refer to figure 2.) For example, the device could be attached between your computer and a modem, so that "clear" information being transmitted from your computer will be encrypted before it goes into the modem and out into the world. Most of the special hardware currently offered for sale for this purpose use the Data Encryption Standard (DES), also called the Data Encryption Algorithm (DEA). This method of encryption was developed by the National Bureau of Standards to provide a standard, secure encryption method, and it involves many stages of transposition and substitution. Furthermore, there are several modes for data to pass through the encryption scheme: the method any individual will use depends upon the application. According to the developers, the DEA is intended for use only with hardware encryption schemes for several reasons, two of which are security of operation and verification of correctness.

The first reason includes protecting the key and the encryption method: if it is in special hardware, you have to enter the key into that piece of hardware, and it won't be "floating around" your computer system as it might be if a software program was used. Similarly, only the manager in charge of the special hardware knows what the key is: you don't have individual users losing their keys (or giving them away). In addition, there are often ways for one user to monitor another user's program on the same computer (for example, to watch someone type in their key), and it was felt that it would be more difficult to tap into a separate piece of hardware. With the protection in hardware there is the additional advantage that no-one can forget to encrypt data before sending it out: anything which is transmitted on that line is automatically encrypted. It was stated before that encryption might not prevent "hackers" or other unauthorized persons from accessing a system, but the one exception is if there is a hardware encryption device placed between the system and the modem which always encrypts the data on that line. Encryption would then prevent unauthorized access, as anyone who wishes to dial in on that line must have an



Normal connection without Encryption



Connection with Hardware Encryption Device

Figure 2 : DES Hardware

encryption device which uses the same cipher and key. In a similar manner, a hardware device can be placed between a computer and a peripheral device: for example, a disk. If this is done, then all data on the disk is automatically encrypted, and you don't have to worry about users forgetting to encrypt sensitive data, or service personnel reading it during maintenance.

The second reason, that it would be easier to test if the hardware is working correctly than to test if a program is working correctly, is a reason with which I do not entirely agree. It also means that the use of DES would be limited to those applications that can send the data through a line to the special hardware, and that you would have to buy the special hardware for every location which wanted to encrypt data: this meant that locations with personal or small business computers had to buy an encryption device that was as large and as expensive as the computer itself. This is changing rapidly as more large scale integrated circuits which implement the DES are being placed on the market, so that the cost of a peripheral device that does encryption in hardware is decreasing, but it still has many drawbacks for some users. As a result, software houses are offering data encryption programs that use the DES method to encrypt data on the system itself with no special hardware.

DES in the future.

Use of the DES was expected to increase over the next several years, especially where information has to be exchanged between different companies, because it is a standard and it is possible to obtain different pieces of hardware or software which implement it and will still be compatible, as they have to meet the standard to be able to say they use DES; but recently, a snag has developed. Like most modern ciphers, DES uses a numeric key, and there were some arguments about how secure DES really is, based on the length of the key, which is 56 bits (the scheme adds bits to make it 64 bits long). Some of the developers suggested that the key should be 128 bits long, but the National Security Agency required the NBS shorten the key: some critics suggested that a key of this length is such as to be virtually unbreakable by anyone except the NSA itself. Even so, it was expected that the DES would probably be secure enough for most commercial users for the foreseeable future, or at least through 1987, but recently the NSA has been privately telling hardware companies not to put the DES into any new equipment, and to stop using it now. They apparently want to use a new algorithm which will not be made public, ostensibly for better security, but possibly for

other motives. In spite of this, the DES will continue to be very difficult for commercial and home users to break, so it will probably continue in use for some time (remember what was said earlier about determining from whom you wish to protect your data).

Additional Precautions.

If you expect a real effort will be made to defeat your encryption scheme, there are a few extra precautions that can be taken to reduce the risk. The easiest way to break a code is if you have a copy of the enciphered message and the clear text together, and can compare the two to work back to the cipher. This indicates that access to important information should be carefully restricted: for example, if encryption is used to protect data during transmission, then when the data is deciphered and safe, the enciphered copy should be erased or destroyed. If it is carelessly discarded it might give someone a chance to work on it at leisure, especially if the threat is within the company where the clear text might also be available. Some newspaper codes were broken because the text of an article was transmitted in cipher (by radio, where it could be heard) and then printed word for word the next day in the paper: sending the contents of the article but re-wording it before releasing it to the public helped solve that problem. Similar precautions could be taken if such things as financial reports are to be transmitted: if possible, don't transmit the data in exactly the same form in which it will be published. In the case of business letters and memos, most start with a date and the person to whom it is addressed, and someone could know (or guess) how the message starts, and use that to cut down the number of attempts needed to find the key to the cipher: one way to stop that is to arbitrarily cut the memo in the middle somewhere, and put the last part before the first. The recipient, after deciphering, can easily see where the real beginning is, and move it back where it belongs. In short:

Don't be predictable.

There are also a few other precautions one can take if you feel that someone is really trying to defeat your encryption scheme. If you think someone is trying to get your key by brute force, you can put random garbage at the beginning and end of your data: anyone who is trying a key and checking only the beginning of the file to see if the data makes sense will not realize it if they do find the right key, as the decrypted data still won't make sense. Of

course, anyone can simply check the entire contents of the message for every key tried, but this is much slower, and anything that slows the process of defeating an encryption scheme means the scheme is that much more secure. If there is some reason to believe that whole messages are being intercepted and stored (with some ciphers, the more data you have, the easier it is to find the key), then you should change the key more often than you might otherwise do. In any event, you should not use a given key for too great a period of time, just in case someone is collecting your messages: with many ciphers, having

just two messages enciphered with the same key makes breaking the cipher very much easier than having just one message, and the task is greatly simplified with each extra message intercepted with the same key. [There is a method, at least theoretically, where having two messages sent with the same key, and where the general nature of the encrypted data is known (for example, English text in ASCII characters), which would allow breaking virtually any known cipher. It would be rather difficult to accomplish, but the more valuable the data, the greater the effort worthwhile to obtain it.] You can also

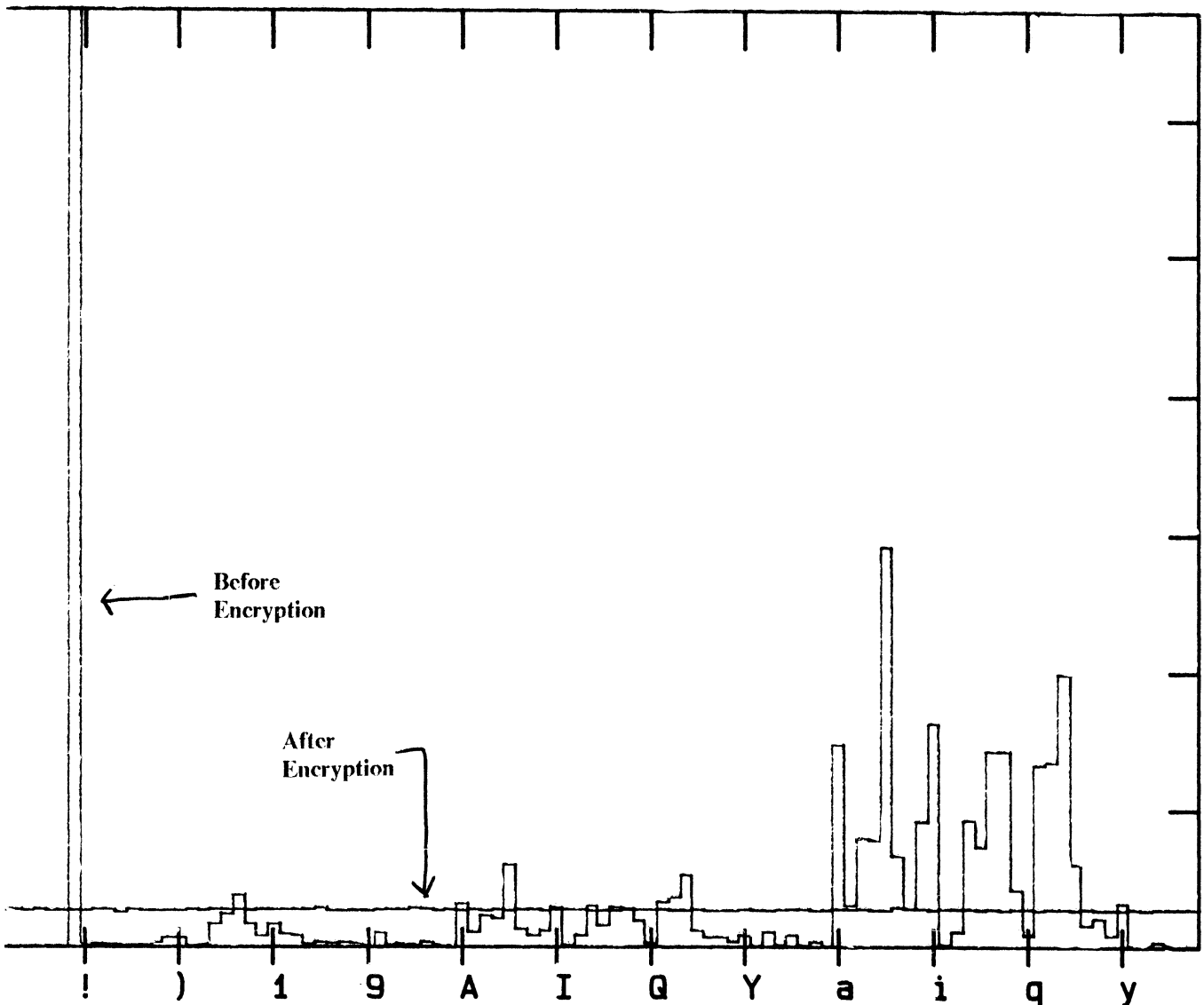


Figure 3 : Character Count of ASCII Text

occasionally send out messages which are the same length and otherwise look like your real messages, but which contain enciphered garbage. The contents (before enciphering) should look as much like real data as possible, without actually meaning anything. This will add to the difficulty of defeating the encryption scheme, but is only worth while if there is a real possibility that someone is making a concerted effort to break the cipher or if the cost of doing this is low.

Some timing comparisons.

As a test of the time required to encrypt data, I took a sample file of 743 blocks of text (the combined HELP files on an RSX system), which contained 380,416 characters, or 11,423 lines, or about 200 printed pages. To print this entire file on a fast (600 lines per minute) printer would take about 20 minutes: printing on a 120 character per second printer or transmission over a 1200 baud line (with no error checking) would take about 53 minutes: printing on a 30 character per second printer or transmission over a 300 baud line would take 3 hours. Using a simple FORTRAN program and the Infinite Key cipher on a PDP-11/70 and RP06 disks, encryption took about 4 minutes (or about 1500 characters per second): the same program on a PRO-350 (equivalent to a PDP-11/23) took 22 minutes (or about 300 characters per second). In both cases, performance was basically I/O limited, and the program was not doing anything special to increase I/O speed. It can be seen that encryption need not add to the time taken to process data: the program which reads a file and transmits it over a communication line could also encrypt and do so at a speed high enough continue to drive the line at it's full speed. I tried a program that did simple substitution (to produce some of the examples in this paper), and since it does the same I/O, the only possible difference is in the cipher code: accessing a substitution matrix was not significantly faster than generating the pseudo-random number sequence, so using a "cheap" substitution cipher does not save any computer or clock time over a much better cipher, and provides much less security.

Testing Ciphers.

Testing a cipher for security is really a job for experts: history has demonstrated many times that it is much easier to think up a cipher than to test it for security, and that only a person well acquainted with the methods of breaking ciphers can determine if a new cipher can be broken. Some familiarity with the methods used are useful in knowing what precautions to take against them, though, and the references given below list many of them. The graph reproduced below (please refer to figure 3) also demonstrates some of the basic principles: it is a count of all of the various ASCII characters in a text file (a smaller version of the file used for the timing tests discussed before). The jagged line with the sharp peaks is the character distribution: the greatest peak is for the blank space, with other peaks for the lower case vowels. A substitution cipher simply moves these peaks to different letters; finding the peaks reveals the mapping and is the reason why most substitution ciphers don't provide any real security. The nearly flat line near the bottom of the graph is the same text file after being processed with the Infinite Key cipher. Because the pseudo-random number generator used is fairly good, all of the peaks have been flattened out, and frequently occurring letters no longer provide a "handle" for breaking the cipher. What this chart does not show is that other patterns, such as commonly occurring pairs of letters such as "th" must not map into pairs of encrypted characters, and that other patterns in the clear text must not be carried through into the encrypted text. This demonstrates that while removing the simple letter frequency pattern may be a requirement of a good cipher, it is not sufficient in itself, and much less obvious tests must be made if the cipher is to withstand expert attacks.

Bibliography

There are a number of good descriptions of cryptography in popular literature. In addition to the two examples of the simple substitution cipher given before (*The Gold Bug* by Edgar Allan Poe and *The Adventure of the Dancing Men* by Sir Arthur Conan Doyle), two books by Dorothy L. Sayers (in addition to being entertaining in themselves) are of interest. *Have His Carcass* contains a good description of the Playfair cipher (a good combination transposition and substitution cipher which is easily worked with only a pencil and paper), and a good description on one way to attempt to break it which also clearly shows the hazard of sending messages in a form which allows the content to be deduced. *The Nine Tailors* contains an extremely ingenious example of secret writing. Both are currently published in paperback.

On a more formal basis, the following will be useful:

Cryptanalysis, a Study of Ciphers and their Solutions by Helen Fouche Gaines (Dover Publications, Inc.)

though written before computers were developed, contains thorough descriptions of many ciphers, and specifically the methods used to defeat them, with worked examples and reference tables. Dover has a mail order department.

Security and Privacy in Computer Systems by Lance J. Hoffman (Melville Publishing Co.) treats a wide variety of computer security subjects, one of which is the use of data encryption. It includes a good description of the "Infinite Key" cipher, with a mathematical test of its effectiveness. It also covers operating system security, physical plant security, and other subjects.

Cryptanalysis for Microcomputers by Caxton C. Foster (Hayden Book Co. Inc., Rochelle Park, New Jersey) Contains explanations of many ciphers, with programs in BASIC to implement them or act as aids in defeating them. The programs may require some work to implement (you have to search through the book to find the subroutines, and sometimes the names of variables change), but some good material is included. The programs are in a simple version of BASIC which most computers should handle "as is" or with only minor changes.

"Securing Data Inexpensively via Public Keys" by Brian Schanning (*Computer Design*, April 5 1983, Vol. 22 #4) is an article which describes the mathematics used to generate the two halves of a Public Key.

"The Data Encryption Standard, Recent Controversies" by John E. Hersey, (*Telecommunications*, Sept. 1983, Vol. 17 #9) gives an encapsulated history of the development of the DES, with some of the arguments for and against its method of implementation and use.

The Codebreakers by David Kahn (Macmillan) gives a good history of ciphers (and other data protection schemes such as voice scrambling) and their use, and a description of how some good modern ciphers were broken. The paperback version may be abridged. Considered one of the classic works on the subject.

I have not been able to review the following sources myself, but they may be useful.

"RSA: A Public Key Cryptograph System" by C. E. Burton, (*Dr. Dobbs' Journal*, Mar 1984, 16-21)

"Mathematical Games" by M. Gardner, (*Scientific American*, 237(2), August 1977, 120-124)

The following government publications may also be useful:

"Data Encryption Standard"
Federal Information Processing Standards
Publication 46

"DES Modes of Operation"
Federal Information Processing Standards
Publication 81

Standards Information Office
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D.C. 20234

The Smithsonian Institution has a section devoted to cipher machines, and give the following address for inquiries for more information on the subject:

Division of Mathematics
The National Museum of American History
Smithsonian Institution
Washington, D.C. 20560



DESIGNING FRONT-END AND INTERFACE SYSTEMS

FOR THE CASUAL END USER

BUD PINE
NCP COMPANY
111 ANZA BLVD.
SUITE 300
BURLINGAME, CA 94010

ABSTRACT

This paper describes techniques that were found useful in designing a front-end and interface system for the casual user of a database management system. These techniques can be applied to other front-end and interface systems.

INTRODUCTION

This paper describes techniques that have been found useful in designing a Front-End and Interface System for the Casual User and for maximizing usage on a system. The author believes that similar techniques can be used when designing other interface systems.

NCP Company has been producing system and application software packages for the DECsystem 10/20 community for the past eight years and more recently for the VAX family. During that time we have experienced a variety of requests for "user friendly" application and interface systems. It was not easy to ascertain exactly what the user community meant by this, until the System 1022 community and the NCP Calc community became very vocal and fairly explicit about their needs. It was at this point that NCP Company became interested in the concept and undertook the development of an interface system.

The primary requirement for the interface system was that it should be a transparent vehicle for the end-user of application software packages. It should not require a systems support staff to transfer the information from a database to another software package. A classic example given to us was of an executive that needed to get at financial data which was stored in a database file, make the appropriate selections, and bring it into a spreadsheet format in order to be able to manipulate the data within the confines of the spreadsheet.

At the onset, this seemed a fairly straightforward and "simple" task to the development team. In fact their original development schedule was three weeks! Once the process of design was underway the individual steps necessary to accomplish each task became visible to the design team. They came to realize that this "user" would have to have knowledge of the specific DBMS, (i.e., know how to open database files, specify fields, and select records), before a transfer to the spreadsheet could take place. In three weeks, we went back to the user community and said we could transfer the

data from the database to a flat file through a very simple menu system. Their response was, "no, we don't want to have to have the user know the database". They did not want to limit the usage of the DBMS to the current knowledgeable users but rather to increase the usage of the database management system within their organizations. In order to meet this criteria, it became necessary to design a user-friendly front-end to the database before the transfer of data could be made. With these parameters in mind NCP IFS took shape; and the design, development, and production began. Following are some of the techniques, ideas, and key features that we found useful in the design of this system and subsequent systems.

THE PROTOTYPICAL CASUAL USER

The reason for advocating this approach to designing interface systems for the casual user is the following: The end user gets a system that is truly user friendly. This is achieved because actual user views are fully considered and integrated into the design of the system. The design team can try out various ideas, scenarios, and approaches to designing the system directly with an interested user.

By considering the attitudes, responses and reactions from the prototypical casual user, the design team can build a more useful system for the end user. They can achieve this with lower development effort because the development priorities and resources go into developing the features that the casual end user will find useful, rather, than in features that the software designers feel will be required. If the prototypical user is selected properly, they are much better able to contribute to the design effort than the system designers are without input from a qualified casual user.

SELECTING THE PROTOTYPICAL CASUAL USER

The prime requirement of the prototypical casual user is that they understand the problems of every casual user. They have to have been a user of a variety of software so that they understand and can deal with the various kinds of problems that casual users experience.

Ideally, the prototypical casual user should also have been involved in the support of other casual users and in the training of casual users in the use of other computer software systems.

There is a danger however, if, the prototypical user becomes too knowledgeable. When they know all the subtleties and instantly know how to solve them, they get to the point where they no longer have the views of the "casual user", they cease to be that person. If this occurs, the proper approach should be to find another person to serve as the prototypical casual user. You want to avoid getting people that are too active in software design because these people are clearly not casual users. They are often heavy users and/or expert users, and cannot adequately empathize with kinds of situations that the truly casual user can get into, because an expert user knows how to avoid most of them.

Another requirement for the prototypical casual user is that they be very good at communicating with the software designers. It is insufficient to know what the casual users need and how the software system should be designed, if one cannot communicate that to the software designers. This communication ability is extremely important and if your choice does not have this ability it would be wise to look for another person to serve as the prototypical casual user.

This person needs to be very very good at visualizing. Often the software designers listen to a description of a problem and come up with a solution that is not exactly what the recommendation or description of the problem was, and the prototypical user, in order to continue to make a contribution needs to be able to visualize this new possibility. Often, the systems designers will come up with two or three solutions to each problem. It is more efficient if the basic decisions can be made without implementing each of them, merely to show how each proposed solution works. So, visualization is a key characteristic in the prototypical casual user.

BASIC CHARACTERISTICS OF THE CASUAL END USER

This paper now shifts attention from the prototypical casual user who is an active member of the design team, to a description of the characteristics that represent the casual end-user.

The prime characteristic of the "casual end user" is that they use the system infrequently. Their usage is sufficiently infrequent that for the most part they forget things by the time they need to do them again. They know what they want to do, they just don't know how to deal with a computer system and telling that system the necessary steps that must occur before the system can solve their problem.

Another characteristic is that the casual user typically gets very little training. They are often told, "Well, it's really easy, you do X, Y, and Z and the problem is solved." They're told to read manuals. The casual user's propensity to solve problems by reading manuals is matched only by the propensity of software designers to read manuals. Minimal training is justified in the case of the casual user because casual usage means that they're going to forget everything, so it's of limited benefit to spend company resources on training casual users.

Another characteristic of the casual user is that at the point in time that they have a problem they are often not close to documentation, that, and, most people's propensity not to read the documentation, means that they need other forms of "help" aside from written documentation. This can be in various forms. Help from the software system itself, or if that is inadequate, help from the systems staff to get resolutions to their problems.

SUGGESTED GUIDELINES FOR DESIGNING SOFTWARE SYSTEMS FOR THE CASUAL USER

A major problem with end-user software today is that as software becomes more powerful and sophisticated, the user tends to need to have knowledge of the product. However, most organizations do not have the time or the resources to train each and every employee, that will need, or should need, to use the system.

One of the best things you can do for the casual user is to minimize things that they have to remember. Even though they have done it fifty times before, they have done it fifty times over two years. That averages out to once every two weeks. They will not remember the specific keystrokes that are needed in order to accomplish each task.

"Casual" or "infrequent" users of a computer system basically means that they usually, or at least often, don't remember things that they've done before. When you do something once a week or once a month you tend not to remember all of the little details that are required when dealing with a computer software system. Unlike humans, the system is relentless in its need for precise instructions. There are a variety of things that the casual user needs to know in order to use a DBMS, such as, what the file name is, or which directory it is in. One of the features that we found helpful, was to build a "Database of Databases".

The Database of Databases is a Library or list of descriptions of the available data bases. The casual user need not remember a specific file name he merely needs to say, "I want to select a database from a library of those that are available". This Library is pre-built into the system by the Database Administrator, so that security is not violated. What the user sees in this Library, is a description of the database not necessarily the precise file name. The directory and the file names are built into the system, and therefore are invisible to the end user. For the experienced user an Expert Option, (i.e., "Specify Data Base Name") can be included.

Even when the user remembers the database name, they may not always remember the attributes/fields in the database. A built-in ability for the user to say, "I want to display what fields are available in the database, so that I can remember which ones I want to select on, or which ones I am not interested in showing, or which fields/attributes I am interested in interfacing to another software product", is a major requirement. The casual user needs to look at these fields in two ways. They could look at a List of all the Attributes contained in that database file or they could look at a specific Attribute and its Characteristics. With this ability, the occasional user can now look at the fields in sequence or look at a specific field. They can go forward, backward, and basically look at anything with respect to the characteristics of the database. Further, this has been done with a very simple menu system that requires little memorization or remembering of how to deal with the computer software system. Additionally, at this point they can specify which fields they would like to show (look at) later. This again gives the infrequent user the ability to specify fields while they appear on the screen, instead of having to remember them later.

This kind of approach has two benefits. It makes it much easier on the end user, in that they don't have to remember the precise characters of the file name and that they don't have to remember the precise syntax of the command in order to open the database. There is also a more subtle benefit. The system's staff doesn't receive as many phone calls to tell the casual users the characters in the database name, and the directory it is in. It is easier on the casual user and it is easier on the systems staff.

The casual user is often unsure of how to do what they want to do. It is often convenient and an excellent learning tool to let them try things and see if they work. We found the ability to "back-out" of commands wherever possible so that the casual user can try different options. If they get to a point where they determine that they don't want to be, they can either "back-out" one menu at a time or they can "back-out" all the way to the Main Menu. This allows the casual user to learn and get on-line training without needing an expert. If they're confused about "selecting" records or "showing" records of the database, they can try one option and see which one does what they want. It is unfortunate that the syntax and the words used in the computer business do not always match the words that the end user would use. Sometimes the user thinks of finding records in a database and sometimes they think of selecting records from the database and sometimes they want to use words like "match", or

"correlate", or "type", or "show"; there is no single "best" choice of key words in software systems, that is why customization is so valuable. Therefore, the casual user often needs to try things and see if the "buzz word" that is built into the software system matches what they want to do.

The system needs to include subtle, but valuable features. It is important to make the menu system very, very simple even at the cost of verbosity. It needs to be built sufficiently simple so that people "pick it up" without any extra effort. When the casual user makes a menu choice, the system needs to make something change as quickly as possible on the screen. Usually, this means that a line has been cleared to indicate that the character has been received by the computer and the requested option is being performed. The system also needs to be designed so that if the trained user types ahead, it skips displaying menus to the screen, as long as a valid menu choice has been entered into the system. This way, the user can optimize system efficiency because the system does not continue to type everything out.

FEATURES THAT MINIMIZE HAND-HOLDING FOR THE CASUAL USER

The casual user's attitude toward the systems support staff is often very much like our attitude toward dentists. We like to know they are there when we need them, but we certainly hope that we don't need them. For the competent professional who knows his business, and is intelligent, articulate, and trained, it is an unpleasant situation to have to go to someone else, (i.e., the systems support staff), and openly acknowledge their ignorance. Especially when they're often greeted with the comment, "Read the manual!" or, "Anybody should know that!" Hand-holding is unpleasant to everyone involved and it is a very expensive use of an organization's technical resources.

We believe in three kinds of help facilities. The first one, is analogous to many other systems. It is an on-line Help facility that is accessed by typing a "?" at any menu choice, or by typing a "?" as the first character to file names, requests, or the entering of contents of fields, and so forth. The on-line Help facility looks up a help message, and displays it on the screen and then gives the user a choice as to what to do. The author believes this is the minimally acceptable Help facility that any user friendly software system should have built into it. We've taken this two steps further. We've found that it is possible to predict the many kinds of trouble that the casual user can get into and how to build in an automatic facility such that when it is detected that the user has gotten into one of these situations an automatic help menu is displayed. For example, the user is selecting records from the database and no records are found, the system should come back and tell them that no records were found and offer advice as to the usual reason this problem occurs. Various forms of built-in automatic help are desirable where it is predictable that the casual user is going to find problems on a frequent basis. We

also found that some of the situations are sufficiently infrequent that by offering the user the choice of getting how-to-use help on a sophisticated feature it enables them to have instant access to an expert.

Menu systems are usually abhorred by the expert user and this is true whether the "expert" is a systems programmer, an experienced word processing operator, or an accountant working with electronic spreadsheets. Once the user becomes familiar with the product they tend to get bored going through a myriad of menus. With this in mind, we built-in the ability to type ahead; thereby, bypassing the screen having to refresh itself with every command. As long as a valid command was entered, the system should continue to work behind the scenes until it either finds an unrecognizable menu choice or you have ceased typing ahead.

One of the greatest "fears" of an occasional user of a computer system is, "What do I do if I type the wrong character?". In looking at alternative solutions to that problem we decided on a "back-up" ability. This allows the user to go back through each menu that they have traversed, one-by-one or all the way back to the Main Menu. This again eliminates the need for the systems support staff to have to correct "fatal" errors day in and day out.

CONCLUSION

It is a fact, in the computer software industry that you can have developed the greatest software package in the world, but if you don't have a marketing and sales force to inform the world of its existence, it is of little value to anyone. In fact, it is a reel of tape gathering dust. It is equally true that if you've developed a good software package and if that package has been marketed and sold appropriately, and is not installed at an organization's site, it is of little value if there is little or no usage of it. In fact, it becomes very expensive. Formal, internal and external training resources can only go so far. Very few organizations can afford to train all of the employees that could or would use the system. And even if an organization has an excellent in-house training program and succeeds in training an adequate number of employees there is always turnover. Part of the cost of turnover is the training cost of a new hire. Even if its not in the computer department, an employee will need to be trained at some level in the use of the computer system and its' software. Therefore, the key to increasing usage of sophisticated software packages is to make them easy and non-intimidating.

As more and more compute power and resources become available to a wider variety of users both software developers and in-house programming staffs are going to need to be more and more aware of the problems of the untrained professional in the use of computers and computer software systems. In this paper we've described them as casual users, infrequent users, and occasional users. They're not computer professionals. They are professionals in other areas that need to efficiently use the computer.

A DBMS Performance Evaluation Tool Description and
Methodology of Use

Alexander B. Wasilow

Robert L. Ewing

Gary B. Lamont
Visiting Professor
Department of Computer Science and Computer Engineering
Wright State University
Dayton, Ohio 45435

Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB Ohio 45433

ABSTRACT

This paper describes the use of a DBMS performance evaluation Tool and presents a methodology for its use. The tool, DBMON, has been developed at the Air Force Institute of Technology on a VAX 11/780 computer system. DBMON is capable of recording and analyzing performance measurements collected during the operation of the INGRES and TOTAL DBMSs. The DBMON User's Methodology is presented to describe the use of DBMON in the analysis and evaluation of DMBMS performance problems.

1. Introduction

The advantages of using a DBMS are numerous, but come at the expense of the overhead placed on the resources of a computer system. It is only natural to want to maximize the benefits of a DBMS while minimizing the overhead incurred through its use.

DBMS overhead and performance quality can depend on hardware configuration, operating system characteristics, DBMS query structures. In order to measure and identify areas where reductions to DBMS overhead can be made, a DBMS performance monitor named DBMON (Data Base MONitor), has been developed. DBMON is a tool that

has been designed to help free the Data Base Analyst from having to rely solely on intuition, experience, and trial and error techniques for identifying and correcting DMBMS performance problems. DBMON provides a means of measuring DBMS performance and its effect on computer system resources. The data provided by DMBMON can be used to evaluate DBMS performance based on specific user objectives. The evaluation provides the basis for identifying areas for performance improvement.

DBMON has been developed as an objective of the Air Force Institute of Technology Information Sciences Laboratory (1;2;3;5). DBMON is a part of an ongoing development of a software engineering environment that will include DBMS applications and analysis.

2. DBMON System Description

2.1. Introduction

Performance monitors can be grouped into one of three categories: software, hardware, or hybrid (4:82-108). The DBMON system implementation has been entirely software based thus far. It relies on VAX VMS Utility programs and custom DBMS monitoring software.

DBMON primarily operates by using performance measurement and collection software that is embedded into DBMS application programs. This embedded software is referred to as the Instrumentation Utility. DBMON allows the user to establish a monitor session, during which computer system statistics are recorded and DBMS application programs are executed. The Instrumentation Utility records DBMS Data Manipulation Language (DML) statements. Once the monitor session is completed, the collected data is analyzed and can be viewed by the DBMON user.

2.2. DBMON Structure

The structure of DBMON consists of four main functional areas as shown in Figure I. The four functional areas are as follows:

1. User Interface - This section allows the DBMON user to specify performance parameters to be measured, duration of the monitor session, and provides user control of monitor operation.
2. Measurement of System and DBMS - This portion of DBMON controls the execution of the monitor utilities and collects a file of performance measurement data.
3. Analyze Performance Measurement Data - This section performs mathematical, statistical, and graphical analysis of performance data in order to reduce the collected mass of raw DBMS performance data.
4. Present Performance Data to the User - This section presents the collected and analyzed performance data to the DBMON user in the form of reports and graphs.

Functional areas one and four are implemented with the DBMON User Interface Program. Area two functions are handled by the Instrumentation Utility and VAX/VMS Utilities under DBMON control. Area three is the responsibility of the DBMON Data Analysis Program.

2.2.1. Functional Areas One and Four.

The DBON User Interface is the primary interface between the DBMON user and the DBMON performance measurement process. This is a multilevel menu driven program that allows the setup, initiation, and control of a performance measurement session. After performance data analysis, the User Interface provides (1) graphical viewing of performance data represented by bar graphs; (2) terminal screen viewing of Performance Reports; (3) statistical comparisons between sets of performance measurement data; (4) storage of performance reports in a DBMON maintained library of historical performance data.

2.2.2. Functional Area Two.

The actual collection of performance data is under control of the User Interface but is carried out by DBMON and VAX/VMS Utility programs. These utilities are the DBMON Instrumentation Utility, the VAX Monitor Utility, VAX Accounting Utility, and the VAX SYE (System Error Log) Utility.

The DBMON Instrumentation Utility is a collection of modules that create a file of DBMS performance data. Instrumentation Utility calls are placed before and after the DBMS DML statements that are of interest to the SJDBMS Analyst. The instrumentation Utility uses before and after views of the DBMS to determine what system resources can be attributed to the DML statement being measured.

The Instrumentation Utility records performance statistics on the following six parameters:

- (1) Total response time for the DMB Instruction.
- (2) CPU time used.
- (3) Buffer I/O count.
- (4) Direct I/O count.
- (5) Number of page faults.
- (6) Working set size.

These parameters are recorded into a file that is later used by the DBMON Data Analysis Program.

The VAX Monitor Utility collects data on computer system performance at user specified intervals. The collected data is in four classes: I/O, Modes, States, and File System ACP Statistics (FCP).

The VAX Accounting Utility uses the VAX Job Accounting File to produce a summary report of jobs running on the computer system.

The VAX SYE Utility uses the system error log to produce a summary report of system error conditions.

2.2.3. Functional Area Three.

The DBMON Data Analysis Program uses the raw measurement data file produced by the DBMON Instrumentation Utility to produce the Instrumentation Report, graphical analysis files, and statistical analysis files. The Instrumentation Report is the summary of the data collected by the Instrumentation Utility. This report may be printed, or viewed with the DBMON User Interface Program. Examples of this report are in Figures 9 and 10. The two analysis files produced by the Data Analysis Program are used by the User Interface Program to provide graphical representations and statistical comparisons of performance data.

The data collected by the three VAX Utilities is merged and formatted to produce the System Parameter Report. This report shows the state of the computer system during the measurement session by indicating the system turnaround, throughput, device errors, security (login failures), processor utilization, and queue wait parameters. The information in the System Parameters Report helps allow the DBMS Analyst to determine the effect of computer system workload on DBMS operation. An example of this report is in Figures 2 through 7.

3. DBMON User's Methodology

Methodology can be defined as the synthesis of methods and tools. The tool in this case of DBMS performance evaluation is the DBMON system. DBMON and the methods of its application comprise a DBMON User's Methodology as represented in Figure 8. The DBMON User's Methodology for DBMS performance improvement consists of the following five phases:

1. Understand the System - This phase involves familiarization with the computer system, the DBMS, and the DBMS application being studied.
2. Identify Problem Areas - This phase involves analyzing the use of a DBMS to identify any potential problem areas that need to be addressed. This phase could also involve the creation of DBMS performance benchmark measurements against which to gauge any future DBMS performance measurements.
3. Formulate a Performance Improvement Hypothesis - Once familiar with the system, and having identified a problem area, possible performance improvement hypothesis need to be developed. These hypotheses should be analyzed to see if they are feasible. Unrealistic solutions to performance problems can be rejected at this stage.

The above DBMS performance improvement methodology would be cumbersome to apply without a complete DBMS performance analysis tool. DBMON can be directly applied to the identification of DBMS performance problems and the testing of performance improvement modifications. Data collected during the identification stage by DBMON can be applied to the formulation of a performance improvement hypothesis. Final implementation of a performance improvement can be validated with DBMON performance evaluation measurements. DBMON has been developed to be the workhorse in the application of the DBMS performance evaluation methodology.

4. Application of DBMON to the User's Methodology.

An example is the most straightforward method of demonstrating the use of DBMON and the application of the User's Methodology. Considering the case of a DBMS application program, the following discusses a step by step application of the User's Methodology.

DBMON was applied to the Data Analysis Program which is a part of the the DBMON software. The Data Analysis Program is an INGRES application program. The DBMON User's Methodology was applied with the following steps:

- (1) Understand System - Familiarization with the computer system, DBMSs, and the DBMON system was gained during the course of DBMON system development.
- (2) Identify Problem Areas - During testing of the Data Analysis program, it became evident that the response time was lengthy (more than 10 minutes). As a matter of convenience an improvement in program response time would be useful. The use of INGRES calls became the prime suspect in the lengthy response times, since the previous versions of the Data Analysis program ran quickly (less than 1 minute). In order to evaluate the response time contribution to the program overall response time, the Data Analysis Program's INGRES calls were instrumented with the DBMON Instrumentation Utility. The instrumented program was run during a measurement session, and the resulting measurement data was analyzed. A statistical analysis file was created against which to gauge any future performance improvements. Graphical analysis of the measurement data as well as examination the Instrumentation report (Figure 9) revealed that an average response time of 1.1 seconds. Considering that 703 INGRES instructions were executed, 12.9 minutes of Data Analysis Program execution time could be attributed to DBMS activity.

(3) Formulate a Performance Improvement Hypothesis - The collected measurement data was analyzed, and it became evident that more than 90% of the program response time was attributed to its embedded INGRES calls. Figure 9 shows the measurement report for the pre-improvement Data Analysis Program operation. From examination of the Instrumentation Report, a large amount of page faults occurred during use of the data base. This suggested a change in the structure of the stored data. By examining INGRES reference manuals, a possible performance improvement would be to use the INGRES provided features of restructuring the Data Base into a hash structure from its initial heap structure.

(4) Test Performance Improvements - During this step the data base was optimized using INGRES supplied procedures. The instrumented data analysis program was run again and a second set of measurement data collected. This data was analyzed and a second statistical analysis file was created. Figure 10 is the instrumentation report for this session. The performance improvements reduced INGRES response times approximately 50%. From examination of the reports it can be seen that the improvements came partially as a result of less CPU time used, but primarily because of the much smaller number of page faults encountered in restructured data base. These observations were confirmed through

the statistical comparison the statistical analysis files created for both measurement sessions.

(5) Implement Performance Improvement Modifications - Due to clear benefits of the hash data base structure, it has been adopted for use by the DBMON data base and has been specified in the DBMON User's Guide.

This sample application of the DBMON User's Methodology is representative of the procedures that can be used in applying DBMON as a performance improvement tool. It is obvious that DBMON is only a tool and is not a cure all for DBMS performance problems. The DBMS Analyst is still responsible for uncovering solutions, though his job can be much simpler with the use of DBMON.

5. Conclusions and Recommendations

The result of the DBMON development effort is a comprehensive data base performance measurement tool. As a stand alone DBMS performance evaluation tool it is capable of providing the DBMS analyst the means of collecting and analyzing large amounts of DBMS performance data. Testing performance improvement modifications becomes a straightforward application of the DBMON system. DBMON and the DBMON

User's Methodology are presented as a comprehensive methodology for analyzing and helping cure DBMS performance problems.

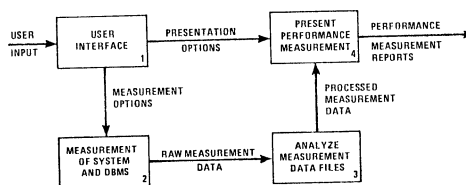


FIGURE 1 DBMON BLOCK DIAGRAM
Source: 12-11-83

```

DDDDDD  BBBB  M  M  M  OOO  N  N
D  D  D  B  MM  MM  O  O  NN  N
D  D  BBBB  M  M  M  O  O  NN  N
D  D  B  B  M  M  M  O  O  N  N
D  D  B  B  M  M  M  O  O  NN  N
DDDDDD  BBBB  M  M  M  OOO  N  N

DATA BASE PERFORMANCE MONITOR (VERSION 3.0)
AIR FORCE INSTITUTE OF TECHNOLOGY
WRIGHT-PATTERSON AIR FORCE BASE, OHIO
COPYRIGHT 1984 BY ELECTRICAL ENGINEERING DEPT (AFIT/EN)

CAPT ALEXANDER B. WASILOV
CAPT TIMOTHY D. BAUER
CAPT PAUL D. BASLER
DR. GARY B. LAMONT

```

```

DBMON SESSION INFORMATION
-----
MEASUREMENT SESSION NAME : SVSTEM PARAMETER REPORT TEST
START DATE                : 13-SEP-1985
START TIME                : 13:37:28
STOP DATE                 : 13-SEP-1985
STOP TIME                 : 18:00:00
ANALYSIS OPTION           : STANDARD
PRESENTATION OPTION       : PRINT
COLLECTION INTERVAL      : 15
SESSION STATUS            : COMPLETE
PARAMETER SET             : ALL

```

FIGURE 2

```

-----
* PERFORMANCE PARAMETER REPORT                                     START TIME
*                                                                 STOP TIME
-----
PRODUCTIVITY DATA
-----
DBMS SUBPROCESS THROUGHPUT
USER  ACCOUNT  JOBNAME  START TIME  ELAPSED TIME
DBMON ENGR    13-SEP-1985 13:38      0 00:00:16.39
DBMON ENGR    13-SEP-1985 13:44      0 00:16:32.29

SYSTEM THROUGHPUT
TYPE  USER  JOB NAME  RECORDS ELAPSED TIME
PROCESS DBMON          40 04:32:56.80
PROCESS GATRAD        10 03:12:05.88
PROCESS ICECAP        10 04:02:55.61
PROCESS PICTURE       10 01:14:09.00
PROCESS ??            10 00:00:00.00

INTEGRITY DATA
-----
DBMS SUBPROCESS COMPLETION STATUS
USER  ACCOUNT  JOBNAME  STATUS CODE  STATUS TEXT
DBMON ENGR    00000001  SVSYSTEM-S-NORMAL, normal successful comp
DBMON ENGR    00000001  SVSYSTEM-S-NORMAL, normal successful comp

DEVICE ERRORS
DEVICE  HARD  SOFT  ERRORS THIS SESSION
_MSA0:  5.      28.

SYSTEM ERROR SUMMARY
DEVICE ERROR BIT SET  5.

```

FIGURE 3

```

*****
* P E R F O R M A N C E   P A R A M E T E R   R E P O R T
*****
                         START TIME
                         STOP TIME
*****
                         *****
                         * EFFICIENCY PARAMETERS *
                         *****

SECURITY DATA
-----
LOGIN FAILURES BY USER
USER   ACCESS ATTEMPTS
<login> 4
SYSTEM 1
TOTAL   5

LOGIN FAILURES BY TERMINAL
TERMINAL USER   ACCOUNT   STATUS TEXT
TTA4:    <login>  NLOGIN-E-NDSUCHUSER, no such user
TTA4:    <login>  NLOGIN-F-CMDINPUT, error reading command input
TTA5:    <login>  NLOGIN-F-CMDINPUT, error reading command input
TTB4:    SYSTEM    <no text>
TTB4:    TOTAL    <no text>
TTB5:    <login>  NLOGIN-F-CMDINPUT, error reading command input

```

FIGURE 4

```

*****
* P E R F O R M A N C E   P A R A M E T E R   R E P O R T
*****
                         START TIME
                         STOP TIME
*****
                         *****
                         * EFFICIENCY PARAMETERS *
                         *****

Page Write I/O Rate      0.01

DEVICE UTILIZATION DATA
-----
DEVICE QIOS THIS SESSION
   _WSAD: 1182805.

QUEUE DATA
-----
DBMS SUBPROCESS QUEUES
USER   ACCOUNT   JOBNAME   QUEUE NAME
DBMON  ENGR
DBMON  ENGR

QUEUE WAIT PARAMETERS
Collided Page Wait      0.00
Common Event Flag Wait  1.39
Compute (Outswapped)    0.00
Current Process         1.00
Free Page Wait          0.00
Hibernate               4.84
Hibernate (Outswapped)  0.00
Local Ext Flag (Outswapped) 0.00
Local Event Flag Wait   9.28
Misc & Misc Resource Wait 0.02
Page Fault Wait         0.01
Suspended              0.00
Suspended (Outswapped) 0.00

```

FIGURE 7

```

*****
* P E R F O R M A N C E   P A R A M E T E R   R E P O R T
*****
                         START TIME
                         STOP TIME
*****
                         *****
                         * EFFICIENCY PARAMETERS *
                         *****

ALLOCATION DATA
-----
DBMS SUBPROCESS ALLOCATION
USER   ACCOUNT   JOBNAME   PRIORITY   VOLUMES MOUNTED
DBMON  ENGR       4         0
DBMON  ENGR       4         0

UTILIZATION DATA
-----
DBMS SUBPROCESS PROCESSOR TIME
USER   ACCOUNT   JOBNAME   PROCESSOR TIME
DBMON  ENGR       0 00:08:27.78
DBMON  ENGR       0 00:00:07.78

PROCESSOR UTILIZATION BY USER
TYPE   USER   JOB NAME   PROCESSOR TIME
PROCESS DBMON  0 00:23:12.45
PROCESS GAITROS 0 00:18:51.75
PROCESS ICECAP1 0 00:12:18.09
PROCESS PICTURE 0 00:03:24.80
PROCESS TI     0 00:00:04.41

PROCESSOR UTILIZATION PERCENTAGE
Executive Mode      2.9
Idle Time           85.8
Supervisor Mode     0.0
User Mode           21.7

MEMORY DATA
-----
DBMS SUBPROCESS MEMORY DATA
USER   ACCOUNT   JOBNAME   PG FAULTS   PG READS   PEAK WS
DBMON  ENGR       4494      23          512
DBMON  ENGR       1293      24          512

```

FIGURE 5

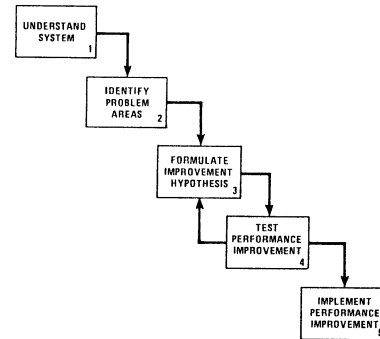


FIGURE 8 DBMON USER'S METHODOLOGY

```

*****
* P E R F O R M A N C E   P A R A M E T E R   R E P O R T
*****
                         START TIME
                         STOP TIME
*****
                         *****
                         * EFFICIENCY PARAMETERS *
                         *****

MEMORY UTILIZATION BY USER
TYPE   USER   JOB NAME   RECORDS   PEAK WS   PAGE FAULTS
PROCESS DBMON  4 3000 38619
PROCESS GAITROS 1 1024 56740
PROCESS ICECAP1 1 1024 21816
PROCESS PICTURE 1 850 10774
PROCESS TI     1 639 614

MEMORY UTILIZATION PARAMETERS
#max List Size      3939.08
Max/Min List Size  163.94

I/O DATA
-----
DBMS SUBPROCESS I/O DATA
USER   ACCOUNT   JOBNAME   DIRECT I/O   BUFFERED I/O
DBMON  ENGR       57        88
DBMON  ENGR       8768

I/O UTILIZATION BY USER
TYPE   USER   JOB NAME   RECORDS   BUFFERED   DIRECT   PAGE   QIOS
PROCESS DBMON  4 58053 13795 2182 0
PROCESS GAITROS 1 32957 5280 1160 0
PROCESS ICECAP1 1 19454 23851 1203 0
PROCESS PICTURE 1 2799 994 389 0
PROCESS TI     1 133 27 51 0

I/O UTILIZATION PARAMETERS
Buffered I/O Rate  11.86
Cache Hit Rate    2.35
CPU Tick Rate     1.63
Direct I/O Rate   5.51
Disk Read Rate    0.28
Disk Write Rate   0.55
FCP Call Rate     1.86
Page Fault Rate   13.20
Page Read Rate    4.22
Page Write Rate   0.86
Page Read I/O Rate 0.37

```

FIGURE 6

```

PROGRAM NAME: DATA ANALYSIS PROGRAM TEST 1
DML STATEMENT SUMMARY
-----
RETRIEVAL COMMANDS
COMMAND NAME   TYPE   EXECUTION COUNT   RESPONSE TIME(sec)   CPU TIME(msec)   BUFFERED I/O   DIRECT I/O   PAGE FAULTS   WORKING SET
RETRV         total 1 4.860 1490 2 54 193 512
              average 4.860 1490 2.0 54.0 193.0 512.0
-sum         total 1 4.860 1490 2 54 193 512
              average 4.860 1490.0 2.0 54.0 193.0 512.0

STORAGE COMMANDS
COMMAND NAME   TYPE   EXECUTION COUNT   RESPONSE TIME(sec)   CPU TIME(msec)   BUFFERED I/O   DIRECT I/O   PAGE FAULTS   WORKING SET
APEND         total 687 377.420 74240 3454 1387 7 512
              average 0.549 111.0 5.0 2.0 0.0 512.0
DELET         total 6 190.220 29700 23 411 1558 512
              average 31.703 4950.0 3.8 68.5 259.3 512.0
REPLA         total 9 196.240 75570 87 2317 3838 512
              average 21.804 8841.1 9.7 246.3 426.2 512.0
-sum         total 702 763.880 185510 3544 4015 5399 512
              average 1.088 264.3 5.0 5.7 7.7 512.0

SUMMARY OF ALL COMMANDS
COMMAND NAME   TYPE   EXECUTION COUNT   RESPONSE TIME(sec)   CPU TIME(msec)   BUFFERED I/O   DIRECT I/O   PAGE FAULTS   WORKING SET
-sum         total 703 766.740 187000 3546 4089 5592 512
              average 1.094 266.0 5.0 5.8 8.0 512.0

```

FIGURE 9

PROGRAM NAME: DATA ANALYSIS PROGRAM TEST 2

DML STATEMENT SUMMARY

RETRIEVAL COMMANDS

COMMAND NAME	TYPE INFORMATION	EXECUTION COUNT	RESPONSE TIME(msec)	CPU TIME(msec)	BUFFERED I/O	DIRECT I/O	PAGE FAULTS	WORKING SET
RETRV	total	1	2.230	1290	2	82	113	512
	average		2.230	1290.0	2.0	82.0	113.0	512.0
"SUM"	total	1	2.230	1290	2	82	113	512
	average		2.230	1290.0	2.0	82.0	113.0	512.0

STORAGE COMMANDS

COMMAND NAME	TYPE INFORMATION	EXECUTION COUNT	RESPONSE TIME(msec)	CPU TIME(msec)	BUFFERED I/O	DIRECT I/O	PAGE FAULTS	WORKING SET
APEND	total	687	206.180	74480	3432	1387	2	512
	average		0.300	108.4	5.0	2.0	0.0	512.0
DELET	total	6	51.210	28510	31	459	364	512
	average		8.535	4751.7	3.5	76.5	94.0	502.7
REPLA	total	9	125.900	79800	85	2513	2929	512
	average		13.989	8866.7	9.4	279.2	325.4	512.0
"SUM"	total	702	385.290	182770	3538	4359	3495	512
	average		0.548	260.4	5.0	6.2	5.0	511.9

SUMMARY OF ALL COMMANDS

COMMAND NAME	TYPE INFORMATION	EXECUTION COUNT	RESPONSE TIME(msec)	CPU TIME(msec)	BUFFERED I/O	DIRECT I/O	PAGE FAULTS	WORKING SET
"SUM"	total	703	385.520	184080	3540	4421	3808	512
	average		0.548	261.8	5.0	6.3	5.1	511.9

FIGURE 10

References

1. Bailor, Paul D. Development of a Data Base Management System Performance Monitor. MS thesis, AFIT/GCS/EE/83D-2. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1983.

2. Bailor, PAUL D., Lamont, G. B., and Ewing, R. L. Development of a Data Base Management System Performance Monitor. MS thesis, AFIT/GCS/EE/84D-6. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1984.

4. Svobodova, Liba. Computer Performance Measurement and Evaluation Methods: Analysis and Applications. New York, NY: American Elsevier Publishing Co., 1976.

5. Wasilow, Alexander B. Development Completion of a Data Base Management System Performance Monitor. MS thesis, AFIT/GCS/EE/85D. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1985.

THE COMPLEX NETWORK: A DATABASE DEFINITION DILEMMA

Mildred D. Lintner and David W. Chilson
Bowling Green State University
Bowling Green, Ohio

ABSTRACT

This study examines complex networks as implemented by DBMS-20, a CODASYL DBTG database management system. Textbooks define a "complex network" in one of two ways (sometimes asserting both): (1) as a many-to-many relationship between two record types, and (2) as a one-to-many relationship which goes in both directions between two record types. Using a student-course database and a state-president-vice president database, this paper demonstrates that a true complex network involves the first definition and requires decomposition of the many-to-many relationship into a pair of simple networks through the introduction of an intersection record type. The paper further demonstrates that a one-to-many relationship in both directions is **not necessarily** a complex network and, if not, can be implemented directly by DBMS-20 without decomposition and without the introduction of an intersection record type. The term "compound" network is proposed as a name to designate this second type of relationship.

INTRODUCTION

As a technology matures, definition of its terminology is continually being refined in the laboratories of research and experience. This refinement allows the technology to be described, and therefore understood, with increasing accuracy and precision. Especially important are definitions of the underlying structures upon which the practical applications of the technology are founded. Sometimes, as various applications expose the central issues of a technology, small differences among definitions cause discrepancies to arise between the conceptual basis and the practical applications of the technology. Such is the case with one of the underlying structures of database technology, the complex network.

The most common definition of a complex network is that it involves a many-to-many relationship. In this data structure, a parent-record may have many child-records, and a child-record may have many parent-records of the same record type. Database literature contains many examples of this definition in use. For example, Joseph A Vasta, in his text Understanding Data Base Management Systems (1985), defines a complex network as one in which "a many-to-many relationship exists between the child and the parent" (Vasta, 1985: p. 73). A similar definition is used by James Martin, in Computer Data-Base organization (1977), stating that a complex network is represented schematically by "one of the lines [connecting record types] having double arrows in both directions" (Martin, 1977: p. 113).

While this definition is most widely used, another definition exists in the literature. In this case the relationship between the two record types in the structure

is described as a pair of parallel, or binary, one-to-many relationships. That is, there is a one-to-many relationship in both possible directions. Jeffery D. Ullman, for example, in Principles of Database Systems (1980) defines complex networks as "binary many-one relationships" (Ullman, 1980: p. 83). And David Krpenke, in Database Processing: Fundamentals, Modeling and Applications (1977) offers the following definition: ". . . we shall refer to networks having one-to-many relationship in both directions as complex networks." (Kroenke, 1977: P. 79).

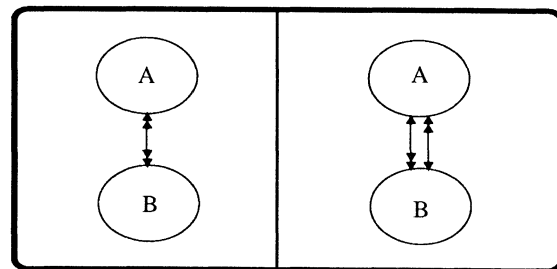


Figure 1

While these two pairs of definitions may appear similar, fundamental differences exist between them. Basically, any many-to-many relationship is also a binary one-to-many relationship that goes in both directions. The reverse, however, is not necessarily the case. That is, a one to many relationship that goes in both directions between two record types is not necessarily a many-to-many relationship. To demonstrate these differences, consider two databases: (1) a student-course database, and (2) a state-president-vice president database.

A STUDENT-COURSE DATABASE

The classic example of a many-to-many complex data definition is the student-course database. Each student can be enrolled in one or more courses during a particular academic term, and any course can in turn have one or more students enrolled.

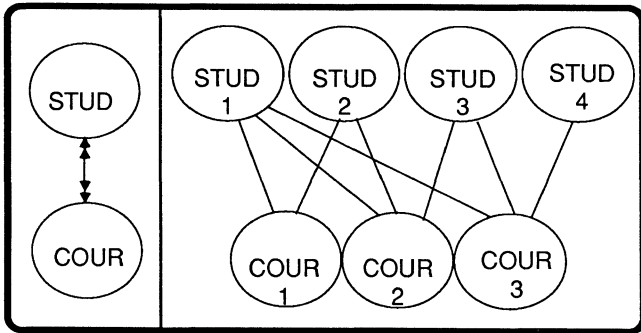


FIGURE 2

The relationship between students and courses can be expressed as either many-to-many or bi-directional one-to-many. As such, it is a true complex network, and cannot be directly implemented by DBMS-20. The reason for this is that DBMS-20 is based on the CODASYL DBTG database model. Under this standard, a member record can have only one owner in a given set. A member record occurrence in a particular set can have only one owner pointer for each set type in which it is designated as a member. Multiple owner pointers can be used, but each would have to be for an owner in a different set type. As you see in Figure 1, a given course would need one or more owner pointers within the student-course set, since a given course could have one or more student owners. Likewise, a given student record would need one or more owner pointers within the course-student set since a given student could have one or more course owners.

What is needed to solve this problem is the introduction of a third record type, an intersection or connector record, which contains the intersecting primary keys of pairs of student and course records, as well as pointers in each of two set types -- a student-intersection set and a course-intersection set. Introducing the third record type reduces the original complex network to a pair of simple networks. The resulting structure of record types is as follows:

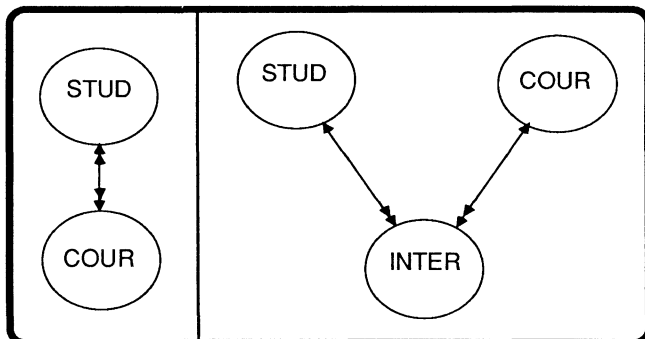


Figure 3

The student-course database is now in a form that can be directly implemented by DBMS-20. The student records are owner records in the student-intersection set, the course records are owners in the course-intersection set, and the intersection records are members of both sets. Each member record in this case has two owners, but each owner is in a different set type.

A STATE-PRESIDENT-VICE PRESIDENT DATABASE

In the state-president-vice-president database, the structure of record types is the following:

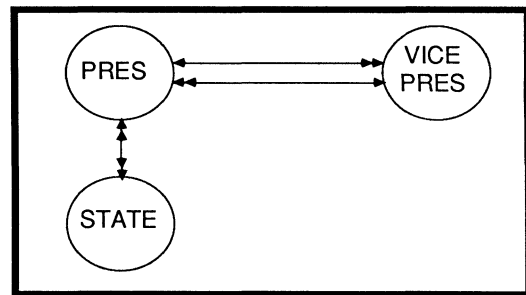


Figure 4

A state could have had one or more presidents come from that state (i.e., state of birth), and a president could have had one or more states admitted to the union during his term(s) or office. Additionally, a president could have had one or more vice presidents serve under him during his administration, and a vice president could have served under one or more presidents. Note that in this database the state-president relationship is one-to-many in both directions, while the president-vice president relationship is many-to-many. The state-president relationship **is not** a complex network. Therefore, it **does not** require the introduction of an intersection record type to reduce the relationship to two simple networks. The president-vice president relationship **is** a complex network, and therefore **does** require the introduction of an intersection type.

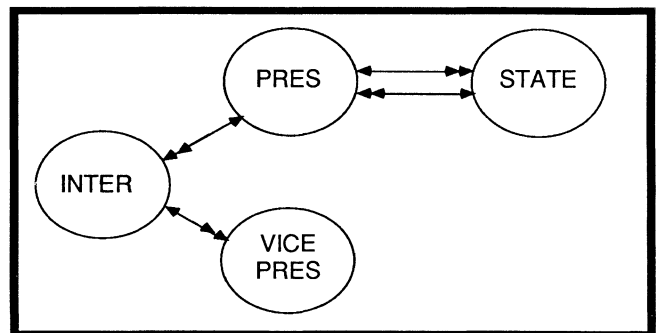


Figure 5

The state records are owner records in the state-president set and member records in the president-state set. The president records are owner records in the president-state set and member records in the state president set. In contrast to the student-course database, in which multiple owner pointers would have been needed for direct

implementation, note that president records (as member records) require only a single owner pointer in the state-president set. Any given president could have been born in only one state. Similarly, state records (as member records) require only a single owner pointer in the president-state set. A given state could have been admitted to the union during the term of only one president.

The president records are also owner records in the president-intersection set. The vice president records are owner records in the vice president-intersection set. Finally, the intersection records are member records in both sets.

CONCLUSION

What distinguishes the state-president relationship from the student-course relationship (as well as from the president vice-president relationship is that the meaning if the relationship is not the same in both directions. In one case, "state" means "state of birth"; in the other, it means "admission of the state to the union". The relationship might be classified as a special case of a bi-directional one-to-many relationship between record types, but it is certainly not a complex network. Some clarification of terminology, therefore, is needed regarding the definition of complex networks.

Usually, when there are dual definitions for one technical idea, those definitions are refinements of one another, or restatements, made for the sake of clarification. This study demonstrated that that cannot be the case with 'complex network'. Major conceptual and implementational differences between data structures described by the two definitions were found to exist.

A many-to-many structure comprises a single relationship which can be stated as parallel statements: There are many X in Y: There are many Y in X. The relationship remains the same whichever way it is stated.

The bidirectional one-to-many structure, however, comprises two different, indeed independent, data relationships, which coincidentally depend on the same two record types. Typically, these relationships cannot be reduced to parallel statements. For example, the president-state database used in this study presented this bidirectional relationship: Many presidents could have been born in one state: Many states could have been brought into the union during the administration of one president. The conceptual difference between this pair of statements and the pair above is clear. Neither of these can be stated as the inverse of the other and still maintain accuracy in describing the relationships. Thus, the two data structures were seen in this study to be not two definitions of a single concept, but two distinctly different entities.

In conformance with accepted expectations, it was found that the many-to-many relationships studied could not be directly implemented in DBMS-20, a CODASYL database system. Both many-to-many structures (supplier-part and president-vice president) needed to be decomposed into simple networks by the introduction of intersection or connector records before DBMS-20 could implement them.

The bidirectional one-to-many relationship exemplified by the president-state structure, however,

needed no such artificial insertions with the bidirectional one-to-many structure. Being declared as two separate sets (each representing one of the two relationships), the president-state database was directly implemented.

The difference in implementation between the many-to-many structure and the bidirectional structure adds the weight of its evidence to the significance of the conceptual difference described above. If the two definitions describe different entities, and if the two entities behave differently and are treated differently by at least one model of database management system, should they not be given different names?

The many-to-many structure behaves and is treated in accordance with our expectations and predictions concerning 'complex network'. It therefore should be given sole possession to that title, thereby maintaining the accuracy of an overwhelming compendium of database literature, including the CODASYL standards documents.

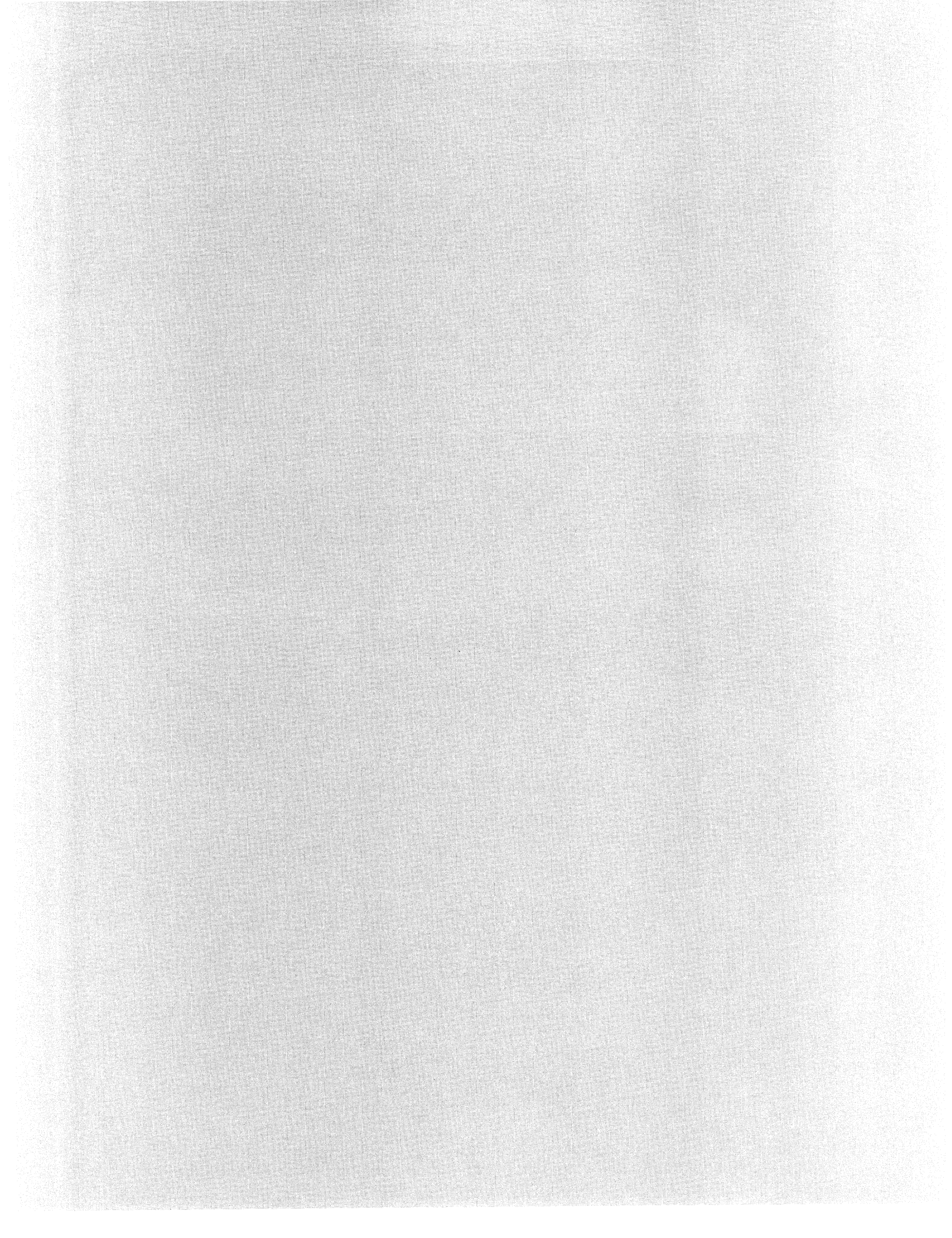
The bidirectional one-to-many structure, however, should be renamed with a unique title descriptive of the relationships it contains. An appropriate title might be 'compound network'. This term is in keeping with the names given to other data structure types in that the three terms together, simple, compound and complex, describe an increasing degree of complexity in the data types, in much the same way that identical terms describe more and more convoluted sentence structure in English.

With the compound network data structure defined and differentiated from the complex network structure, efforts should be made to investigate it further, and to treat it as a separate data type in text and scholarly literature. Only in this way can the accuracy of data structure description be maintained and the definition dilemma be solved.

BIBLIOGRAPHY

- CODASYL Data Base Task Group April 71 Report, Conference on Data Systems Languages, April, 1971. Third Printing, 1977.
- Kroenke, David. Database Processing. Chicago: Science Research Associates, Inc., 1977.
- Martin, James. Computer Data-base Organization, second edition. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1977.
- Ullman, Jeffrey D. Principles of Database Systems. Potomac, Maryland: Computer Science Press, Inc., 1980.
- Vasta, Joseph A. Understanding Data Base Management Sysyems. Belmont, California: Wadsworth Publishing Company, 1985.787

DATATRIEVE SIG



DATATRIEVE Novice Questions & Answers

Joe H. Gallagher
4GL Solutions
Kansas City, MO

Chris Wool
DuPont Corp.
Wilmington, DE

B. Z. Lederman
Brooklyn, N.Y.

Larry Jasmann
U.S. Coast Guard
Burke, VA

Transcribed by B. Z. Lederman

Abstract

This is a transcription of a panel presentation which answers some of the most common questions asked by new DATATRIEVE users. The transcription may paraphrase some questions or answers for clarity, and the transcriber apologizes in advance for any misspelled names: the usual convention of placing square brackets around interpretations or material supplied by the editor is followed in this paper. DTR is used in this paper as an abbreviation for DATATRIEVE.

Dana Schwartz: *I already have a record definition for a domain, and I want to add a field in the middle of the record. How do I do that?*

(Joe:) This is relatively easy. Ready the existing domain under an alias, for example:

READY FOO AS OLD READ

This brings the old definition into memory and sets it aside so you can read the data in the domain.

EDIT FOO_RECORD

to bring the record definition into the buffer, and add the new field(s). Exit from the editor, and the new definition is now in the dictionary.

DEFINE FILE FOR FOO

to create a new file for the revised record definition.

READY FOO AS NEW WRITE

to ready the new domain.

NEW = OLD

to move all of the old data into the new domain.

This will leave default values (blanks or zeros) in the new fields.

If you want to put something into the new fields, you can do the following (in place of **NEW = OLD**):

```
FOR OLD STORE NEW USING BEGIN
NEW_REC = OLD_REC
NEW_VARIABLE = constant
END
```

It is quite easy to add a new variable and restructure a domain with a few statements. One caution: if the system crashes while you have NEW and OLD readied, the definition will be in the new form, while the data is in the old form. You may have to delete the new (partially filled) file, and the new record definition and go back to the old ones (or rename them to save the new definitions).

Rand Wilson, Wilson Concrete: *I've done restructuring a few times, and it seems to compress the data (the new file occupies less space). Will you comment on that?*

(Chris:) You are using an indexed file? (Yes) You have also done multiple deletes before restructuring? (Yes) When you delete records, you don't delete everything: records have what is called a Record File Address (RFA). When you delete a record, the RFA cannot be reused, so there is a little piece of information which is still in the file. When you restructure the file and rewrite the data, you are eliminating all of the little pieces of unusable space. If you are doing a lot of deletes (or storage of new records) on a file, you should run CONVERT fairly often to rewrite the file. One thing you have to watch out for is VMS V4 CONVERT, which reduces the amount of blocks used, but not allocated: if you start with an 8000 block file and after conversion you only need 3000 blocks, 8000 are still allocated. That is a bug which is known and should be fixed some time in the future. If you are tight on quotas you may have to use FDL to define a new file of the proper size and have CONVERT use the new definition to set a file of the proper size. CONVERT is a DCL level command, and is faster than using DTR to read one file just to write to another. If you are adding a field to a large data file (more than 1000 - 2000 records), you may want your NEW domain (from the previous question) to be a sequential file, and then use CONVERT to change it to an indexed file. DTR is not the most optimum tool for populating and empty indexed file from both performance and time stand points, because DTR populates it by descending primary key one record at a time. (Larry:) one final comment: if you have reasonably large files, you should get familiar with the RMS utilities and how to tune files. It can pay big dividends in disk space and especially in performance.

Kevin Cullan, Vitamix Corp: *I have a situation where I have a lot of zoned numeric fields, they are used as key values, and DTR doesn't seem to recognize them as keys. I've been forced to define them as unsigned numeric, and that isn't what I really want to do. Is there some way around this problem? I realize that if they really are zoned numeric they won't be sorted in the order normally expected.* [Editor's note: the sign flag, which you don't see when the number is printed, can change the apparent sort order if the data is taken as characters or numbers without sign flags.] (Question from panel: is there a reason why you have to use zoned numeric?) *I have to use zoned numeric for compatibility with another language. There does not seem to be any reason why it shouldn't work.* (Question from panel: I assume you define the fields in DTR and then do a DEFINE FILE KEY = field: have you ever looked at the FDL description of the file to see what data type DTR is using?) *I don't create the file with DTR I do it with FDL, and the field is defined as STRING.*

(Bart:) That might be the problem. DTR may be recognizing your data as numeric and will not be sending it out to RMS as a string. As a first shot, I'd create a file with DTR with a zoned numeric field as a key, and then look at the file DTR creates to see what data type it is expecting for that field. What we usually recommend in these cases is to always define the file with DTR first, to insure that the keys have the proper data type, and are in the correct location. Then, if you want to, by all means use FDL to optimize the file for your application, but use DTR first to get the keys in the right place.

Kevin Cullan: What I've done is to use unsigned numeric, and that does work.

(Chris:) The other thing to do is to run DTR in DEBUG mode and do an RSE to see if DTR identifies that as an indexed read or not. [Rather than run the DTR image directly, issue the command DEBUG SYS\$SYSTEM:DTR32 using the appropriate directory and file name for your DTR image. After DEBUG issues some informational messages, enter the command GO and you will see the normal DTR prompt. From that point you use DTR as you normally would, but you will receive extra informational messages about such things as the keys being used for record retrieval.]

Kevin Cullan: I have not done a DEBUG, but from the time it takes I believe it is not doing indexed reads. It takes hours if the field is zoned versus

seconds for unsigned numeric. I can see how zoned variables could cause problems, and am wondering if they are not allowed as keys. I'd prefer DATATRIEVE to do what I ask, whether it thinks it's a good idea or not.

Actually, DTR usually allows you to do what you want, even if it is a poor idea, such as crosses over fields which are not keys. We may have to pursue this problem in the campground.

Wayne Heiderman, (?) International: *Do you have any recommendations on bucket sizes?*

(Bart:) On the PDP-11, use the smallest bucket size that will hold the data. On the VAX, increasing bucket size may or may not get you anything. If you are reading through the file sequentially, a larger bucket size may help you; if you are retrieving a particular record and are likely to immediately use the next record(s) then an increased bucket size may help; if your accesses are more random, and are scattered throughout the file, an increased bucket size won't help and may even hurt a little. If you want to get fancy, you can use FDL to examine the file definition: a larger bucket size which will flatten the index structure usually improves accessing the file, but it's usually a case of try it on your particular data file and application and see what happens. (Chris:) I'd also suggest you look for efficient use of the buckets. If you have a 600 byte record, then you don't want a 2 block bucket size, as you may only have one record per block and about 400 bytes unused space. A bucket size of 3 might be better: you don't want a large amount of unused space. You do want some extra space for the RFAs mentioned earlier, otherwise if you delete one record there may not be enough space for a new record and RFA and again space will be wasted, so do leave a little space for future deletes.

Frank Schipani, Emery University. *I manage a comparatively large database with DTR and due to various changes in the database I have to use CONVERT a great deal. One feature I miss that is available on other systems that I don't have in CONVERT is a field description language, so if I am going to insert a field in the middle of a record is to go through an involved procedure.* [Editor's note: a process using RMS utilities and sometimes COBOL was described.] *Is there any prospect of enhancing CONVERT.*

(Bart:) I don't think a data restructure facility will be added to CONVERT. However, if you are doing things like that often you may want to use the SORT utility, which does have a field description language and a way to restructure records. SORT might be faster than DTR though not as fast as CONVERT.

Frank Schipani: SORT is going to have to sort a very large file that doesn't really need to be sorted.

(Bart:) I believe that MERGE has the same field description capability, and I believe there is also a qualifier that you can use to tell SORT that the data is close to being sorted already, or use /STABLE to tell it that it has less work to do.

(Chris:) I guess I don't understand why you would have a problem doing a redefine as was described [in the first question].

Frank Schipani: it took two weeks as a background job.

(Chris:) Were you going from an indexed file to an indexed file? (Yes) OK, that is your problem. You should try going indexed to sequential to restructure the data.

Frank Schipani: That's true but then it would still take overnight to do that. There should be a faster way: for example, if I use COBOL to unload to a sequential file, the same file that takes [] hours in DTR takes 8 minutes.

(Chris:) what you can also do is write a special record definition which has only three large fields. One goes from the beginning of the record to the place the new data will be inserted, then a field for the new space, and a third to cover the data to the end of the record. This will increase speed by reducing the time DTR takes for a field by field copy. This lets you stay in DTR rather than having to write in COBOL.

Frank Schipani: that's true.

(Bart:) also, when you define your sequential file, did you also pre-allocate space in the output file? When you move very large amounts of data, you want to go from one disk to another if possible, and you want to open the file with the space allocated all

at once at the beginning rather than having to get it in chunks while the program is running. Otherwise, going from an indexed file to a sequential file should be about as fast in DTR as any other language unless you are doing a lot of fancy things with the fields.

Frank Schipani: I think that MOVE CORRESPONDING works faster than whatever DTR is using. I think now that the difference was more like 1/2 hour versus 8 minutes.

Lisa Axelrod, EG&G.: *I have a record definition I cannot change, and it has a history element which occurs 18 times, and has numeric and non-numeric information. I can't find any way to get "down there" unless FIND a unique record; and if they are all blank, how do I get down to a particular one?*

(Chris:) what you are trying to do is find the first blank record?

Lisa Axelrod: no, I have 18 occurrences, and depending on what is happening, I may have to get to the nth element to store something in it.

(Chris:) there are several things you can do. Are you in a procedure, or doing this interactively? Is it within a BEGIN-END block so you can't do FIND and SELECT?

Lisa Axelrod: I'll do it any way I can.

(Chris:) [To do an interactive query,] you find the record you want to operate on, and SELECT it. Then you do a FIND giving the name of the list (the name on the OCCURS clause). This gives you what appears to be a collection of all of the data in the occurs clause. You can then do a PRINT ALL., and if you want the 5th one say SELECT 5, and you can then print, modify, store, etc.

(Don Stern:) if you check the April 1986 issue of the newsletter, there was some "magic" printed from the Anaheim symposia on using running counts to get to a particular occurs clause record. (Chris:) that is better for procedures, rather than interactive queries which may be better with FIND / SELECT. (Bart:) The trick is to get one particular record, then treat the OCCURS clause as if it was another domain: that is what you have to remember. There was also an earlier newsletter that shows another method of getting to OCCURS clauses. [The two

articles referred to are both transcriptions of Wombat Magic sessions. The first was by Diana Washburn, doing "subscripting" or "indexing" an OCCURS, printed in the *Wombat Examiner*, Volume 6 Numbers 1&2, April 1985 (combined Winter/Spring issue), pages 30 to 35. The second was by Rowland W. Fox, Doing a Modify to the nth Field in an OCCURS, and is on page DTR-20 of the Combined Newsletter for April 1986, Volume 1, Number 8.]

Doak Bane (?) Coleson Inc. *I have a problem getting a date field defined as a date field in FMS. It does not seem to go into DTR as a date field. Is there some way around this?*

(Chris:) The simple thing is that you cannot use FMS date fields with DTR date fields. The field in FMS must be character.

Doak Bane: I found that out. I had defined a date field and found that FMS sent more characters than I had defined. (Was the form defined for a domain or using a DISPLAY_FORM?) It used DISPLAY_FORM.

(Chris:) The simple thing to do then is to set the edit string in DTR to the way you want it, then use a character field in FMS (if you want MM/DD/YY then the FMS field must be 8 characters), then you can do a PUT_FORM fms_field = FORMAT(date_field). This will force DTR to output the date to FMS as specified by the edit string. When you read it in, do a field = GET_FORM fms_field, which DTR will read as ordinary characters and convert to a date as normal. It's the same with TDMS [as it is with FMS].

Wayne Heiderman: *I'm dumping some information from an IBM mainframe, and it has a dollar figure with an overpunch on the last digit. Is there an easy way to get rid of it?*

(Joe:) That particular problem has been addressed, and has been published in the newsletter. Basically, the method is to use define the last character of the field (the one that has the sign) separately from the rest of the numeric field, and use tables to convert the character that appears here to extract the sign and value for that field. You then use a COMPUTED_BY field or equivalent to reconstruct the number by multiplying the leading digits by 10 (the least significant digit was stripped off, remember), multiply by the sign (+ 1 or -1) from the sign table, and add the least significant digit from the conversion table. [An alternate method used by Bob Lott to convert DEC

COMP_5 fields to IBM format using a CHOICE statement to change the least significant digit was published in the *Wombat Examiner*, Volume 5, Number 4, Fall 1984, pages 46 and 47.]

Paul Merbeme, Contro Co.: *We have several records which were built with DIBOL and which have many dates in them. How do we convert that to DTR?* (Question from Chris: what format are the dates in? Are they 6 characters, or 8 characters?) *I believe they are 6 character.* (Chris: like two digit month, two digit day, etc.?) *Yes.*

(Chris:) it turns out that DTR is smart enough to do this during file restructuring. Your old file record would be PIC X(6), the new file would have USAGE DATE, both fields have the same name, if you do an NEW = OLD [as in the first question] DTR will do the date conversion. (Larry:) in some cases where the dates in the old files are not matching up correctly, for example they go year, month, day, then you can separate them into separate fields, and re-shuffle them around [as in a COMPUTED_BY], and as long as you can put it into something that looks like one of the DTR date formats it will do the conversion. Sometimes you have to put in the "/" characters, etc., but you should be able to make it work.

Paul Merbeme: I've tried that with EXTRACT to try flipping around the fields, with some success.

(Chris:) do you have the capability of changing that data file?

Paul Merbeme: no, otherwise the DIBOL program has to change. I don't have any problems printing out the dates, [but I want to use the date field in selection expressions to find certain dates].

(Chris:) presently, a COMPUTED_BY field cannot have a USAGE DATE. (Joe:) do you have the ability to add an additional field at the end of the record, so that DTR could read the file and put in the data in USAGE DATE format in the extra field?

Paul Merbeme: No, I can't change the file. I could copy the whole thing over, but it would take a lot of space and time.

(unidentified:) why not just define the field in DTR with an edit string of "MMDDYY"?

(Joe:) it would not be a DATE data type, to be used for proper date comparisons. DTR treats the DATE data type in a very special manner. There is a difference between a data type of date, and numeric values corresponding to a month, day, and year.

(Bart:) It sounds like you want to prompt somebody in DTR for a date, and then find the one in your file. (We want a starting and ending date). Doing a BETWEEN on data types other than USAGE DATE will be rather difficult.

Paul Merbeme: I don't remember the exact solution, but basically I changed the date to strings.

(Chris:) you can also convert the date to a numeric value using year * 10,000 + month * 100 + day, to yield a numeric value which will sort dates properly. Then your starting and ending dates would also be PIC 9(6), and this format will compare properly.

Paul Merbeme: I have tried this.

(Bart:) I believe that if the COMPUTED_BY is done with the FN\$--- function that converts an ASCII string to a date type, that perhaps that will yield the proper COMPUTED_BY field type. I have used this to convert dates. The format must be DD-MMM-YYYY, and you may need a table to convert the numeric month to JAN, FEB, MAR, etc. (Don Stern:) If you are willing to do the conversion in a FOR loop, you can declare a temporary variable of USAGE DATE outside the FOR loop, and then use FN\$STRING_EXTRACT to put it back into a date form, then do the math on the declared variable. (Chris:) we've got it: FN\$DATE does the conversion from ASCII characters to 64 bit date type, so you can change your MM/DD/YY to a USAGE_DATE. (Bart:) I've used it, and I'm fairly certain the date input has to be DD-MMM-YYYY. I had to do a field = FN\$DATE(day|"-"|month via mon_tab|"-"|year) to get the conversion to work. The result was a standard USAGE DATE field that did all of the normal DTR date functions and searches. (Joe:) I think the converted month must be upper case. [Converting the numeric month via a table yields an upper case month.]

Ken Fox, Shearing Plough (?). *I heard earlier this week that VMS was allowing shared access to sequential files. Does this mean we should expect DTR to follow in the near future?*

Andy Schneider, DEC: [Editor's note: though not scheduled as a panelist, the person primarily responsible for the VAX-DATATRIEVE product obviously could not resist the opportunity to answer the question.] Please, I repeat, *please*, don't use shared sequential file access with DTR if you are going to write to the file: let me explain why.

DTR, when it connects to a sequential file, connects to END-OF-FILE. If you connect to END-OF-FILE, and your neighbor [sharing the file] connects to END-OF-FILE, and you are writing records to the file, and then your neighbor decides to add records, he is trying to add to his END-OF-FILE, but your END-OF-FILE is past that so he is going to write over your records because the END-OF-FILE markers don't get updated [from one user to another]. We are frantically looking at the correct method to do this, working with the RMS developers to determine what the right method is. For now, the watchword is CAUTION. Version 3.4 warns the users of the pitfalls. The problem has to be addressed by RMS as well as DTR.

(unidentified:) in the current version of DTR, will it even let you have more than one person write to a shared sequential file?

Andy Schneider: sure. What DTR does when you say READY domain SHARED, DTR does not look at what type of file it is, it simply passes the request off to RMS. Up until VMS V4.4, RMS did support shared writes to sequential files if they were 512 byte fixed length records, and DTR will work with that file. The rejection comes from VMS.

(unidentified:) on shared files, if a file is locked by another user, is the only switch I have to set the DTR WAIT_LOCK switch, or are there also CDD and VMS switches I have to set for my users to wait before access?

Andy Schneider: you're talking about records in an RMS file? (Yes) Yes, WAIT_LOCK is the only switch you have to set to have the DTR user stall until the record is free. (Not CDD?) No, you are not in the CDD at that point. This has no effect on your data files. [There were some comments about dictionary sharing, which was lost as the tape changed sides.] If you set CDD wait, then it should not affect multiple users executing procedures, but will effect users if one person is editing a procedure.

(unidentified) *It was stated at a previous session that SELECT is not supported within a BEGIN-END block. I'd like a clarification of what "not supported" means, because in an application I wrote I have several people sharing update access to a single record in an indexed file (to increment a counter), and in order to release the record so the other users can get to it it is necessary to de-select the record. SELECT NONE apparently works.*

Andy Schneider: SELECT NONE does work, but let me explain. There are two restrictions in DTR that people seem to stumble over every 5 minutes: the restriction that you can't use a FIND and you can't use a SELECT within a BEGIN-END block. Now, if you try to do it, DTR won't tell you "YOU CAN'T DO THAT!". The reason we say it isn't [supported] is because of the way DTR parses, compiles and executes a BEGIN-END block. DTR treats the entire BEGIN-END block as one statement, not a bunch of separate ones. If you have a FIND within a BEGIN-END block it will work; for example:

```
BEGIN
      FIND YACHTS
END
```

will work. However, if you have FIND --- and SELECT in the BEGIN-END block, it won't work, because when it goes through and tries to generate the context for the SELECT, the FIND hasn't taken place, UNLESS: there was a FIND done outside the BEGIN-END block, and in that particular case the context for the SELECT is driven off of that FIND. As long as that FIND is for the domain the SELECT is supposed to be on, that's OK. If the SELECT is in a loop, you will probably be selecting the same record over and over again, I believe, [because the context has already been set outside of the loop].

(user:) That's exactly what I want to do: I want to SELECT the same record over and over again, and de-select the same record.

Andy: well, in that case it's not a restriction. We designed DATATRIEVE just for you! [laughter] Next question!

Larry: I think you should realize that at nearly every symposia someone comes up with something that's never been done, or we never ever conceived of being done in DTR, and I think you just blew Andy's mind for this symposia.

Andy: any other new features you'd like us to put in like that? [more laughter]

(user:) *I'd also like compiled procedures.*

Andy: I was probably supposed to talk about this at some time. Any of you who have seen wishlists in the past have seen our response, which is, "that's a good idea, we will consider it for the future".

Well, we spent about a month looking at compiled DTR, and what it would mean to compile DTR language. With the current architecture of the product, there would be so many restrictions that READY YACHTS might not even work properly, because of it's flexibility and the point in it's path at which it resolves certain specifications. We investigated at a gut level how many man-months it would take to re-design DTR, and I won't give the figure but it's very large. At this stage in DATATRIEVE's stable life, and I say stable because we put in a lot of effort to make DTR a stable product, without bugs, it just doesn't seem to really work [to make it compile].

(user:) what about the intermediate stages in the parse/compile/execute process? Does it have to be stored at the text level?

Andy: sometimes, maybe. There are certain things we do opening the dictionary and parsing things (described in the internals session), and it turns out that the earliest stage we could break the process and store the results and guarantee that it would work is at the lexical analysis stage, which is essentially where it is now in the CDD. We could find some simple cases where it could work at a later stage, but you couldn't do anything with database products, forms products, or any distributed work. This eliminates a lot of functionality from DTR. We are not throwing the idea away: I can fairly safely say that compiling DTR (the language) is something you will not see. The concept of compiling an entire application is something we are very seriously looking at for whatever else may come down the pike in this area.

Record Definition Tutorial

B. Z. Lederman
2572 E. 22nd St.
Brooklyn, N.Y. 11235

Abstract

This session will supply examples and suggestions which go beyond the material in the Datatrieve manuals, and show how various types of problems may be solved using the options available within the record definition. The material will include some comparisons between different approaches to the same problem, the use of VIEWS (which are created from record definitions), and methods of transferring data from one domain to another, which also depends in part upon record definitions.

It is not unreasonable to state that the record definition is the foundation of any Datatrieve application, as it is the reference by which all data is stored and retrieved. Therefore, the first rule for any application is:

KNOW YOUR APPLICATION

from which immediately follows:

KNOW YOUR DATA

I find that the best way to work out record definitions is with two very simple pieces of equipment: a pencil, and a piece of paper marked off in squares such as graph paper, or a printer form layout sheet, or CRT display form, or an old coding sheet. By marking off the fields, using one square per byte, the number of data items, the length of each field, and it's alignment and relationship to other fields are easily determined. This is especially important with the REDEFINES clause, which will be shown later.

Often, an unsuspected benefit of taking an existing manual operation and implementing it on Datatrieve is that the people involved must sit down and figure out exactly what pieces of data they are dealing with, and in what manner: this is often the first time anyone actually does this, and they are often surprised by the amount of data involved.

Some applications move onto Datatrieve almost automatically. If you are using a preprinted form (and almost every company has some sort of printed form for orders, absence reports, pencil requisitions, etc.) then one can simply copy the fields into the record definition: there are cases of new users moving applications like this in one day. If your records are not as well organized, then you must analyze them yourself. If you are using Datatrieve to read an existing file created by some other program, then it is necessary to obtain the file record layout and follow it.

Keep in mind that, while it is nice to get a good record definition at the beginning, it is always possible to define a new domain and record and read the data from the old domain to a new one, so if you have 10,000 records stored, and find you need to add a field, don't panic. Examples of this will be given.

While one could use ADT or follow the simple examples in the manuals and develop many useful applications with Datatrieve, there is a much wider range of applications which may be addressed with a few simple techniques. For example, suppose the YACHTS domain was being used in a show room, where the customers are allowed to look up data at a terminal, but the seller doesn't want the price to appear. Using the simplified record definition for YACHTS, here are two possible solutions.

Original Definition	Second Definition	View
01 BOAT. 06 BUILDER PIC X(10). 06 MODEL PIC X(10). 03 SPEC. 06 RIG PIC X(6). 06 LOA PIC XXX. 06 DISP PIC 9(5). 06 BEAM PIC 99. 06 PRICE PIC 99.	01 BOAT. 06 BUILDER PIC X(10). 06 MODEL PIC X(10). 03 SPEC. 06 RIG PIC X(6). 06 LOA PIC XXX. 06 DISP PIC 9(5). 06 BEAM PIC 99. 06 FILLER PIC XX.	DEFINE DOMAIN LOOK OF YACHTS USING 01 LOOK OCCURS FOR YACHTS. 03 BUILDER FROM YACHTS. 03 MODEL FROM YACHTS. 03 RIG FROM YACHTS. 03 LOA FROM YACHTS. 03 DISP FROM YACHTS. 03 BEAM FROM YACHTS. ;
;	;	;

The definition on the left is for the whole domain which the show room owner will use, and is the short definition given in the manual in the optimization chapter. PDP-11 users who are short of pool space should look at this chapter, and compare the short definition, which uses much less pool space by having fewer clauses, with the definition created by the installation package. The customers could use the view on the right, which does not have the price, or a second domain using the record definition in the center could be used to access the **same file** as is used for YACHTS. This shows two useful features. First, it is possible to have more than one domain access the data in a single file: this makes it possible to look at the data in more than one way, with more than one record definition. The only restriction is that the user must document the domains accessing each file so that, if it is ever necessary to change a file, the domains to be affected will be known. There is also an important difference between the domains and the view: you cannot store or erase records in a view, but you can do all operations on the second domain. The limitation on views can be bad or good, depending upon the application: in this example, you probably would not want customers to add or erase records, so using a VIEW would be one way of preventing this.

The second domain shows the use of the special field type FILLER to "skip" over data in a record. The data is still there, and may be accessed by the original domain, but not by the second domain: if you also protect the record definition itself to be execute but not read, the user will never see the filler field (it doesn't appear in SHOW FIELDS), and will not know there is data there. This may be extended for use in 'hiding' fields.

```
01 CUSTOMER.
  03 ADDRESS.
    06 STREET PIC X(10).
    06 CITY PIC X(10).
    06 STATE PIC X(2).
    06 ZIP PIC 9(5).
  03 FILL-ENG.
    06 FILLER PIC X(10).
  03 ENG REDEFINES FILL-ENG.
    06 ENGINEER PIC X(10).
```

If you normal command PRINT you get:

```
STREET      CITY      STATE  ZIP
2572 E. 22  BROOKLYN  NY     11235
```

but if you say PRINT ADDRESS, ENG the result is:

```
STREET      CITY      STATE  ZIP  ENGINEER
2572 E. 22  BROOKLYN  NY     11235 LEDERMAN
```

Thus the ENGINEER field is always available, but will not print out unless specifically asked for. This can also be a pool saving technique for very large records, or may be used to control access to information where several users must access the same file by doing something like this:

```
01 ALL-DEPT-REC.          01 DEPT-A-REC.          02 DEPT-B-REC.
  03 DEPT-A.              03 DEPT-A.              03 FILLER PIC X(5).
    06 BUDGET PIC 999.    06 BUDGET PIC 999.
    06 MANAGER PIC XX.   06 MANAGER PIC XX.
  03 DEPT-B.              03 FILLER PIC X(10).   03 DEPT-B.
    06 BUDGET PIC 999.   ;                               06 BUDGET PIC 999.
    06 MANAGER PIC XX.   ;                               06 MANAGER PIC XX.
  03 DEPT-C.              ;                               03 FILLER PIC X(5).
    06 BUDGET PIC 999.   ;                               ;
    06 MANAGER PIC XX.   ;                               ;
;                               ;
```

This is a very small example, but it shows how a single file may be accessed by one domain having access to all fields, and by several other domains, accessing only some of the data. Each of the smaller domains can read and write only their own data, and the big domain could be used for report giving all of the data. This is an alternative to having separate domains for each department, and using a view to tie them together for reports. (For PDP-11 and PRO users, each of the smaller record definitions uses less pool than the big definition, allowing more complicated procedures to be used (or more sort space, etc.). If a record definition is very large (hundreds of bytes), then the only way to access it usefully in Datatrieve-11 may be to have more than one domain each access a portion of the record.)

Another approach to the same problem would be to use a VIEW. First, a definition may be given for the domain which holds data for all departments.

```
01 BUDGET-REC.
  10 DEPARTMENT PIC XX.
  10 PROJECT PIC X(10).
  10 AMOUNT PIC 9(6)V99 EDIT-STRING $$$$,$$$$.00.
  10 MANAGER PIC X(10).
;
```

The person in charge of budgets would have full access to this domain, and so could access all of the data.

Each individual department would have their own VIEW defined like this:

```
DEFINE DOMAIN AA-BUDGET OF BUDGET USING
01 AA-BUDGET OCCURS FOR BUDGET WITH DEPARTMENT = "AA".
    10 PROJECT FROM BUDGET.
    10 AMOUNT FROM BUDGET.
    10 MANAGER FROM BUDGET.
;
```

This definition should be protected so that the department can execute it, but not read or modify it, otherwise they might want to change the definition to allow access to other departments. Because the selection criteria is fixed, they will see only their own department's data. This configuration would be of greatest use when the different departments must read the information, but only the central controller will enter or erase it.

Another very useful feature of the REDEFINES clause is that it allows one to look at the data in a domain in more than one way within a single domain. An application for this could be a file which has more than one record type in a single file. The author does not recommend this for new applications, but there may be existing files set up like this (COBOL and RPG are often the source) which one would like to access with Datatrieve. Consider a file with data that looks like this:

Key Type

0001 N	Lederman	Bart	Z	! Name information
0001 A	2572 E 22nd New York	NY 11235		! Address information
0001 P	38 DPG 2222 Distributed Proc			! Business information
0001 T	212-555-5555	718-555-5555		! Telephone numbers

A possible record definition is:

```
DEFINE RECORD MULTI-REC
01 MULTI-REC.
    03 KEY PIC 9999 EDIT-STRING ZZZ9.
    03 TYPE PIC X.
    03 NAME-PAGE.
        06 LAST PIC X(14).
        06 FIRST PIC X(12).
        06 M PIC X.
        06 N PIC X.
    03 ADDRESS-PAGE REDEFINES NAME-PAGE.
        06 STREET PIC X(11).
        06 CITY PIC X(10).
        06 STATE PIC XX.
        06 ZIP PIC 99999.
    03 PERSONNEL-PAGE REDEFINES NAME-PAGE.
        06 FLOOR PIC 99 EDIT-STRING Z9.
        06 SECTION PIC XXX.
        06 CLOCK PIC 9999.
        06 DEPARTMENT PIC X(19).
    03 TELEPHONE-PAGE REDEFINES NAME-PAGE.
        06 BUSINESS PIC X(10)
            EDIT-STRING XXX-XXX-XXXX.
        06 HOME PIC X(10)
            EDIT-STRING XXX-XXX-XXXX.
        06 FILLER PIC X(8).
```

;

The first part of the definition is for the fields which do not change (the key and the type, which are common to all records). The next part is the definition for the first record, the name fields. Because this is the first definition (highest in the hierarchy), it is used by default when accessing the data. If the data is printed, the result is:

```

KEY   TYPE      LAST           FIRST           M N
1   N   Lederman       Bart            Z
1   A   2572 E 22ndNew   York NY112 3 5
1   P   38DPG2222Distr  ributed Pr o c c
1   T   21255555557185  555555

```

Notice how all records have been printed as if they were NAME-PAGE as this is the first group in the hierarchy, but because I have the redefined fields, I can also access the data in different ways. Each redefines has it's own group name, which makes access much easier as I can specify a group name for one whole page, and note that a redefines must never be longer than the original field and/or group. In other applications, you should be certain the longest group comes first, or that you fill the first group to be as long as the longest group. It is acceptable for the redefines to be shorter than the original field, but I prefer to fill all groups in for my own reference. In this case, the telephone data is shorter, and the additional length is made up with FILLER. With the redefines, a simple procedure will access the data correctly.

```

DEFINE PROCEDURE PRINT-MULTI
READY MULTI
FOR MULTI BEGIN
    IF TYPE EQ "N" PRINT NAME-PAGE
    IF TYPE EQ "A" PRINT ADDRESS-PAGE
    IF TYPE EQ "P" PRINT PERSONNEL-PAGE
    IF TYPE EQ "T" PRINT TELEPHONE-PAGE
END
END-PROCEDURE

```

When this procedure is invoked, the data prints like this:

```

          LAST           FIRST           M N
Lederman       Bart            Z

          STREET        CITY          STATE  ZIP
2572 E 22nd New York      NY     11235

FLOOR SECTION CLOCK      DEPARTMENT
38      DPG      2222  Distributed Proc

          BUSINESS      HOME
212-555-5555  718-555-5555

```

If you have more than one set of records, the following sets will not have headers when they print out (this is the normal way the Datatrieve PRINT command behaves) but the data will print out using the correct field definitions. The same technique may be used to select the proper page for storing, and so on. Individual fields of each page may be accessed simply by using the name of the field, at any time: Datatrieve will go through the record hierarchy, as it would for any other domain, to resolve the field references.

An alternative to the procedure is to define two separate domains for the data.

```
DEFINE RECORD BASE-REC
01 BASE.
    03 KEY PIC 9999 EDIT-STRING ZZZ9.
    03 TYPE PIC X.
    03 LAST PIC X(14).
    03 FIRST PIC X(12).
    03 M PIC X.
    03 N PIC X.
;
DEFINE DOMAIN BASE USING BASE-REC ON MULTI.SEQ;

DEFINE RECORD OTHER-REC
01 OTHER.
    03 KEY PIC 9999 EDIT-STRING ZZZ9.
    03 TYPE PIC X.
    03 ADDRESS-PAGE.
        06 STREET PIC X(11).
        06 CITY PIC X(10).
        06 STATE PIC XX.
        06 ZIP PIC 99999.
    03 PERSONNEL-PAGE REDEFINES ADDRESS-PAGE.
        06 FLOOR PIC 99 EDIT-STRING Z9.
        06 SECTION PIC XXX.
        06 CLOCK PIC 9999.
        06 DEPARTMENT PIC X(19).
    03 TELEPHONE-PAGE REDEFINES ADDRESS-PAGE.
        06 BUSINESS PIC X(10) EDIT-STRING XXX-XXX-XXXX.
        06 HOME PIC X(10) EDIT-STRING XXX-XXX-XXXX.
;
DEFINE DOMAIN OTHER USING OTHER-REC ON MULTI.SEQ;
```

Now, I can define a view of these two domains to bring all of the separate records together into what will look like one single record:

```
DEFINE DOMAIN MUL OF BASE, OTHER USING
01 MULT OCCURS FOR BASE WITH TYPE EQ "N".
    06 LAST FROM BASE.
    06 FIRST FROM BASE.
    06 M FROM BASE.
    06 N FROM BASE.
03 ADDRESS-PAGE OCCURS FOR OTHER WITH
    TYPE EQ "A" AND OTHER.KEY=BASE.KEY.
    06 STREET FROM OTHER.
    06 CITY FROM OTHER.
    06 STATE FROM OTHER.
    06 ZIP FROM OTHER.
03 PERSONNEL-PAGE OCCURS FOR OTHER WITH
    TYPE EQ "P" AND OTHER.KEY=BASE.KEY.
    06 FLOOR FROM OTHER.
    06 SECTION FROM OTHER.
    06 CLOCK FROM OTHER.
    06 DEPARTMENT FROM OTHER.
03 TELEPHONE-PAGE OCCURS FOR OTHER WITH
```

```

        TYPE EQ "T" AND OTHER.KEY=BASE.KEY.
    06 BUSINESS FROM OTHER.
    06 HOME FROM OTHER.

```

```
;
```

The reason for the two domains is to be able to use the key field to tie together the appropriate separate records. The BASE domain will occur once for each group of associated records, and the KEY field will be used to retrieve the other records of the same group. When this view is readied and printed, it looks like this (I have added a second set of data records to show that the view works properly):

LAST	FIRST	M N	STREET	CITY	STATE	ZIP	FLOOR	SECTION
Lederman	Bart	Z	2572 E 22nd	New York	NY	11235	38	DPG
2222	Distributed Proc		212-555-5555	212-555-5555				
Hackinbush	Hugo	Z	11 Julius	Hialeah	FL	33999	13	MRX
1313	Sales & Promotion		305-555-3131	305-555-1476				

It appears wrapped around here as there are only 80 columns on this page, but on 132 column paper, all fields print out with their headers. This view would be very useful in "flattening" the data record so it could be processed as other domains are.

Another use of the redefines can be for break fields.

```

DEFINE RECORD TTN-REC
01 TTN.
    03 PORT PIC 999.
    03 BREAK REDEFINES PORT.
        06 B1 PIC 99.
        06 FILLER PIC X.
    03 GROUP PIC 999.
    03 SWITCH PIC 9.
    03 TRUNK PIC 9999.
    03 COMMENTS PIC X(21).

```

```
;
```

Although PORT is an integer number, I am using the DISPLAY data type so I can redefine it as a two digit field. The reason can be seen when reporting the data.

```

DEFINE PROCEDURE RPT-TTN
READY TTN
REPORT TTN ON TTN.RPT
SET REPORT-NAME="Show Breaks with Redefines"
SET COLUMNS-PAGE=50
PRINT COL 1, PORT, COL 8, GROUP, COL 20, SWITCH,
    COL 24, TRUNK, COL 40, COMMENTS
AT BOTTOM OF B1 PRINT COL 1,
"-----"
END-REPORT
END-PROCEDURE

```

When this procedure is invoked, the resulting report is:

PORT	GROUP	SWITCH	COMMENTS
040	347	4 1154	88/89
041	347	4 1155	88/89
042	347	4 1156	88/89
043	347	4 1157	88/89
044	347	4 1417	88/89
045	347	4 1440	88/89
046	347	5 4521	88/89
047	347	5 4522	88/89

050	347	5 4523	88/89
052	131	7 5311	TYPE 0
053	131	7 5312	TYPE 0
054	131	7 5313	TYPE 0
055	131	7 5314	TYPE 0
056	131	7 5315	TYPE 0
057	131	7 5316	TYPE 0

060	131	7 5317	TYPE 0
061	131	7 5320	TYPE 0
062	131	7 5321	TYPE 0

As may be seen, by having a field which acts on the first two digits of PORT, it is possible to put in a break line every 'n' entries, something which would be difficult otherwise. (Incidentally, the ports are numbered in octal, which is why there is no port 048 or 049, but the system works just as well in decimal.) Not shown is a LINES-PAGE command: if you want groups of records to print out on successive pages with the breaks aligned, you will have to set the number of lines per page to match the group breaks. It may be noted that there is no SORTED BY clause in the report statement: it is not necessary to sort the data if it is already in the proper order, as it will be if a sequential file is reported in its present sequence, or if an indexed file is reported in the order of its primary key. (One of the uses for indexed files can be to keep data ordered.) If there is no SORTED BY clause, Datatrieve will issue a warning that unsorted records are being reported, but will then report and allow a break on any field: if there is a SORTED BY clause, then only fields which were sorted can have breaks, and in this case I want the report in the order of PORT, not the order of B1. Reports also come out faster if you don't have to sort the domain first, and exhaustion of sort pool space is avoided.

A commonly used feature is variable length records, as in the FAMILIES domain. While certainly useful, variable length records have some draw backs, including more difficult access to the fields in the variable length portion, and having to know the maximum number of variable occurrences when defining the domain. There is an alternative approach using two files and a view. First, for comparison, is a shortened record definition for the FAMILY domain.

```

DEFINE RECORD SHORT-FAMILY-REC
01 FAMILY.
    03 FATHER PIC X(10).
    03 MOTHER PIC X(10).
    03 NUMBER-KIDS PIC 99 EDIT-STRING Z9.
    03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER-KIDS.
        06 KID PIC X(10).
        06 AGE PIC 99 EDIT-STRING Z9.

```

;

```

DEFINE DOMAIN FAMILY USING SHORT-FAMILY-REC ON FAMILY.DAT;

```

The alternative uses one domain for the fixed data, and one for the variable occurrence data, with one field in common to tie the two together.

```

DEFINE RECORD PARENT-REC
01 PARENT.
    03 KEY PIC 999 EDIT-STRING ZZ9.
    03 FATHER PIC X(10).
    03 MOTHER PIC X(10).

```

;

```

DEFINE RECORD OFFSPRING-REC
01 OFFSPRING.
    03 KEY PIC 999 EDIT-STRING ZZ9.
    03 KID PIC X(10).
    03 AGE PIC 99 EDIT-STRING Z9.

```

;

```

DEFINE DOMAIN PARENT USING PARENT-REC ON PARENT.DOM;

```

```

DEFINE FILE FOR PARENT KEY=KEY(NO DUP);

```

```

DEFINE DOMAIN OFFSPRING USING OFFSPRING-REC ON OFFSPRING.DOM;

```

```

DEFINE FILE FOR OFFSPRING KEY=KEY(DUP);

```

The two domains are then connected with a view:

```

DEFINE DOMAIN HOUSEHOLD OF PARENT, OFFSPRING USING
01 HOUSEHOLD OCCURS FOR PARENT.
    03 FATHER FROM PARENT.
    03 MOTHER FROM PARENT.
    03 KIDS OCCURS FOR OFFSPRING WITH OFFSPRING.KEY EQ PARENT.KEY.
        06 KID FROM OFFSPRING.
        06 AGE FROM OFFSPRING.

```

;

When printed, the HOUSEHOLD domain looks just like the FAMILY domain, without the NUMBER-KIDS field.

FATHER	MOTHER	KID	AGE
JIM	ANN	URSULA	7
		RALPH	3
JIM	LOUISE	ANNE	31
		JIM	29
		ELLEN	26
		DAVID	24
		ROBERT	16
JOHN	JULIE	ANN	29
		JEAN	26
JOHN	ELLEN	CHRISTOPHR	0
ARNIE	ANNE	SCOTT	2
		BRIAN	0
SHEARMAN	SARAH	DAVID	0
TOM	ANNE	SUZIE	6
		PATRICK	4
BASIL	MERIDETH	BEAU	28
		BROOKS	26
		ROBIN	24
		JAY	22
		JILL	20
		WREN	17
ROB	DIDI		
JEROME	RUTH	ERIC	32
		CISSY	24
		NANCY	22
		MICHAEL	20
TOM	BETTY	MARTHA	30
		TOM	27
GEORGE	LOIS	FRED	26
		JEFF	23
		LAURA	21
HAROLD	SARAH	HAROLD	35
		CHARLIE	31
		SARAH	27
EDWIN	TRINITA	ERIC	16
		SCOTT	11

This view can be readied, printed, reported, examined, etc. just like the FAMILY domain, but there are several important differences. First, because they are two separate domains, it is not necessary to know how many occurrences there are for the variable portion, and there is no limit to the number of variable records. Also, because there are two separate domains, it is possible to work on each portion separately, and protect each portion separately. We had an application where the fixed portion was the basic information on a communications circuit (the customer name, location, date due, and so on), and the variable portion was a record entered each time a workman attended to the circuit: the domains were protected so the workmen could read the fixed portion but never had write or modify access to it, but could write to the variable domain. This mixed protection cannot be done on a single domain with variable occurs. Another benefit is in Datatrieve-11, where pool space may be saved by readying only the portion required for a given application, rather than always having to use the larger single domain.

One drawback is that one cannot STORE to a view, but a simple procedure solves this:

```
DEFINE PROCEDURE STORE-FAMILY
DECLARE PROMPT PIC X.
DECLARE KEY-FIELD PIC 9999.
READY PARENT WRITE
READY OFFSPRING WRITE
KEY-FIELD = MAX KEY OF PARENT           ! Get the last key used
KEY-FIELD = KEY-FIELD + 1              ! make it the next key
WHILE *."Y to store a family" CONT "Y" BEGIN
    STORE PARENT USING BEGIN           ! Use the common key
        KEY = KEY-FIELD
        FATHER = *.FATHER
        MOTHER = *.MOTHER
    END
    PROMPT = *."Y if there are any kids"
    WHILE PROMPT CONT "Y" BEGIN
        STORE OFFSPRING USING BEGIN    ! Use the common key
            KEY = KEY-FIELD
            KID = *."Kid's name"
            AGE = *.AGE
        END
        PROMPT = *."Y for another kid"
    END
END
FINISH
RELEASE KEY-FIELD
RELEASE PROMPT
END-PROCEDURE
```

Note that a temporary field KEY-FIELD is used to obtain the key (either by prompting or as is done here, by obtaining the last key previously used), and this field is used to store the same value in both domains, thus insuring the fields will match. The prompting for repeats could easily be made more sophisticated than this simple example.

The field KEY is used to tie the two domains together, but need not appear in the final view. I have used the name KEY for this field to emphasize the fact that this is a good application for a keyed field in an indexed file. Note the condition on matching KIDS, where the two KEY fields are matched: if KEY were not a keyed field in domain OFFSPRING, then for every record in PARENTS, **every record in offspring would have to be searched for a matching field.** While it is possible for the variable length portion domain to be a sequential file, don't complain when it takes several hours to print out a few records. In the fixed length portion domain, it is not absolutely necessary for the KEY field to actually be a key, but making it so takes advantage of another aspect of keyed fields, that of preventing duplicates. If there were duplicates in PARENTS, than one set of KIDS would be assigned to more than one set of PARENTS, a confusing situation to say the least. In most applications, it is desirable for each set of variable records to be assigned to one fixed record, though the use of a VIEW will allow you to have multiple associations, one more possible advantage of the VIEW over the variable occurs for some applications. A similar use of key fields not shown here is the NO CHANGE attribute, which can prevent a field from being modified, regardless of what access or privileges a user has to that domain. This can be very useful in protecting data from accidental or deliberate modification.

The technique of tying two domains together with a matching field can be extended to do something else the occurs clause can not do: a domain (view) with more than one variable portion. To the previous definition, I will now add:

```

DEFINE RECORD ANIMAL-REC
01 PETS.
    03 KEY PIC 999 EDIT-STRING ZZ9.
    03 NAME PIC X(10).
    03 SPECIES PIC X(10).
;

DEFINE DOMAIN ANIMALS USING ANIMAL-REC ON ANIMAL.DOM;

DEFINE FILE FOR ANIMALS KEY=KEY(DUP);

DEFINE DOMAIN HOUSEHOLD OF PARENT, OFFSPRING, ANIMALS USING
01 HOUSEHOLD OCCURS FOR PARENT.
    03 FATHER FROM PARENT.
    03 MOTHER FROM PARENT.
    03 KIDS OCCURS FOR OFFSPRING WITH OFFSPRING.KEY EQ PARENT.KEY.
        06 KID FROM OFFSPRING.
        06 AGE FROM OFFSPRING.
    03 PETS OCCURS FOR ANIMALS WITH ANIMALS.KEY EQ PARENT.KEY.
        06 NAME FROM ANIMALS.
        06 SPECIES FROM ANIMALS.
;

```

The same rules about keys, etc. apply to this view. I have chosen to match the PETS with the PARENTS, but I could have added an additional field to the OFFSPRING domain if I had wanted PETS to be matched to OFFSPRING. When printed, the first few entries look like this:

FATHER	MOTHER	KID	AGE	NAME	SPECIES
JIM	ANN	URSULA	7		
		RALPH	3	GAYLORD	CAT
JIM	LOUISE		31	FREDDY	PARAKEET
		ANNE	29		
		JIM	26		
		ELLEN	24		
		DAVID	16	RAGMOMMA	DOG
JOHN	JULIE	ROBERT	29		
		ANN	26		
		JEAN			

It is not necessary for every entry in the PARENTS portion to have either OFFSPRING, or ANIMALS, or both, and neither OFFSPRING nor ANIMALS require the presence of the other. However, because of the way HOUSEHOLDS was defined, there must be a PARENT record if an OFFSPRING or ANIMAL record with the same key is to appear in the view. If PETS had been matched to OFFSPRING, then an OFFSPRING would have to be present for PETS to appear.

An alternative method of combining data from two domains in VAX-Datatrieve (and DTR-20) is to use the CROSS statement. Instead of the VIEW joining PARENT and OFFSPRING, I could use something like this:

PRINT PARENT CROSS OFFSPRING OVER KEY

This would give me:

KEY	FATHER	MOTHER	KEY	KID	AGE
1	JIM	ANN	1	URSULA	7
1	JIM	ANN	1	RALPH	3
2	JIM	LOUISE	2	ANNE	31
2	JIM	LOUISE	2	JIM	29
2	JIM	LOUISE	2	ELLEN	26
2	JIM	LOUISE	2	DAVID	24
2	JIM	LOUISE	2	ROBERT	16
3	JOHN	JULIE	3	ANN	29
3	JOHN	JULIE	3	JEAN	26
4	JOHN	ELLEN	4	CHRISTOPHR	0
5	ARNIE	ANNE	5	SCOTT	2
5	ARNIE	ANNE	5	BRIAN	0
6	SHEARMAN	SARAH	6	DAVID	0
7	TOM	ANNE	7	PATRICK	4
7	TOM	ANNE	7	SUZIE	6
8	BASIL	MERIDETH	8	BEAU	28
8	BASIL	MERIDETH	8	BROOKS	26
8	BASIL	MERIDETH	8	ROBIN	24
8	BASIL	MERIDETH	8	JAY	22
8	BASIL	MERIDETH	8	WREN	17
8	BASIL	MERIDETH	8	JILL	20
10	JEROME	RUTH	10	ERIC	32
10	JEROME	RUTH	10	CISSY	24
10	JEROME	RUTH	10	NANCY	22
10	JEROME	RUTH	10	MICHAEL	20
11	TOM	BETTY	11	MARTHA	30
11	TOM	BETTY	11	TOM	27
12	GEORGE	LOIS	12	JEFF	23
12	GEORGE	LOIS	12	FRED	26
12	GEORGE	LOIS	12	LAURA	21
13	HAROLD	SARAH	13	CHARLIE	31
13	HAROLD	SARAH	13	HAROLD	35
13	HAROLD	SARAH	13	SARAH	27
14	EDWIN	TRINITA	14	ERIC	16
14	EDWIN	TRINITA	14	SCOTT	11

By specifying the fields I want to print in the CROSS statement I could suppress the KEY fields, but I let them print out this time to show what is happening. Note that the way this particular CROSS statement was entered results in picking up only those parents who have at least one offspring. The CROSS statement can often be thought of as a short way to do a VIEW, and regarding it as such may help you see what it is doing. Note particularly what was said before about OFFSPRING having to be keyed for fast retrieval: you can see here that the second domain listed in the CROSS statement is taking the same place as the second domain in our VIEW example; therefore, it should also be keyed if the CROSS statement is to execute quickly. Just as with the view, if both domains are keyed it doesn't matter which comes first, but if only one is keyed it should given last in a CROSS statement for best results. Because the CROSS can be implemented in a single statement, it is easier to use during interactive sessions, and when you are investigating relationships between various domains and groups of data. If you find a particular

relationship which you expect will be used many times, you may want to convert your CROSS into a VIEW, as this will fix the relationship and you will be able to ready the domain and print records without having to remember what the joining conditions are.

Lest it be thought that I am completely against the use of the OCCURS clause, I will now show a good use for it: de-blocking records. With some other languages, a person will sometimes write more than one logical record or associated group of data into what the operating system and Datatrieve consider to be a single record. An example file containing some names, looks like this:

Dump of DB2:[300,3]DEBLOCK.SEQ;1 - File ID 4556,33,0
Record number 01. - Size 512. bytes

```
000000      W o l f                J F l y
000020      w h e e l
000040      M a h a t m a        K J e e
000060      v e s
000100      H u g o                Z H a c
000120      k i n b u s h
000140      O t i s                C r i
000160      b b l e c o b l i s
000200      S                        Q Q u a
000220      l e
000240      S a m                G r u
000260      n i o n
000300
000320
000340
000360
000400
000420
000440
000460
000500
000520
000540
000560
000600
000620
000640
000660
000700
000720
000740
000760
```

*** EOF ***

This data is all one physical record as there is no separation between logical records (each name), but it can be "de-blocked" with the proper record definition.

```

DEFINE RECORD DEBLOCK-REC
01 DEBLOCK-REC.
    03 FIELDS OCCURS 16 TIMES.
        06 FIRST PIC X(12).
        06 MI PIC X.
        06 LAST PIC X(14).
        06 FILLER PIC X(5).
;

```

Note that there is no "fixed" portion to this definition: everything is in the "variable" portion (inner list) of the record definition. When a domain with this record definition is readied and printed, it looks like this:

FIRST	MI	LAST
Wolf	J	Flywheel
Mahatma	K	Jeeves
Hugo	Z	Hackinbush
Otis		Cribblecoblis
S	Q	Quale
Sam		Grunion

The blank lines at the bottom are due to the fact that the entire inner list is printed by default, and the unused entries are filled with blanks. This record definition can be made to print better looking data with the redefines and computed by expressions.

```

DEFINE RECORD DEBLOCK-TWO
01 DEBLOCK-TWO.
    08 FIELDS OCCURS 16 TIMES.
        16 DUMMY.
            24 FILLER PIC X(32).
        16 N REDEFINES DUMMY.
            24 FIRST PIC X(12).
            24 MI PIC X.
            24 LAST PIC X(14).
        16 NAME PIC X(29)
            COMPUTED BY FIRST||" "||MI||" "||LAST.
;

```

Once again, space is allocated with filler, then redefined with the real fields to "hide" them. The COMPUTED BY may surprise persons who expect that only math functions can be used with this clause, but it works just as well with character strings. The effect of this is to squeeze down the three fields to a single field with blank spaces removed.

NAME
Wolf J Flywheel
Mahatma K Jeeves
Hugo Z Hackinbush
Otis Cribblecoblis
S Q Quale
Sam Grunion

Processing the name in this way removes one of the objections many people have to computer output; that it looks too "regimented" or too rigidly organized. Removing the blanks makes the names look more as they would when written or typed by a person: simple little things like this can significantly improve the appearance of a report. The use of the COMPUTED BY in the record definition allows us still to enter the first and last names separately so we can, for example, sort the data by last name or otherwise access the individual fields, and then use the concatenated name where desired.

It was stated that if a record definition is found not to meet the requirements of an application, it can be changed. The rules for changing an existing record definition (without having to change the domain or file used by the domain) reduce to a simple requirement: the length of the record cannot change. This condition results in the following rules:

Field names, group names, query headers, query names, and edit strings may always be changed, as they do not affect the length of the record, but watch out for any views which access that record definition: if you change a field or group name, any views which use that field must be changed so that their corresponding group and field names match.

Query headers, query names, and edit strings may be added or deleted.

Group names may be added or deleted, provided they do not cause REDEFINES or OCCURS boundaries to be crossed.

REDEFINES fields or groups may be added or deleted.

As a general rule, PICTURE cannot be changed, and elementary fields cannot be added or deleted. There are a few cases where a change can be made, very, VERY, very carefully, when the length of a field will not change. For example:

03 A PIC 999. can be changed to 03 A PIC XXX.

but the data can no longer be used as a number, nor can leading zeros be suppressed. Going the other way, from PIC XXX to PIC 999 may cause very strange numbers to appear.

03 A PIC 99 USAGE IS INTEGER. can change to 03 A PIC 999 USAGE IS INTEGER.
but not to 03 A PIC 9999 USAGE IS INTEGER.

because the first two are 2 bytes in length and the last is 4 bytes in length.

Fields can be combined if the total length is the same:

```
03 CITY PIC X(10) .  
03 STATE PIC X(2) .
```

can both be replaced with

```
03 CITY-STATE PIC X(12) .
```

There is one change which is always allowed, and that is to replace any field or combination of adjacent fields with FILLER of the same length.

A VALID IF statement can also be added, deleted, or modified. It should be noted that VALID IF applies only when storing or modifying data with Datatrieve: it does not check data which is already in the domain. Thus it is entirely possible to store data in a field, then add a VALID IF clause which makes that data invalid: no more data of that kind may be added, but the existing data will be unchanged. This is similar to the condition stated above where PIC XXX could be changed to PIC 999: if the data in the field happens to all be numeric digits, the data will be correct, but if there is anything else (including leading blanks), Datatrieve will assume they are supposed to be numeric digits, and this will result in some strange numbers being printed out. Datatrieve will not check the existing data, but will not allow you to modify or store a new field with non-numeric data.

If all of this scares you, it should be noted that the worst that can happen is that if you do change the length of the record definition, when you READY the domain you will get an error message telling you that the record lengths don't match, or you will read the data and get strange results. As long as you extract a copy of your definition before you start, and ready the domain for read only the first time after you change the definition and look at the data, it is very unlikely that you can damage your data, and the worst that can happen is you will have to make additional corrections to your record definition, or go back to the old one.

When changes are required which will not fit the above rules, such as adding an additional elementary field, then a new file will be needed to hold the new length record. One way to transfer information is shown under "Creating New Domains from Old" in the Datatrieve manual. This basically depends on the FOR statement, which reads the old domain one record at a time. Suppose I defined an address file like this:

```
DEFINE RECORD OLD-ADDR-REC
01 OLD-ADDR-REC.
    03 NAME PIC X(20).
    03 STREET PIC X(20).
    03 CITY PIC X(10).
    03 STATE PIC X(2).
```

;

and I have stored some data,

NAME	STREET	CITY	STATE
B. Z. Lederman	2572 E. 22nd St.	Brooklyn	NY
Hugo Z. Hackinbush	11 Julius Ct.	Hialeah	FL

and then I realize I forgot the Zip code. I can define a new record and corresponding domain:

```
DEFINE RECORD NEW-ADDR-REC
01 NEW-ADDR-REC.
    03 NAME PIC X(20).
    03 STREET PIC X(20).
    03 CITY PIC X(12).
    03 STATE PIC X(2).
    03 ZIP PIC 99999.
```

;

with an additional field and an increased field size for CITY, and move the data.

```
READY OLD-ADDR
READY NEW-ADDR WRITE
FOR OLD-ADDR STORE NEW-ADDR USING NEW-ADDR-REC=OLD-ADDR-REC
```

and the data will now be

NAME	STREET	CITY	STATE	ZIP
B. Z. Lederman	2572 E. 22nd St.	Brooklyn	NY	00000
Hugo Z. Hackinbush	11 Julius Ct.	Hialeah	FL	00000

Note that ZIP was filled with default characters, which in the case of numeric fields is zero, and that the city field has been moved. When one says USING new = old, Datatrieve will match up fields with the same name in moving data: any new fields get zeros or blanks. If the new fields are at the end of the record, it would be faster to move the data outside of Datatrieve using one of the RMS utilities (CNV or IFL on the 11, CONVERT on the VAX), or SORT, any of which can pad the new records and will transfer data between files faster than Datatrieve. If you are removing a field from the end of the definition, the same rule applies, as the utilities will truncate long records. If you are changing fields in the middle as was done here when the size of CITY was increased, then the SORT utility can be used, but if it is a one time only change, it will be easier (if slower) to use Datatrieve than to set up the sort commands. For example, suppose I decided I needed a second address line.

```
DEFINE RECORD NEW-ADDR-REC
01 NEW-ADDR-REC.
    03 NAME PIC X(20).
    03 STREET PIC X(20).
    03 SECOND-LINE PIC X(20).
    03 CITY PIC X(12).
    03 STATE PIC X(2).
    03 ZIP PIC 99999.
```

;

Using the same statement as before for conversion yields:

NAME	STREET	LINE	CITY	STATE	ZIP
B. Z. Lederman	2572 E. 22nd St.		Brooklyn	NY	00000
Hugo Z. Hackinbush	11 Julius Ct.		Hialeah	FL	00000

with the new line filled with blanks. One can do more than take the defaults, however.

```
DEFINE RECORD NEW-ADDR-REC
01 NEW-ADDR-REC.
    03 ENTRY PIC 99.
    03 NAME PIC X(20).
    03 STREET PIC X(20).
    03 CITY PIC X(12).
    03 STATE PIC X(2).
    03 ZIP PIC 99999.
    03 ENTRY-DATE USAGE IS DATE.
```

;

This time I want to add an entry number, and a date, so I have to give Datatrieve a few more commands.

```
DEFINE PROCEDURE CONVERT-ADDR
READY OLD-ADDR
READY NEW-ADDR WRITE
DECLARE COUNTER PIC 99.
COUNTER=0
FOR OLD-ADDR BEGIN
    COUNTER=COUNTER + 1
    STORE NEW-ADDR USING BEGIN
        ENTRY=COUNTER
        NAME=NAME
        STREET=STREET
        CITY=CITY
        STATE=STATE
        ENTRY-DATE="TODAY"
    END
END
RELEASE COUNTER
FINISH
END-PROCEDURE
```

Here I have a temporary variable which will automatically count up the number of entries and store it in the new domain: also, the ENTRY-DATE will automatically be time stamped (on the PDP-11, it will have only the date, not the time) as entered. Note there is still no ZIP = ... command: as I have no ZIP data in the old domain, I will let the new domain be filled with the default value of zero. I could also put in a prompt here and have someone enter the zip code during conversion, but on a long domain this would be very tedious, so it is better to convert automatically and modify the individual zip codes later. The new data is:

ENTRY	NAME	STREET	CITY	STATE	ZIP
01	B. Z. Lederman	2572 E. 22nd St.	Brooklyn	NY	00000
02	Hugo Z. Hackinbush	11 Julius Ct.	Hialeah	FL	00000

One can go on from here to make more elaborate changes if desired: the basic idea is that Datatrieve can be made to convert data from one domain to another if it should happen that a record definition needs to be changed, and this can be simplified though a careful choice of field names. As another example, suppose I had to change 5 digit zip codes to 9 digit.

```
DEFINE RECORD OLD-ADDR-REC
01 OLD-ADDR-REC.
    03 NAME PIC X(20).
    03 ZIP PIC 9(5).
;

DEFINE RECORD NEW-ADDR-REC
01 NEW-ADDR-REC.
    03 NAME PIC X(20).
    03 ZIP-CODE.
        06 ZIP PIC 9(5).
        06 PLUS4 PIC 9(4).
;
```

Because I have defined the field ZIP in both domains, the data can be directly transferred.

```
FOR OLD-ADDR STORE NEW-ADDR USING NEW-ADDR-REC=OLD-ADDR-REC
```

NAME	ZIP	PLUS4
B. Z. Lederman	11235	0000
Hugo Z. Hackinbush	33999	0000

It would probably be better if PLUS4 had an edit string to suppress zeros, and there was a way to print out the dash that joins the two parts of the ZIP code.

```
DEFINE RECORD NEW-ADDR-REC
01 NEW-ADDR-REC.
    03 NAME PIC X(20).
    03 ZIP PIC 9(5).
    03 DASH PIC XXX COMPUTED BY " - " QUERY-HEADER "".
    03 PLUS4 PIC 9(4) EDIT-STRING Z(4).
;
```

NAME	ZIP	PLUS4
B. Z. Lederman	11235	-
Hugo Z. Hackinbush	33999	-

The zip suffix doesn't print because of the zero suppression, but the data in the domain looks like this:

B. Z. Lederman	112350000
Hugo Z. Hackinbush	339990000

Just to give one last example, I will show how to get the data from FAMILIES to the two domains PARENT and OFFSPRING shown earlier.

```
DEFINE PROCEDURE CHANGE-FAMILY
DECLARE COUNTER PIC 999.
COUNTER=0
READY FAMILY
READY PARENT WRITE
READY OFFSPRING WRITE
FOR FAMILY BEGIN
    COUNTER = COUNTER + 1
    STORE PARENT USING BEGIN
        KEY = COUNTER
        FATHER = FATHER
        MOTHER = MOTHER
    END
    FOR KIDS STORE OFFSPRING USING BEGIN
        KEY = COUNTER
        KID = KID
        AGE = AGE
    END
END
FINISH
RELEASE COUNTER
END-PROCEDURE
```

As may be seen, there is a "loop within a loop". The FOR FAMILY moves through the domain and picks up each set of parents, and within this the FOR KIDS moves through the inner list to pick up each kid within a family. The use of the declared variable COUNTER insures that each offspring has the same key as the corresponding parent.

A final note: the length of each field is important when using the REDEFINES clause, when the data must be read by other programs, or when data is to be transferred between different systems (for example, from a VAX to a PDP-11 or PRO). There is a hidden "gotcha" that should be kept in mind:

```
DEFINE RECORD GOTCHA
01 GOTCHA.
    03 A PIC X.
    03 B PIC 99 USAGE IS INTEGER.
;
```

It might be thought that this record is 3 bytes long, and on a VAX it is, but on a PDP-11 it isn't: it is 4 bytes long, because of something called word alignment. INTEGER, REAL, DOUBLE and DATE must start on a word boundary, which is an even number of bytes: if they don't, Datatrieve inserts a hidden byte to align the data; in this case, between fields A and B. This byte takes up space in the record but can never be accessed, unless you change the definition. As a general principle, space should not be wasted, and if the application must be transported between a PDP-11 and a VAX the records must match, so in this instance, it would be better to reverse the order of the fields, so the INTEGER would be on an even boundary, and the record would then be 3 bytes long; or put an extra one byte field between A and B and use the space to store some other data field; or use the ALIGNMENT clause, which will make the VAX force word alignments in the same manner as the PDP-11, which will insure compatibility.



THE SAR DATA CATALOG SYSTEM:
AN INTERFACE BETWEEN THE SCIENTIST
AND THE DATA SYSTEM

A.A. Pang, A.L. Holmes, J.C. Curlander
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91109

ABSTRACT

The Shuttle Imaging Radar-B (SIR-B) was the second NASA/JPL synthetic aperture radar (SAR) experiment to fly on the Space Shuttle. To date, close to three thousand images have been correlated from the SAR raw signal data collected during the eight-day mission. This paper presents the SAR Data Catalog System (SDCS) developed as a low-cost, electronic information service to support users with timely access to SIR-B mission information, data products and image parameters. The catalog system, developed on a VAX 11/780 minicomputer, utilizes mission software, VAX Datatrieve and VAX/VMS utilities to implement its functions. The SDCS is both menu-driven and user-interactive. The design of the modules comprising the catalog system are discussed, as well as, its potential applications for archiving data from future SAR missions.

INTRODUCTION

Significant advances in remote sensing have been achieved since the first aerial photographs were taken in the mid-19th century [1]. With the advancement of imaging radar systems and digital computers, data returned from airborne and spaceborne sensors have proven to be of great scientific benefit. As an imaging device, the synthetic aperture radar has advantages over both real aperture and optical wavelength instruments in that it operates independent of cloud cover or daylight and its resolution is not a function of the sensor altitude [2]. The SIR-B instrument, part of the payload in the October 1984 mission of the Space Shuttle Challenger, collected a large volume of data over the 8-day mission covering parts of 6 continents and their surrounding oceans. An international team of scientists participated in the SIR-B mission, taking advantage of the unique characteristics provided by the SIR-B radar to study and observe the Earth's surface. Areas of study included geology, hydrology, vegetation, oceanography, cartography, as well as, the radar system itself. The results of these studies will pave the way for the development of innovative utilizations of future radar sensors [3].

*This paper presents the results of one phase of research carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

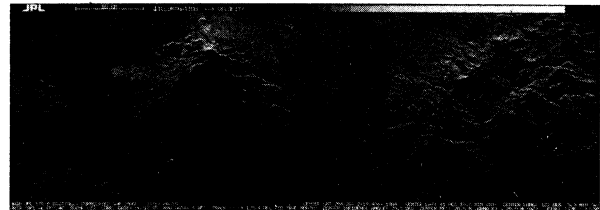


Figure 1. An example of a SIR-B SAR image produced by SDPS (Mt. Shasta, CA)

For the raw digitized SAR data to be utilized by the scientists, it must be processed into image products. The SIR-B Digital Processing System (SDPS) developed on a SEL 32/77 minicomputer is a software-based signal processor originally designed by C. Wu [4]. This system is the primary image production facility in support of the SIR-B science investigators. The processed images are stored on computer compatible tapes (CCT's) and an Optronics imaging device is used to produce the hardcopy photo-products from the image CCT's (Figure 1). These are the standard output of the SDPS archived by the JPL SIR Data Center (JSDC). In addition, the processing parameters associated with each image are saved on tape and transferred to a database on a VAX minicomputer maintained by the JSDC.

The addition of the SIR-B image data has resulted in a significant expansion of the SAR data archived at JPL. Concurrent with this growth is

the increasing problem encountered by the JSDC in managing this large body of data. The primary issue is data accessibility. This includes not only an investigator's ability to obtain data, but also information about the data itself. Therefore, to ensure maximum utilization of this data and its timely dissemination to the user community, it became essential to develop a data management system that has both the sophistication to handle many types of data and the flexibility to allow incorporation of additional information and the future projects into the system. The SAR Data Catalog System, which currently supports the SIR-B mission data set, is such a system. The objectives of this paper are to present some of the functional capabilities of the SDCS and highlight its features.

THE SDCS

This section describes the functions of the SDCS and the hardware and software modules which implement them.

Requirements and Functions

General requirements for the SDCS hardware and software are flexibility and expandability to include future users and data sets [5]. In addition, the catalog software is designed for users with no a priori knowledge of a query language or the existence of a specific data set. The system prompts the user as to a specific interest and provides a uniformly detailed response for each data set. In addition to supplying information, the SDCS accepts information from users for SAR data-product requests. Other features of the SDCS are on-line Help and the capability to contact other users on the system for real-time troubleshooting. The SDCS is, therefore, not only a catalog for SAR data, but also an interface between its users and a diverse set of services.

System Architecture

A VAX 11/780 minicomputer supported the development of the SDCS. A simplified system block diagram is shown in Figure 2. The SIR-B correlated image parameters are transferred from the image processing facility to disk storage on the VAX. Data acquisition from databases are done by VAX Datatrieve through the use of the Datatrieve Call Interface.

Software Design

An effective and low-cost method was derived to implement the SDCS. The approach is to utilize software developed for SIR-B mission operation. The only new programs required for the catalog system are the driver routines, which display the menus of options available and initiate the execution of the selected option. The existing application software used by the SDCS fall into three categories: (1) Mission software; (2) Datatrieve; and (3) VAX/VMS utility software.

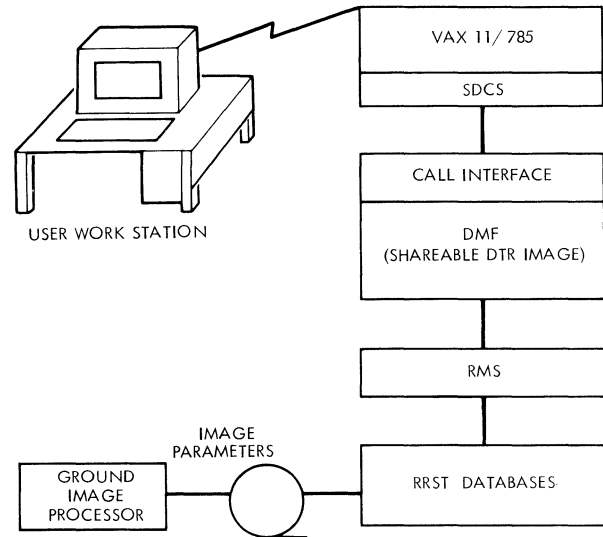


Figure 2. SDCS system block diagram

For some system users, these are not typically accessible and utilizing them might require special knowledge. The SDCS, however, brings these modules together under a central application and attempts to simplify the user interactions.

Figure 3 shows the hierarchical structure of the SDCS design. The top level displays the sensor(s) whose data sets are in the SDCS. Successive levels provide the information pertaining to each sensor. The SDCS is predominately menu-driven. The rationale for this implementation is for simplicity. Users select services or data sets by entering the single character identifier associated with the options presented on a menu. Within the options, more user-interactions are sometimes required. In these instances, the users are prompted for short answers with default values presented along with the prompts whenever applicable. Messages are displayed when user inputs do not conform to those expected by the SDCS. All of these features are designed to simplify the use of the catalog system.

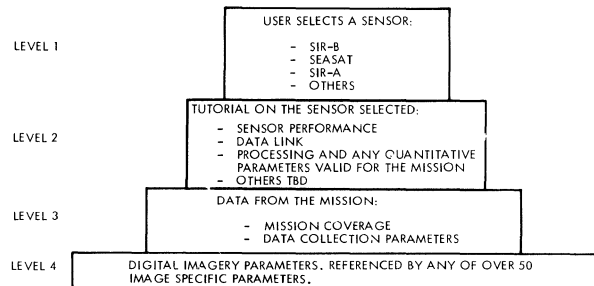


Figure 3. SDCS hierarchy

Descriptions of the remaining modules of the SDCS are given below.

Mission Software

A set of application programs were developed to support the SIR-B mission. These programs

provide insightful information pertaining to actual events which occurred during the mission. For example, planning programs developed to propagate the Shuttle's imaging paths and the graphics software developed to plot out the Shuttle's track are good sources of information for locating the ground areas covered. A collection of these programs have been brought together and modified into subroutines for the SDCS. The information they provide are offered as options under the SIR-B Mission Menu.

Software to Interface with Databases

Datatrieve serves as an efficient relational data management system for relatively small databases of a few thousand records. The JPL SIR Data Center has used Datatrieve as the exclusive data management language. Throughout many missions, the JSDC has taken advantage of Datatrieve's report writing capabilities to produce reports of the existing SAR imagery in its archive.

The Datatrieve Call Interface makes all of Datatrieve's information management capabilities available to application programs [6]. By using the Call Interface, the SDCS can access the databases already set-up by the JSDC for monitoring its SAR imagery collection. At the same time, all of the data query language peculiarities can be handled by the catalog system's driver routines. Thus, users do not have to learn the Datatrieve query language in order to use the SDCS. For options in the SIR-B Image Parameter Menu, the SDCS prompts the user for specific inputs regarding SIR-B imagery, reformats the user's reply into a proper Datatrieve command or data format and issues the command on behalf of the user through the Call Interface. The SDCS also handles the status information returned from Datatrieve and displays the results from data queries according to the user's specifications.

Aside from the SIR-B SAR image parameter data file, two additional databases have been created to support the SDCS. A user request capability for data-products and a user directory are defined and maintained by Datatrieve. Under the Processing Queue Menu of the SDCS, users can make requests and determine the status of their requests. The user directory enables both the JSDC and authorized users in the science community to exchange information and data. The User Directory Menu in the SDCS permits users to enter their address and phone number into a directory database and to access this database.

The Datatrieve Report Writer is an easy-to-use tool for uniformly organizing output data. After data is fetched by Datatrieve, the SDCS uses the Report Writer's data formatting capabilities to present the output to the user. Besides displaying the data on the terminal screen, the Report Writer can also output the data to a permanent disk file, allowing the SDCS to offer users the option of hardcopy reports.

VAX/VMS Utility Software

The VAX/VMS Mail and Phone utilities allow users to send messages and talk with other users on the system, respectively [7]. These functions are useful for real-time or near real-time troubleshooting. Also, since the image processing facility and science contacts are resident at JPL, remote users can conveniently communicate with the processing engineers for information pertaining to SAR data by contacting the appropriate individuals via the system.

The SDCS also has custom Help information designed to assist users while on the system. The VAX/VMS Help structure is followed in formatting this information.

Each SDCS menu offers these utilities as options. When one of these options is selected, the SDCS spawns to a sub-process and executes the utility. Upon exiting the sub-process, the SDCS resumes activity at the point of interruption. Data retrieval functions, performed prior to the execution of the VAX/VMS utility, are always preserved.

DISCUSSION

The development of the SAR Data Catalog System has demonstrated the feasibility of creating a low-cost, sophisticated interface between data and its users. This result is important for future SAR missions which plan to collect and correlate large volumes of data several orders of magnitude larger than SIR-B. The implementation described in this paper greatly simplifies user interactions with the computer while allowing maximum accessibility to archived mission data. The concept and design of the SDCS allows for other data sets to be incorporated with minimum effort. As more data is added, the SDCS could also be enhanced to include features such as a cross-reference capability between the data from different sensors.

FUTURE APPLICATIONS

Currently there are four sources of SAR data at JPL: SEASAT, SIR-A, CV-990 and SIR-B. Each of these data sets consists of raw data and image data in both optical and digital format. Within the next decade there will be no less than five new sources of SAR data, each with its own set of ancillary data and performance characteristics. The goal of the SDCS is to be able to support all of these. Since the volume of data collected during any mission is expected to be quite large, it would not be feasible to have entire data sets on-line. However, enough of the data can be cataloged such that particular attributes between data sets can be characterized.

Upgrades of the SDCS hardware configuration might include several large capacity storage devices as

an alternative to archiving imagery on magnetic tapes. The new optical media, along with a high data-rate satellite or terrestrial link, would give the catalog system the capacity for transferring image data to users in real-time, opening the doors for even greater accessibility and utility of SAR data.

ACKNOWLEDGEMENT

The authors would like to acknowledge D. Casey, B. Jai and M. Kobrick for their software contributions in the SDCS, and E. Chu for his system management support.

REFERENCES

[1] S.A. Hovanessian, Introduction to Synthetic Array and Imaging Radars, Artech House, Inc., 1980.

- [2] J.C. Curlander, "Performance of the SIR-B Digital Image Processing Subsystem," Science, June 1, 1985.
- [3] "The SIR-B Science Investigations Plan," JPL Publication 84-3, July 1, 1984.
- [4] Chialin Wu, Budak Barkan, Walter J. Karplus, and Dennis Caswell, "SEASAT Synthetic Aperture Radar Reduction Using Parallel Programmable Array Processors," IEEE Transactions on Geoscience Remote Sensing, Vol. GE-20, No. 3, July 1982.
- [5] J. Curlander, A. Pang, "Conceptual Design of a SIR-B/SAR Data Catalog and Archival Network," JPL IOM 3348-84-073, August 21, 1984.
- [6] "VAX Datatrieve Guide to Programming and Customizing," Digital Equipment Corporation, September 1984.
- [7] "VAX-11 Utilities Reference Manual," Digital Equipment Corporation, May, 1982.

USING DATATRIEVE AS A COBOL CODE GENERATOR

Lynn D. Duncan
Oak Ridge National Laboratory
operated by
Martin Marietta Energy Systems
Oak Ridge, Tennessee

ABSTRACT

This paper describes the use of DATATRIEVE to automatically generate large blocks of COBOL code for use in a computer aided instruction application. Pro/DATATRIEVE was used to store information describing the attributes of screens that make up the instruction and tests, and to maintain data on expected student responses for each screen. DATATRIEVE procedures were used to create standard, error-free COBOL code that controls the presentation and order of flow of the screens within the lessons and tests. The code generated by DATATRIEVE was incorporated into skeleton COBOL programs through use of the COPY statement.

This project involves the use of DATATRIEVE on a DEC Professional-380. The generated code segments were electronically transmitted to a Honeywell DPS-6 minicomputer where the application was compiled and executed. However, the procedures used for the code generation could be applied to any DATATRIEVE implementation and any application system that requires standardized procedures for processing a variety of functions (i.e., transactions).

BACKGROUND

The Oak Ridge National Laboratory (ORNL) has developed a Computer Aided Instruction (CAI) prototype which is written entirely in COBOL rather than in one of the many course authoring languages commonly used for CAI development. This project was initiated at the request of our sponsors, the Navy Management Systems Support Office (NAVMASSO), the Naval Supply Systems Command (NAVSUP) and the U.S. Department of Energy (DOE), to improve the quality and availability of training for shipboard computer applications.

A major requirement of the project was that the CAI run in the existing shipboard hardware and software environment, and be compatible with NAVMASSO's software development environment. In other words no new software or hardware procurement was possible. This environment included Honeywell DPS-6 computers running the GCOS operating system, the COBOL programming language, a forms creation and display utility (VFORMS/VDAM), and little else. The GCOS operating system was difficult to use, and no programmer productivity tools were available.

The ORNL team completed a detailed structured analysis and structured design as a first step in the project. This resulted in the identification of a modular set of functions that would have to be performed, and allowed for the standardization of those functions. Since development tools were not available on the Honeywell, the development effort was redirected to existing Pro-380 workstations.

APPLICATION SYSTEM

The CAI system that was developed consists of a series of preformatted screens. However, not every student views every screen, and the screens may be presented in a different order for each student. The primary function of this system is to display a screen, accept a student's input, and then select the next screen to display. The next screen is always dependent on the student's input.

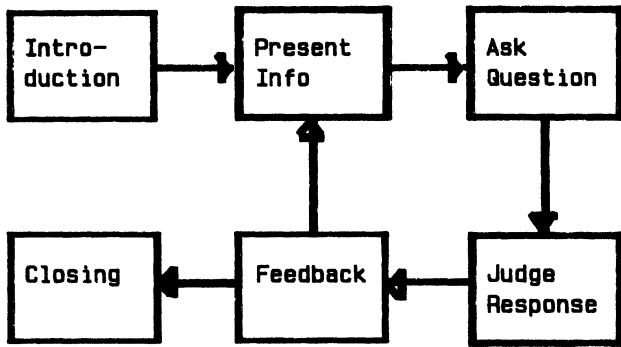
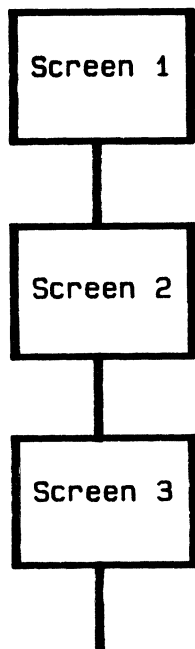


Figure 1 The flow of a tutorial.

This is typical of any general tutorial session (Figure 1). The CAI system presents information, asks a question of the student, and gets some response. Depending on the response, the student will either go on to the next topic or receive some remediation on the current topic. The introductory sections are very linear (Figure 2). One screen leads directly to the next screen, which leads directly to the next screen after that, and so on. The student is only required to press return to continue. Those segments are simple to program, but most of the lessons are a lot more complex. The lessons contain screens that present information and numerous screens that require some sort of response to gauge the student's level of



and so on ...

Figure 2 Introduction.

understanding. Depending on the type of question asked, there may be two possible responses or several dozen (Figure 3). Based on the student's response, one of the potential branches is selected. The COBOL program must dynamically determine what

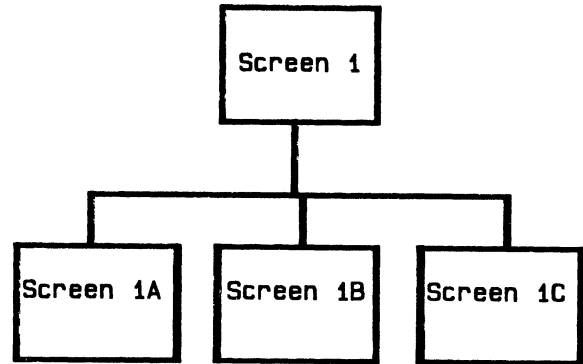


Figure 3 Lesson flow.

information (i.e., which screen) to present next. To accomplish this task, the COBOL programs reference two large tables. The first is called the FORM TABLE and the second, the DECISION TABLE. These two tables control display of the screens and the choice of next screen. The FORM TABLE contains information including

1. the name of the display subroutine which controls how this particular form is displayed;
2. the name of the answer subroutine which controls how to accept and process the information that the student types in;
3. special field locations for certain fields that require special processing; and
4. pointers into the DECISION TABLE for the entries associated with this screen.

The DECISION TABLE contains all of the anticipated correct answers, and all of the anticipated incorrect answers, plus a catch-all answer for each screen. Since the designers cannot anticipate everything that a student might type in, there is always a catch-all answer for the extraneous information. For each one of the potential answers in the DECISION TABLE there is a pointer back to the FORM TABLE, pointing to the specific form that should be displayed next.

These two tables comprise the major portion of the CAI programs and were automatically generated in DATATRIEVE.

DATATRIEVE ENVIRONMENT

This effort was performed using a PRO-380 running P/OS Version 2.0A with a 33 megabyte hard disk and Pro/DATATRIEVE Version 2. The Tool Kit is also installed and most of the work is initiated from the tool kit instead of from the Pro menus. Two data files were created with domains defined to DATATRIEVE (SCREENS and RESPONSES), and a third view domain was defined (FLOW), which includes all the information from SCREENS and RESPONSES joined together. The DATATRIEVE domain definitions loosely parallel the two tables in the COBOL program.

There are DATATRIEVE procedures for data input, for data update, for data display, for code generation, and for documentation reports. The input, update and display procedures were written to maintain consistency between the two datafiles and to make the repetitive functions faster and easier. Once the data is input, it is simply a matter of selecting a procedure to either create the COBOL code or produce reports (Figure 4).

(The procedures are not discussed specifically in this paper, but samples are available in the DTR/4GL Session Notes.)

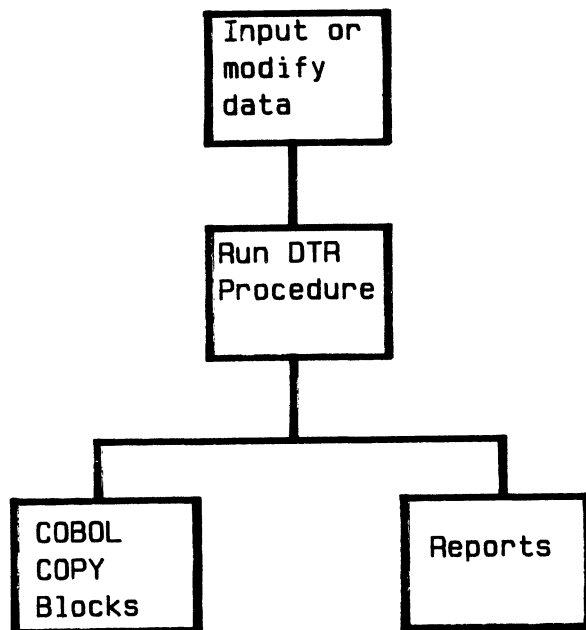


Figure 4 Code Generation Procedure.

PROBLEMS AND SOLUTIONS

Rather than discuss any of those procedures in detail, I will discuss some general problems that I encountered. My personal experience with DATATRIEVE began on a PDP-11, then I migrated to the VAX and most of my experience has been on the VAX.

I still have difficulty trying to do things that work on the VAX but do not work on the Pro. Sometimes I get frustrated, but I generally find I just have to do something differently to achieve the same results.

First of all, Pro/DTR has no ON statement. I used a series of PRINT statements in the procedures and I would have liked to use:

```
ON file-name  
BEGIN  
...  
END.
```

Instead of the ON statement I used an OPEN statement and multiple levels of procedures so that all of the output from the procedure was written to a file. The biggest drawback of this approach was that I was not able to prompt for the file name.

Pro/DTR has no CROSS clause. I would have liked to use CROSS to join domains. Instead, I used nested FORs which worked just as well in most instances. I also used a view to join two domains, which was not as easy to work with. The hierarchical structure of the view adds an extra degree of difficulty. I was forced to learn how to create inner print lists which are not a lot of fun, but, if you get the context right, they work.

There is no REDUCE statement or REDUCE TO which would have been very useful. Instead, I used a more procedural programming approach. I defined variables to use as counters and processed a series of records sequentially comparing each record to the last record in order to do things that I would have done with a REDUCE statement.

Pro/DTR offers limited concatenation, only the single bar and double bar, not the triple bar. Quite often, I needed the functionality of the triple bar concatenation, where all trailing blanks but one are removed. To get the desired output, I used a combination of the available concatenation functions (item1||" "|item2).

APPLICATION SOFTWARE CONFIGURATION

Once the COBOL COPY blocks were created by DATATRIEVE, they still had to be integrated with the rest of the system. A set of standardized skeleton programs and a set of COPY blocks in the library were used to build the system. The compilation process took the skeleton program, and the COPY blocks out of the library to create the object code (Figure 5). There are four standard skeleton programs, one for each program type. There is one to actually present the lesson, one to present summary information, one to present reviews, and one to administer the test. Each one of the skeletons follows a standard format. The skeleton lesson program is the most complex and consists of a total of 13 lines. That includes seven COPY

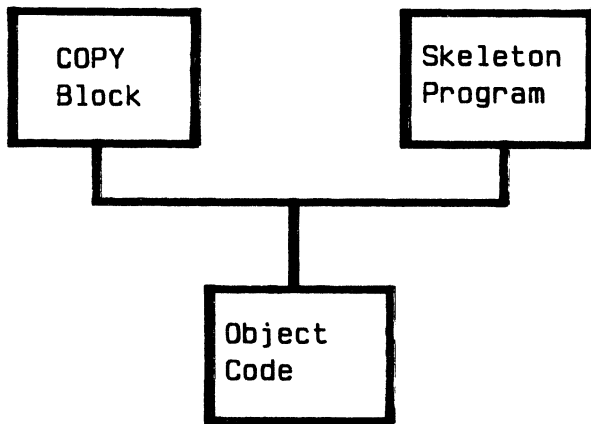


Figure 5 Compilation Process.

statements. The compiled programs that were created from that skeleton are up to 1700 lines. There are 15 lessons and the only variations in the skeleton lesson programs are two items containing a unique Unit/Lesson number. Everything else is static.

There are seven COPY blocks referenced in the skeleton lesson program. Six of those are used without change in every lesson. One of those copy blocks is unique to each lesson. That is the one that is built in DATATRIEVE. That is the one that has all the screen control and answer analysis information. The COPY blocks are all stored centrally and referenced at compile time.

Because of incompatibility of communications protocols between the Pro and the Honeywell, a Zenith PC was used to electronically transfer the generated COPY blocks to the Honeywell DPS-6. The Honeywell is the target machine where the COBOL application actually runs. Once the COPY blocks were transferred to the Honeywell, then it was a simple process to compile the skeleton programs, and produce the object code and eventually the entire computer aided instruction system (Figure 6).

BENEFITS REALIZED

Numerous benefits were realized by using this procedure. First and foremost is the productivity enhancement. The project team spent longer than "average" in the early stages of analysis and design but by so doing, the development and testing time were shortened significantly. By using DTR to generate the repetitive portions of the lesson, summary, review and test programs, the programmer/analysts could concentrate on the more complex aspects of the project.

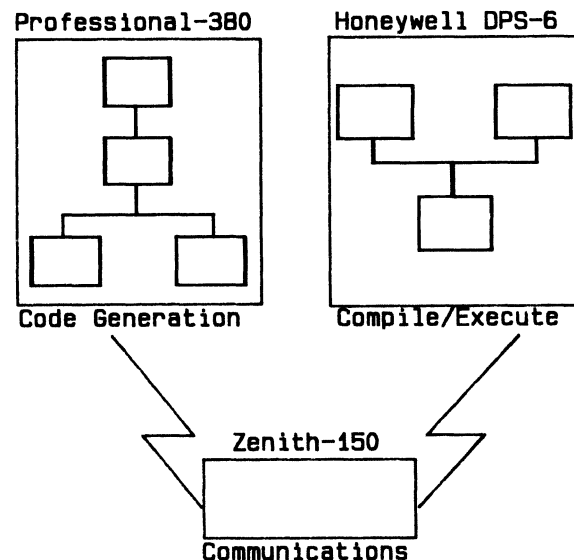


Figure 6 Application configuration.

The programmer/analysts also had a lot of flexibility. It was very easy to change or modify the information about the screens and the order of flow. They could add new screens into the series very easily. They could change the order of the screens, they could delete screens, and it was a simple matter just to go into DATATRIEVE and delete a record and rerun a procedure and send a new COPY block to the Honeywell. The timeliness was increased because it was such a simple matter to make the changes. Modification required as a result of testing or at the request of sponsors was completed very quickly. The modular structure and automated code generation guaranteed that each newly created COPY block would be error free each time we ran the procedure.

There are also quality enhancements. Standardization is automatically built in. Since all of the COPY blocks are created by the same procedure, each one follows the exact same format. Maintenance is also automated, therefore it is easy, timely and very flexible. Again, the changes go into DATATRIEVE, a DATATRIEVE procedure creates a new COPY block, and then the programmer simply transfers the new file and re-compiles on the Honeywell and it produces error free code.

Up-to-date documentation is another major benefit. All the data is stored in one set of data files. That same set of data files can be used to produce the code or produce the report. There is a central point of control so that the documentation and the actual code always stay in sync.

What has been described of the CAI is just the student subsystem. This subsystem consists of the programs that interact with the student. For the student subsystem, the automatically generated code was 74.1 percent of the total, about 20,000 lines. In addition to the student subsystem there is an instructor subsystem. This subsystem can be used by an instructor to track the students' progress and see how they are doing. The instructor system did not use the automated DATATRIEVE code generation. When the instructor system is included in the complete system the automated code accounts for 57 percent of the total system (Figure 7).

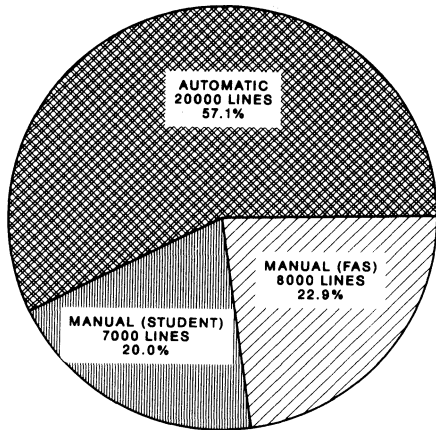


Figure 7 Lines of code.

Anytime you can generate error-free COBOL code, you will realize a savings in development, testing, and debugging time. When that code amounts to over half of the total application, the savings is a very significant achievement.

APPLICABILITY AND EXTENSIBILITY

There are a few key elements in this development strategy that need to be followed to reap similar benefits. First, we followed a strict structured analysis, structured design methodology. We planned what we were going to do. We didn't just sit down and start coding. We did a lot of standardization and modularization. We knew ahead of time what functions were going to be used in every program so it was easy to select the portions that could be generated in DATATRIEVE. The automated code generation using DATATRIEVE really gave us the big productivity increase.

As far as the applicability to other systems, a similar approach could be used in most any application. A similar approach could be used to generate entire programs. In fact we are modifying our system now and the next set of programs will be entirely generated from DATATRIEVE. The approach is particularly applicable to those applications where there is extensive user interaction, where you have standardized functions, or you have some kind of transaction processing.

The same kind of approach is not only possible in DATATRIEVE, but could probably be done with most other data management systems. You have to plan ahead and invest time and effort in the analysis and design; and then you can use the data management software to perform the standardized, repetitive portions of the application development.

SUMMARY

There is general agreement on the benefits of automated code generators but there are certain drawbacks, too. With the approach described in this paper, you have all the benefits mentioned above plus one more. You don't have to select, justify, fund and acquire a new product, and then learn how to use it. You already have DATATRIEVE running on your system and you know how to use it. You simply have to write a new application using existing tools.



Commonly Asked DATATRIEVE Questions & Answers

Larry Jasmann (Chair)

U.S. Coast Guard
Burke VA

Joe H. Gallagher

4GL Solutions
Kansas City, MO

Andy Schneider

Developer, VAX-DATATRIEVE

Dick Azzi

Motorola
Phoenix, AZ.

Chris Wool

E.I. DuPont
Wilmington, DE

B. Z. Lederman

Brooklyn, N.Y.

Transcribed by B. Z. Lederman

ABSTRACT

This is a transcription of a panel presentation which answers some of the most common questions asked about DATATRIEVE. Some of the material has been reordered when that would group logical subjects together. The transcription may paraphrase some questions or answers for clarity, and the transcriber apologizes in advance for any misspelled names. This paper follows the usual convention of placing square brackets around interpretations or material supplied by the editor. Throughout this paper DTR is an abbreviation for DATATRIEVE.

Why is DATATRIEVE so slow?

(Larry:) DTR has a lot of power, and does a lot of things, but it also "sits on top of" one of three other products: RMS, DBMS, or Rdb. If you use DTR in such a way as to cause, for example, RMS to do a sequential search of a file containing 20,000 records, you should not be surprised if it takes a long time to respond with an answer. If you are a

programmer [in a traditional language] you probably wouldn't do such a silly thing when writing a program, but when you are using DTR interactively on a large file it's really easy to do this. Joe has some slides which show the difference between retrieving data with keys and without keys. [Figure 1] You can see that when you get beyond 1000 records, the amount of time to access a file sequentially skyrockets compared with keyed

Order of Keys versus Non-Keys

in CROSS statement

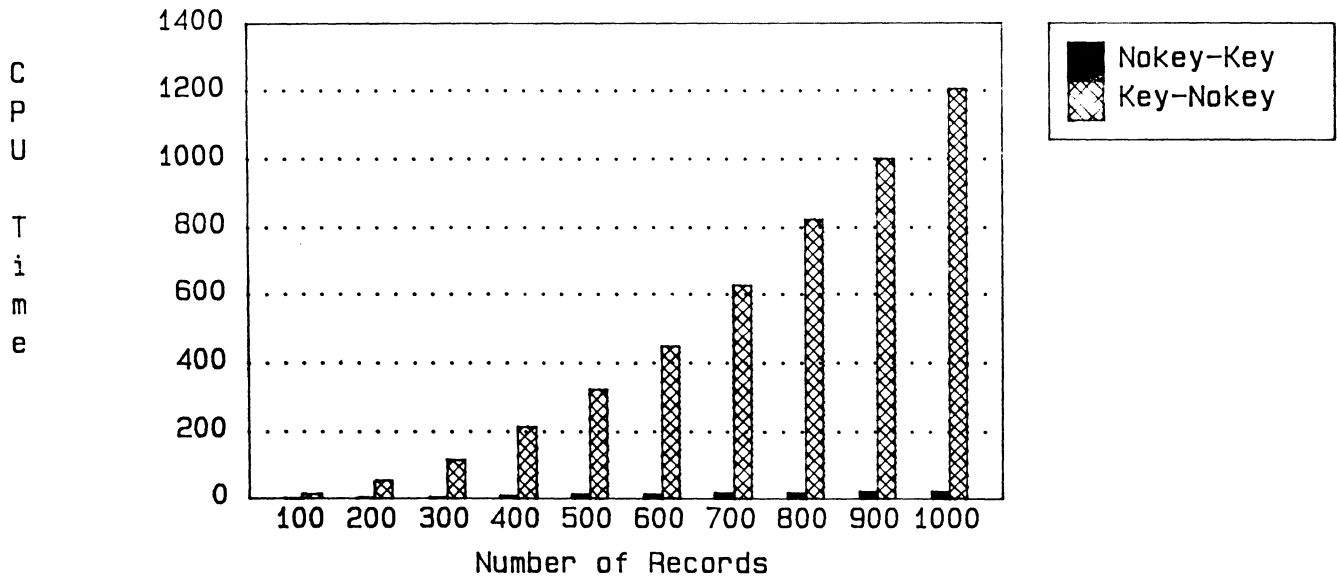


Figure 1

One Key versus Two Keys

Update Time

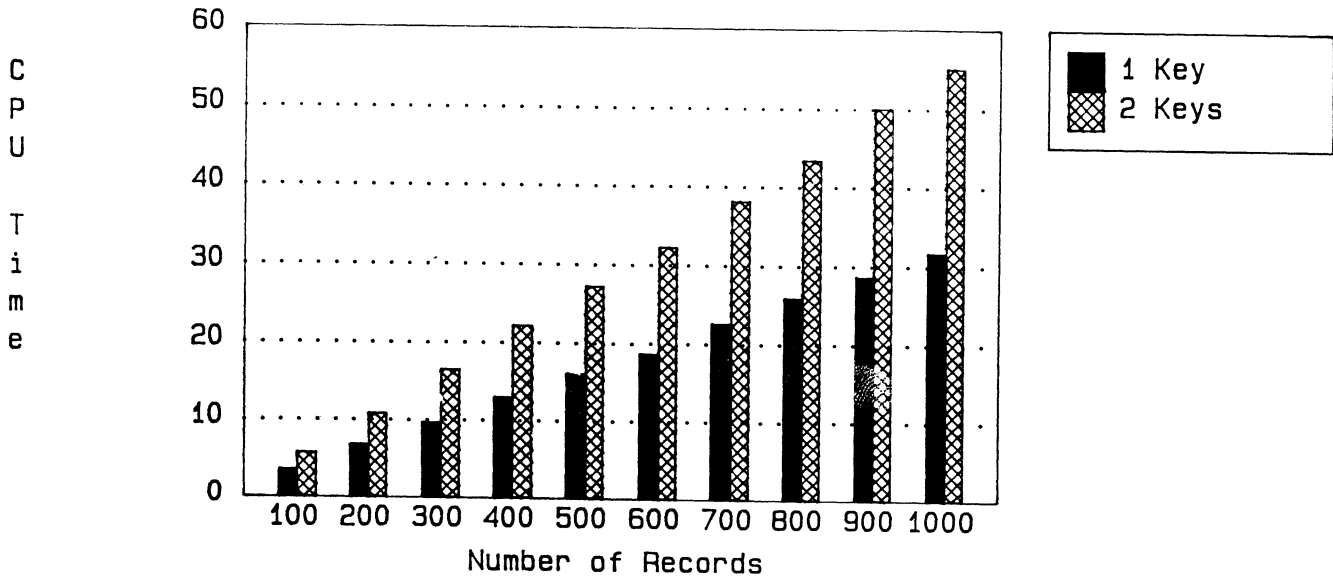


Figure 2

FOR versus FIND

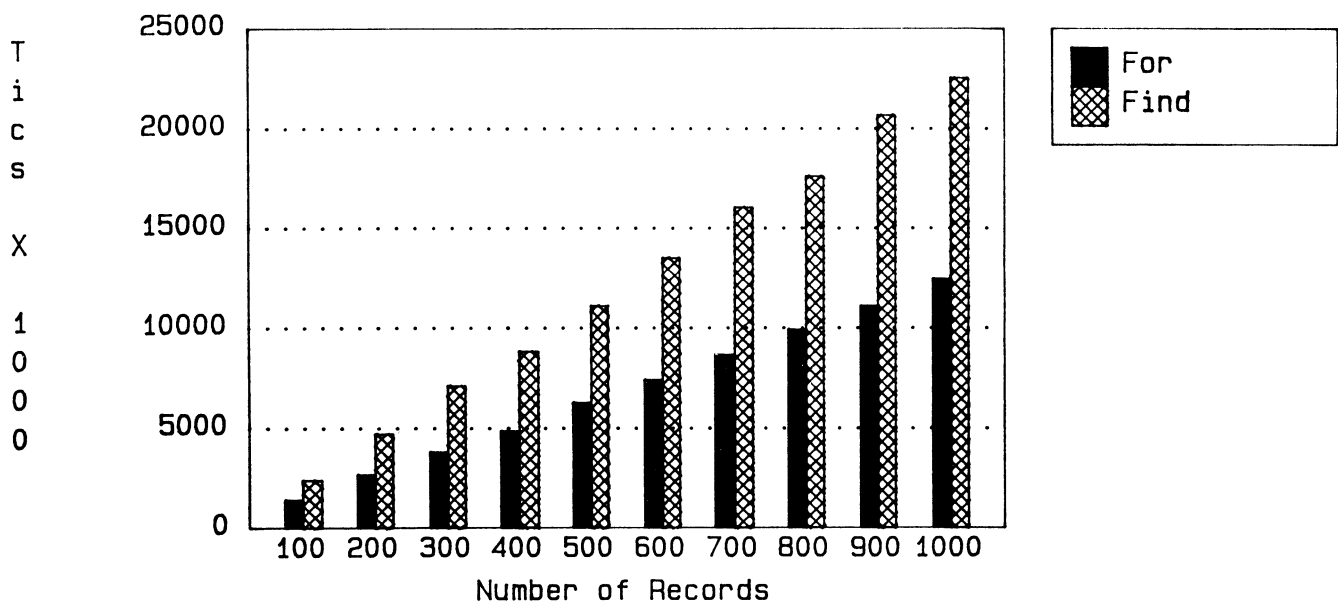


Figure 3

retrieval. *(Joe:)* This example is in fact a CROSS, that is you are doing a relational join between two domains: the second domain does not have a key in one case, and does in the other. The performance ratio is essentially the same as for a simple lookup doing a single keyed retrieval compared with a single sequential search. The point to be made is that DTR is only as fast as what it sits on: it's because DTR hides some of the details of what is going on below it that many times it's possible to do something that seems perfectly reasonable to you, but is very slow because the file design is not appropriate for that function, and performance suffers considerably. *(Larry:)* A corollary: it's not altogether clear unless you've studied it which constructs in DTR will cause sequential searches, and this is something you need to know well if you have a big file. *(Dick:)* My first answer to this normally is: if you are on a VT100, try hitting the NO SCROLL key again. That happens quite often: that will slow the system [your application]. Along with the sequential portion, the number of keys [in an indexed file] has a direct bearing on how fast the application will be: this doesn't matter much on a read, but on a write, the more keys there are the slower it will be. [Figure 2] *(Joe:)* if you are going to retrieve records on both keys, the time needed to store the extra key will be well worth it in the retrieval. However, if you are not going to retrieve records on that key, you are going to pay an overhead price. It's important to choose the keys carefully, to use only those which will be used for retrieval. These underlying factors in file design determine how fast the application will be in Datatrieve. *(Andy:)* One of the advantages to using keys is that RMS will do sorting for you. When you create a primary key and DTR says "give me the records" RMS gives them back in sorted order. If you have an application and you have a primary key, and you enter a command which sorts on that key, why would it take so long? DTR isn't super smart: if you explicitly order DTR to sort the data, it isn't able to tell that it's already sorted. So: don't go out and automatically sort everything, figure out what are primary keys, and don't sort fields that are already sorted. How do you know when your retrieval is using keys? One way is to do this: instead of just running the DTR image, run it with DEBUG.

\$ DEBUG SYS\$SYSTEM:DTR32

You get some VAX DEBUG headers and messages, and then the prompt: simply say "GO", and you will be in DTR. What you are doing is initializing the

DEBUGGER, and then you will be in DATATRIEVE. What happens is that when you perform an RSE, if a key is being used you will receive an informational message on your terminal for every key being used in your RSE or Boolean or whatever. If you were assuming that three keys are being used, this way DTR will tell you if those keys are actually being used or not. If it isn't using it, then perhaps there is a flaw in your design, and you can go back and work on it. This is a debugging technique to see if what you are doing is what you thought you were doing.

(Larry:) Another thing you need to know is FIND and FOR statements. If you do a FIND and create a current collection, subsequent operations on that collection are going to be done sequentially. [Figure 3] This is usually the minimum amount of time you will save with a FOR, but there are other savings that are also obtained. You should remember that any operation on the collection is not keyed, even if it looks as if it was. The only time a FIND is better than a FOR is if you have a very large domain, and you can do a FIND to collect a relatively small number of records, and are going to do several operations on that small collection. For example, if you have 10,000 records and want to work on a subset of 50 or 60 records, it makes sense to use a FIND, otherwise not. *(Dick:)* While we are talking about FINDs, it's important to remember that when you do a SORT, even to do a PRINT (for example, PRINT FIRST 5 --- SORTED BY field), that DTR is going to do the SORT first. If you can do a FIND and reduce the number of records you are going to be using, and then SORT that small number, you will save a lot of time because DTR will not have to sort the whole file.

(Andy:) another bottleneck is access to the dictionary (the CDD). It's crucial, especially at initial access time, that the dictionary not be "top heavy". A lot of people make the mistake of putting everything into CDD\$TOP, and then when you want to ready a domain the amount of time that it takes for CDD to access the pieces in the dictionary is extremely high. If you have a lot of stuff in CDD\$TOP and not much in subdirectories, create a good tree structure and move stuff down the tree.

(Joe:) There is one area I run into that most users don't run into. From a scientific and medical standpoint, we have some users who do calculations in DTR rather than some other language like FORTRAN), so in fact they are doing a lot of calculations in DTR. In many cases they created complex procedures where all of the temporary variables are declared something like PIC 999V999 (string variables). If for some reason you have to

do heavy calculations in DTR you gain a substantive return by converting those variables to COMP variables (REAL, INTEGER) because DTR does a fair amount of conversion, and these are CPU intensive activities where numbers in one format has to be converted to other formats with a lot of sanity checks. If you are going to do a lot of calculations (such as a data base of scientific data) you get a performance improvement by making the variables the appropriate data type.

(Joe:) If you think you are running slow, and you don't know if it's you or other programs [on the system] there is a pair of functions within VAX-DTR which will allow you to initialize a timer and then show the amount of elapsed time between the initialization and the show time. [For example, the following procedure was used when testing the CROSS statement with and without keys to obtain the data shown in Figure 1.]

```
FN$INIT_TIMER
FN$SHOW_TIMER
PRINT field1, field2 of
      domain1 CROSS domain2 OVER field1
FN$SHOW_TIMER
```

You can place them around various sections of code and find those sections that are actually running slow. It will give you information about elapsed clock time, CPU time, page faults, etc., and that's very helpful. If something is not running as fast as you think it ought to, you can go and look and decide if it's your process or someone else who is hogging the system [if CPU time is small but elapsed time is large]. (Bart:) on DTR-11 and PRO-DTR you don't have INIT_TIMER, but you can do remote DTR and the log file will have some information on times and what DTR is doing with the retrievals. This will also work for VAX-DTR, and is another way to find out what is going on inside DTR. You can always do a remote DTR to your own node.

Why can't I put a READY inside my BEGIN-END loop?

(Andy:) [A suggested source for information on this point is the VAX-DTR internals session], but basically this has to do with the difference between commands and statements. In a nutshell, the reason you can't put a command like READY within a BEGIN-END block is because commands go through one path [when being processed by DTR] and statements go through another. When you put a BEGIN-END around something, what you have done is created one big statement out of everything within the BEGIN-END block. If you stick a command in there and DTR runs across it, it's not down the right path [internally] to execute it. (Larry:) the main thing is to understand that there are such things as commands and statements and that one can't go in the other. With the way the language is constructed there is little need to do that: there are ways around it. (Joe:) A simpler explanation is that statements manipulate the data within an environment, and commands change that environment. By putting a command within a BEGIN-END loop you've changed the environment while trying to work in it.

Can I read DTR files from another language?

(Dick:) There is no such thing as DTR files. There are RMS files, Rdb files, DBMS, etc. DTR does not create files of it's own. You can read RMS from BASIC, COBOL, or any language. The converse of that is DTR can read files created by other languages: even the editor if you are careful. (Bart:) the problem with the editor is that you may not align everything properly. You may also run into the problem where you create a record definition 80 bytes long, and you go into the editor and type data 80 bytes long but the editor creates a variable length file. When you ready the domain DTR will give you a warning message that the file types don't match, but it will then go ahead and read it anyway. If DTR gets a record which is too short, it pads it out (and may give an error message): if it's too

long, DTR truncates the record (and in the past, especially on the PDP-11, tends to abort). If you are in doubt, put a FILLER field on the end of the record definition to make the record definition too long: you may get warning messages but DTR will go ahead and read the data. You can then write it to another file with fixed length records and DTR will be happy. (*Larry:*) a related question is, what if you have a nasty system manager who won't tell you what the file is like and you are trying to read it? The answer is, use the RMS utilities to find out how long the record is, then create a record definition of the same length with one big field with a PIC length the length of the record, EDIT_STRING T(80) [to make the data fit on the usual CRT screen), ready the domain, print some records out, and by looking at it you can usually figure out where the fields are, and revise your record definition. (*Bart:*) remember to ready the domain read only and shared, until you are certain you have the record definition correct. You don't want to modify anything until you know what it is.

Can you sort on a non-keyed field?

(*Dick:*) You can sort on any field you have in your record. (Not COMPUTED_BY fields on a PDP-11, but any real field.) On a VAX, it should be any field.

Why can't I prompt for a domain or a field?

(*Andy:*) DTR is really forgiving, but there are certain features intended to be used in some places and not others. Prompting, when you do a *.prompt, is for value expressions and value expressions only. A value expression is a value for a field, or a piece of text. They don't include things like key words or names of things, which is what a domain is. When you say READY *.--, what you are prompting for is a value expression, and DTR will say "oh no I won't!". Essentially, the contents of a quoted string is what it grabs, so when you prompt for anything it must be a value expression or piece. There are some workarounds, one being logical names: you prompt for a string, do an FN\$-- to create a logical name translation, ready the logical name and DTR will translate one level of logical names.

How many records can I have in my domain? How large a record can I have?

(*Bart:*) Basically, the number of records you can have in your domain is limited by how large your disk is (or your disk quota if applicable). As for the size of the record: on the PDP-11 if it's very large you will run out of pool space. On the VAX there may be a limit, but I don't know anyone who has hit it. (Comment from audience indicating it had been reached.) There is a system wide RMS limit on the maximum size for any record on a VAX, and I believe it's set around 32,000 bytes. As far as the number of records, it's limited by the amount of disk space, and I've done domains with over 130,000 records. (Comment from audience indicating a user with 6000+ byte records, stating that the application seemed a bit slow, but when the application was broken up into smaller pieces with relevant sections connected by crosses, it ran faster. Doesn't it make more sense to keep the record size smaller?) (*Bart:*) It's partially record size, and partially the number of fields. If you don't need all 6000 bytes at once, breaking it up into smaller pieces that most logically go together will save you overhead. The other possibility is to have more than one record definition for the same file and use FILLER to skip over the pieces that aren't needed at that time, and that also cuts down the number of fields that DTR has to know about. Either of those approaches would give an improvement. (User: if you use filler, it cuts down the number of fields, but then you have the same number of FILLER fields.) You use one FILLER field to skip over all of them. (*Andy:*) One important point we are looking at for 'way in the future is that access to the CDD is very inefficient for metadata. For every attribute you have for every field DTR has to make a call to the CDD. If you have 400 fields, and each field has a name, a query header, and edit string, a query name, missing value, default value, DTR makes one call for each [at least 2800 calls including the PIC clause]. If you can cut unnecessary attributes, or you have fields that aren't used often and you can skip over with FILLER, DTR jumps over FILLER and it's internal field tree is much, much simpler. Also, less memory is used, as it allocates a big block for each field, and this block is the same size for a field with no attributes or with many attributes. If you can eliminate fields, you save time and memory not allocating blocks. (*Larry:*) Besides, anyone with a record that has 6000 bytes in it needs to go back and re-evaluate how the data is being structured. (*User:*) the records were a complete record of our

field engineers, including their education, experience, etc. In essence, we had 10 major areas of interest, and instead of 1 record we really had 10. It worked a lot faster [after we changed to 10 records.] (*Bart:*) Not just speed but other considerations apply: if you give someone write access to that domain, they now have access to everything in there, and do you really want to give them everything at once? From the management standpoint you also want to separate the data.

Can I do menu-driven applications in DTR?

(*Larry:*) Yes, and I know of about 4 different methods. The first is the way NOT to do it, and that is to use DCL and have it call DTR every time you need to do something (having the menu in DCL). This will work, but it's inefficient: you go through all of the overhead of starting up DTR whenever you want to do something.

I like to use the call interface, and a little program that feeds procedures back to DTR. Essentially, DTR tells the program what it wants to do next, and the program tells DTR to run it [a procedure] next. That works very well. (*Dick Azzi:*) I like to "pre-compile" DTR. Andy mentioned that anything within a BEGIN-END block is treated as one statement, and DTR always has to parse the next statement it is going to work on. We take maybe 75 to 100 "programs", put them all within a large BEGIN-END block with a menu so DTR treats it all as one statement: it takes 15 to 20 minutes to parse that statement (we bring it up on Monday morning and leave it up until everyone goes home Friday night). Included on the menu are functions "sleep" and "pause", which bring up an FMS screen with a no-echo password so that the user can leave a terminal and get back in only if they enter the same password. This process works well in our application, where we treat DTR as the center of the universe: if we have to do something in DCL we will spawn out of DTR, work in DCL (things like word processing, PHONE, running another program), and then return to DTR where all of the pre-compiled statements are still active, all READYs are still there, etc. This gives a very quick turn-around on menu response.

(*Larry:*) Another method is with logical names [calling a procedure which has one fixed name: a logical name assignment is made from the menu to translate that logical name to the name of one of a set of "real" procedures]. There are probably other methods as well.

(*Mike Nickolas, Bank Ohio:*) Another method of doing menus in DTR is to have a simple procedure called DISPLAY_MENU which has a print statement to clear the screen, displays an abbreviated form of the procedure such as ":M1", and that procedure only does one function such as ADD [a record], which would require only a very short compilation time.

(*Chris:*) I'd like to make an exception about how not to do a menu [using DCL described above]. There are times when a DCL menu is appropriate. If you have several choices on the menu and only one is going to be DTR, the startup delay occurs only after the choice of that option is made. The main menu can come up very quickly in a DCL procedure, and if there is only one choice which goes to DTR, or there are many options only one or two of which go to DTR, the total delay is less than if you have to go in and out of DTR a lot.

Using logical names is very similar to the "pre-compile" method. The difference is that in your choice statement you use the FN\$CREATE_LOGICAL, and at the bottom of the choice statement you invoke the logical name and DTR will execute the procedure name used in the create logical function: you can even go back and invoke the procedure you are in now. This appears to be recursive use of DTR but in fact is not really working recursively.

Susan Krantz, NKF Engineering: another way I use DCL and DTR together is in DBMS applications where I have a COBOL program using FMS doing an update function and then have a menu in the beginning asking if I want to use the update program or do I want to go into DTR and do my reporting. That's a good combination because they are either changing the database or they are doing report, and once you are in the report module everything is pre-compiled.

(*Larry:*) as Chris said, if you are doing one-shots then the DCL menu is good, but on the other hand if you are going to switch back and forth between updating and reporting I'd use the call interface and integrate [DTR] right in [to the COBOL program].

Will the DTR Call Interface support new languages?

Ron Swift, Xerox: we use a FORTRAN interface for some of [those applications, such as were discussed for the previous question] which allows us to leave the domains open. It allows us to go through a menu, and appears to speed up tremendously what we're doing. My question is, will there ever be a "C" interface to DTR

(Andy:) Do you mean, will there ever be something which DTR ships as part of it's kit to allow you to automatically run with "C"? You can do it today, but you have to create your own DAB. The bottom line is: any language which conforms to the VAX calling standard can use callable DTR today, which means "C" can use it. We have chosen a subset to ship with our kit which means DABs, examples, and so forth. We have been asked for "C" in the past and that may come, although it doesn't stop you from using it today, it just means you have to do some legwork up front.

(Bart:) A little advertisement for the DECUS library: if the first person to do it would please submit it to the library then everyone else will get it.

How do I used nested FOR loops? (i.e., how do I optimize access to two domains?)

Bob Brown, INTEL: In regards to optimization, could someone explain to me how nested FOR loops work, where you put the keys (on the inside or outside); it's kind of difficult to understand from the manual.

(Bart:) [rather than transcribing the problem description, the example below shows the outline of a nested FOR statement being used to access two domains, with records in the second domain being selected according to the match of some field in the second domain equaling a field in the first domain.]

```
FOR domain_1 BEGIN
  FOR domain_2 WITH field_2 = field_1 BEGIN
    ---
    work done here on one or both domains
    ---
  END
END
```

The field that you are specifying in the second domain (the inner loop) [labeled field_2, belonging to domain_2 in the example above] is the one that should be keyed. This is exactly the same as the example shown earlier for the CROSS, where it was going nokey-key. If it's a VIEW, and you specify the first domain, and then the second domain occurs for a field equal to the first domain, it's the second domain where the field should be keyed. For all cases, for a record in the first domain, DTR has to find the matching record in the second domain. [The following illustrates the case of a VIEW. As in the first example, field_2 belonging to domain_2 is the one which normally should be keyed.]

```
DEFINE DOMAIN view_domain OF domain_1,
  domain_2 USING
  01 FIRST OCCURS FOR domain_1.
    10 field_1 FROM domain_1.
      --- other fields from domain_1.
    10 SECOND OCCURS FOR domain_2 with
      field_2 = field_1.
      --- other fields from domain_2.
;
```

(Larry:) please understand that the field doesn't HAVE to be keyed, but if you want to take advantage of the keys, it must be the second domain that is keyed. (Bart:) If you would like it to run in a reasonable amount of time, the second should be keyed.

Bob Brown: even if the outside loop has an RSE?

(Bart:) Yes, because the outside loop is going to go in the order you specify in the RSE: it may or may not be keyed, depending on how you do it. But if you want the matches on the inner loop to be fast, then the second domain has to be keyed so the retrieval can be on the key.

"Field --- is undefined or used out of context"?

(Joe:) [We now have] the infamous "undefined or used out of context" error message. This is probably the most frustrating and common occurrence for beginning users of DTR. There are some important obvious things, such as misspellings, that cause this problem. The real underlying cause is a cry for help from DTR because it does not understand what you have told it. There is something in the command that it does not understand, and when it doesn't understand it doesn't know where it doesn't understand (or maybe not know where it doesn't understand). (Larry:) here's the point: a lot of times when you get "undefined ..." the error is not on the quoted string in the error message, the error is going to be somewhere back upstream from that string. What I tell my users is to put your finger on that string, and where I'm pointing is where the error is [using your right hand, the error is somewhere left and up]. This can happen because DTR continues to parse for a while until finally things don't make sense, and then you get the error: the thing that caused it not to understand could be a comma or a space or word encountered previously. (Joe:) You look at the thing it's complaining about and back: sometimes what it's complaining about is a misspelling, but it's also possible that it's something back up the line. This happens because DTR is parsing and compiling, and it has a context within which it has to communicate with you, trying to understand what you're telling it: at this point it's saying "I don't understand anymore".

When do I use hierarchies? (OCCURS clauses in record definitions)

Ron Wilson, Wilson Concrete: I wonder if there are any rules when one should use "flat" records versus hierarchies?

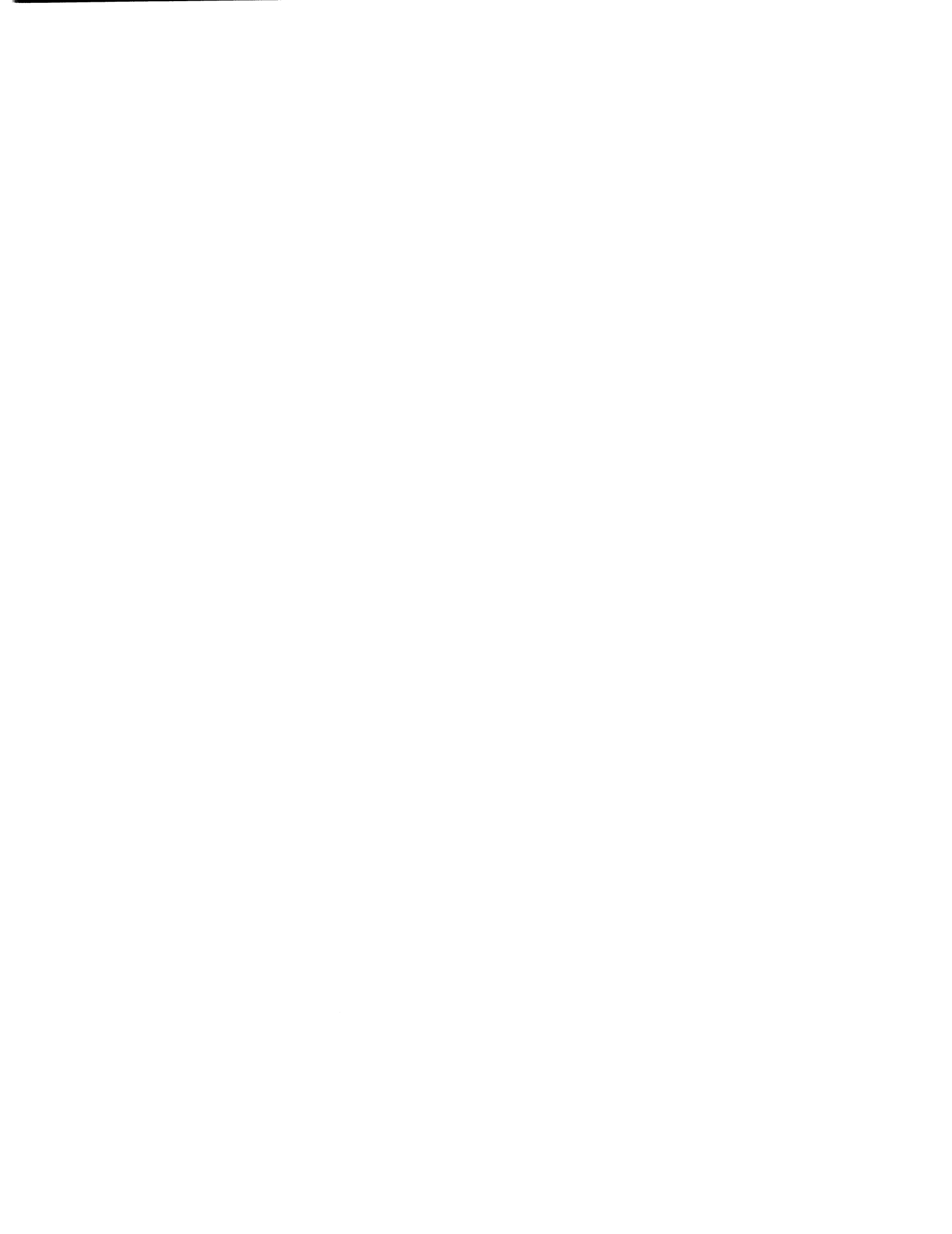
(Larry:) [missing from the tape]. (Bart:) Basically, if you use an OCCURS, you limit how you can access that subordinate data. If you have a very well defined application where some data is definitely subordinate to a main data piece, and I'm absolutely positively always going to be getting the subordinate data with the main data then an OCCURS might make sense. The problem is that when you do an OCCURS, it makes it difficult to get to the subordinate data only. (Larry:) It's really hard to manipulate within an OCCURS clause. You are better off putting it in another domain and use a CROSS and treat it in a relational way.

(Joe:) I'd like to make a dissenting opinion. There are some applications I've used in a medical database where the data is naturally hierarchical: and because the underlying data is naturally hierarchical, DTR is, in my opinion, the best tool for accessing hierarchical data. There are certain prices you must pay in order to do that, but I would argue the other way around. If the data is naturally hierarchical in the way it's used, it should be stored hierarchically, either in an OCCURS within a domain, or in separate domains using a VIEW which in effect creates a hierarchy. (Bart:) [section missing from tape].

How do I pass information from DTR back to DCL?

[name of questioner missing from tape:] ... that logical names created through DTR are user mode logical names. Is there any way that [DTR can create logical names in other modes: question wasn't finished as Andy Schneider was shaking his head "no"]. We use DTR in certain situations to pass values out to DCL and use those values: now we can't do that.

(Andy:) FN\$CREATE_LOGICAL creates a user mode logical name for DTR's purposes only, because 9 people out of 10 will use the logical name while in DTR to optimize [an application] and don't want to have it "kicking around" afterwards so everyone picks it up. What I would suggest is if you want the logical name to be [in existence] afterwards is to use a DTR procedure to create an indirect command file that you execute when you leave DTR to create the logical names. (questioner:) just write it to a file basically. (Andy and Larry:) or create your own function [to create a logical name in some table or mode other than user]. And submit it to the DECUS library.



DATATRIEVE and RMS

Joe H. Gallagher
Research Medical Center
Kansas City, MO

Gary Friedman
Montgomery Engineering

B. Z. Lederman
2572 E. 22nd St.
Brooklyn, N.Y. 11235-2504

Transcribed by B. Z. Lederman

Abstract

This is a transcription of a panel presentation on some of the important features of RMS as seen from the perspective of the DATATRIEVE user. It will give some basic definitions, list some of the options available to users, and how some tools may be used to optimize performance. The usual convention of placing square brackets around material interpreted or supplied by the editor is followed in this paper, as is the use of DTR as an abbreviation for DATATRIEVE.

What is RMS?

RMS stands for Record Management Services: it is a set of system services which provide for a uniform method of accessing data in files. It is built into VMS, and comes with the PDP-11 operating systems, it supports several types of file access (sequential, relative and indexed), several types of data records (fixed and variable length), arbitrates file sharing and block locking (some of these functions are being moved to other parts of the operating system, particularly within VMS clusters), and controls the conversion of data on the disk (tape) to within your program. In short, it's the way of getting stored data into your program.

Types of files.

Sequential files are the simplest, they are the smallest (least amount of storage space) for a given amount of data, they are compatible with programs that don't use RMS, can be stored on magnetic tape, are easier to transmit over communications lines, and generally have the fastest access and least overhead when the data is going to be accessed sequentially. The catch is that most applications don't access data sequentially, and even when they do there are some possible drawbacks to sequential files. For example, new records may only be inserted at the end: if the file is sorted in some

order and you have to add a new record, you must then re-sort the file. They can also normally only be deleted from the end of the file as well. As processing must be sequential, if you want to retrieve a record in the middle or end of the file the only way to get to it is to start at the beginning and read every record until you get the one you want. Sharing the file is limited to read-only for all accessers (this may change in the latest release of VMS).

Indexed files can be read sequentially or by one of the keys. Keys can have duplicate entries or no duplicates, there is automatic sorting in that data is automatically kept in order by the primary key so a sequential read is automatically sorted, and files can be shared for read and write. There is higher overhead in accessing the file (than for sequential), they can only be stored on disks (it will automatically be converted by most backup utilities when stored on tape, but you cannot have indexed access directly to a file on tape), and the file is larger as you are storing both the data and the index information in the file. Records may be added and deleted from any point in the file, but deleting a record does not recover all of the space used until the file is compressed or reorganized. Indexed files must have one primary key, and the data in that field may not be modified: it can only be deleted and the entire record replaced. (Secondary keys may be modified, or modification may be prohibited as you choose when you create the file.) In nearly all applications, the advantages over sequential files far outweigh the disadvantages, and indexed files will be used in nearly all DTR applications.

Buckets.

A bucket is a logical division of the space on the disk. Disks are divided into blocks: in order to use a disk, the space must be divided into manageable chunks, and all DEC disks are divided into blocks of 512 bytes. [A few older devices have smaller "blocks", but the software makes them look as if they had 512 byte blocks.] The data records you are using may be larger or smaller than 512 bytes, so the file is divided into buckets: each bucket holds one or more of your data records, and buckets are stored on the disk (as one or more blocks). One of the options available is to select the bucket size for a file. On the PDP-11, in order to conserve pool space, it should almost always be the smallest bucket size into which your record will fit, and DATATRIEVE-11 will automatically select this size. On the VAX, the option exists to choose

a larger bucket size, which may or may not help performance. A larger bucket size means there are more records stored in one place, and retrieving one bucket from the disk gets you several records from the same area of the file. If you are processing the file sequentially, when you read one record you automatically get the next few records at the same time, and so when you are ready to process the next record you already have it. This will save disk accesses, and generally improve the performance of the program. If, however, your accesses are scattered more or less randomly through the file (for example, you have a telephone directory file and you are not looking up people in alphabetical order so that file retrievals are scattered throughout the whole file in no particular order) then a larger bucket size won't help, and may even hurt a little by having to read extra data from the disk that won't be used. If you have an application where you read a record and then will probably need the next few records for related processing, or you have multiple records with the same key and may need to read some or all of them at the same time, then a larger bucket size may help by obtaining more data with each disk access.

File Prolog Type.

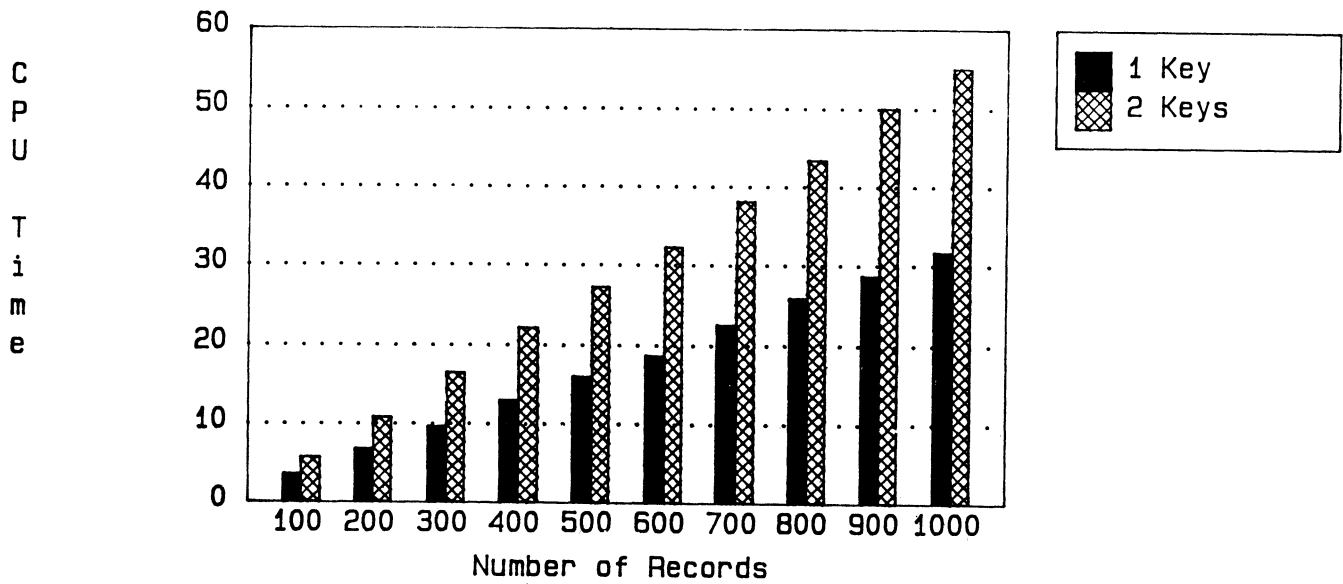
When you create a file (or display the attributes of an existing file), one of the attributes is the file Prolog, which can be Prolog 1, 2 or 3. Indexed files can be Prolog 2 or 3. Prolog 3 files have a tradeoff between speed and size: the index can be compressed, which will save space on disk, but will require more CPU work to compress and expand the data when needed. Also, it was mentioned before that deleting a record from an indexed file left a little unusable space in the file: with a Prolog 3 file, this space may be reclaimed more easily than with a Prolog 2 file with the "CONVERT/RECLAIM" command. [See also some discussion at the end of the paper about access speed.

Alternate Keys.

For each key in an indexed file, some work must be done to store the index information whenever a record is added to the file. The graph in Figure 1 shows the result of a test comparing the time needed to store records in a file with DTR when there was one key, and two keys, and shows the increased overhead. However, Figure 2 shows how much work can

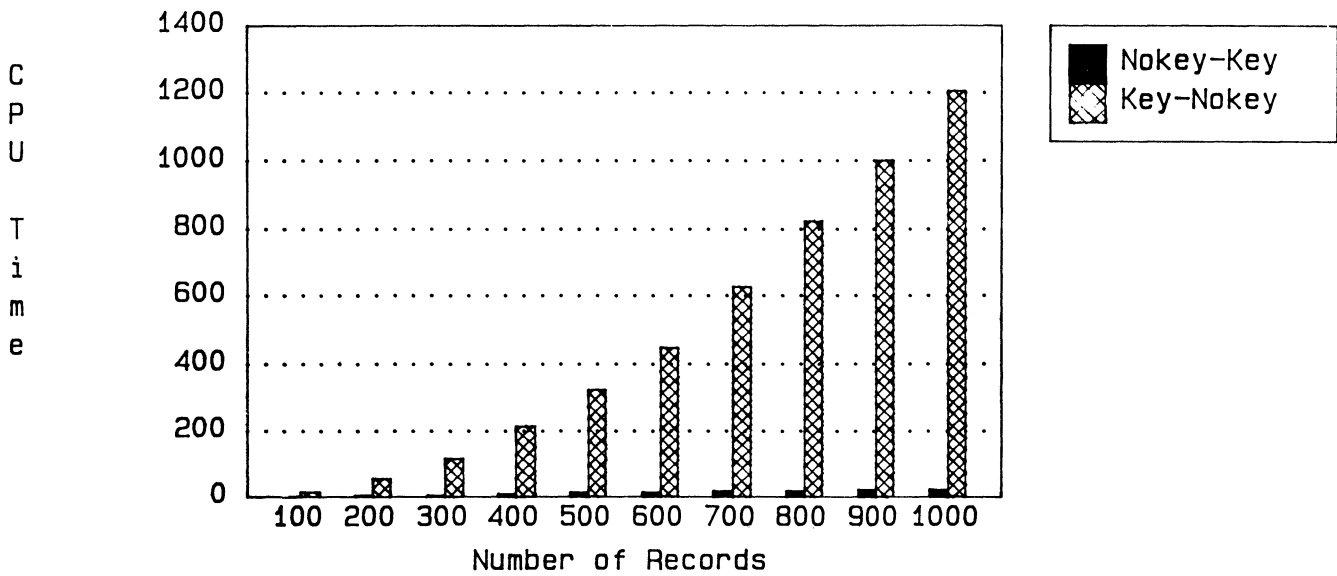
One Key versus Two Keys

Update Time



Order of Keys versus Non-Keys

in CROSS statement



be saved when retrieving with a key as opposed to retrieving without a key: in the case shown here, retrieving the second file in a CROSS (or a VIEW) without a key requires several orders of magnitude more work than if a key is used (this is cause of one of the most common complaints heard from DTR users, that a CROSS or a VIEW is slow: the second file was not being retrieved with a key). The result of this is: if you will be retrieving data fairly often by a particular field, it should be keyed as the time saved in retrieval is much greater than the time taken during storage. If, however, a field won't be used for retrieval it should not be a key, as the extra work for storing all records won't be recovered.

Creating a file.

You can create a file with the "DEFINE FILE" command in DATATRIEVE. This will give you a file which will always work, and will have all of the keys in the right place with the correct data type. It may not give you a file which is optimum for your particular application and data, however. You can also use one of the RMS utilities (DFN on the PDP-11, CREATE or EDIT/FDL on the VAX): this can be quite a lot of work as you have to figure out where (in bytes) in the file each key is, and what data type it is. I recommend that you first create the file with DTR, then use the RMS utilities to examine, and if necessary modify, the file to your particular needs.

Loading a file.

Loading a file all at once, with one of the RMS utilities, is a different operation than storing single records (as with DTR). The utilities (IFL on the 11s, CONVERT on the VAX) do an optimized file load: they sort records, pre-allocate disk space (for data and index), and store the information. If you store individual records, it will be inserted in the middle of the file if it can: if the file was loaded with a fill factor of less than 100%, there will be empty space in the middle of the file to receive extra records. The resulting file will still be in good order, and performance will be good. If the file is already full, then RMS does what is called a bucket split: a pointer must be put into the file to point to another area of the file, which will then contain the data. Subsequent read operations are slowed a little as you have to "jump around" in the file to follow the pointers to an alternate area and back again. Generally, you

want to avoid this. Similarly, there is the space that the file has requested on the disk: if there is extra empty space in the file, new records will be added into this space and all of the data will be close together. If the file has run out of space, then the operating system will try to find more on the disk (subject to user quotas, etc.). If space is available next to the existing file, it will be added, but it is possible that the next free space on disk may be physically distant. This is known as a "fragmented" file, as it is stored in several separate pieces on the disk. This results in a performance degradation, and should be avoided. When a record is deleted, there is a small amount of space which cannot be immediately recovered (until the file is reorganized).

DATATRIEVE is not the best tool for doing a complete file reorganization: the RMS utilities which were written specifically for this purpose will yield better results. If a file is being added to or modified frequently, then it is a good idea to use one of the utilities to re-organize the file at regular intervals.

Some Tools.

There are a number of utilities that give useful information about your files. On the VAX, ANALYZE/RMS/FDL will yield a file full of information about the analyzed file, and you can look at this with ordinary editors or with EDIT/FDL. Much of the information will not be of interest to casual DATATRIEVE users, but a few items are important.

ALLOCATION is how many blocks on the disk are reserved for this file.

BEST TRY CONTIGUOUS means that if the file runs out of disk space and the operating system must get more, it will first try to get contiguous blocks (those immediately next to the existing file), but if it can't it will get what space it can. If the file was marked as CONTIGUOUS only, when it runs out of space if there is no more contiguous space available the attempt to expand the file will fail with an error message.

CLUSTER SIZE is generally set by the system manager for a given disk.

EXTENSION is how many blocks of disk space are added to a file by default when it has to be extended. If you know that you will be adding many

records to your file, especially if they will be added at one time, then you should specify a larger extension value to get a large piece of disk space each time the file is extended: this will minimize fragmentation, and will make the application run faster as adding space to a file is a relatively slow process.

GLOBAL BUFFER has to do with sharing files, and will not be set by most users: it needs to be considered on a system wide basis.

BLOCK SPANNING: if a record spans a disk block, then you have to read both disk blocks to retrieve the record: this may be a little slower than if the record were in one block only. If your records are smaller than one block (512 bytes) in size, and you want every fraction of performance, and are willing to leave a little space at the end of each block empty (wasting a little disk space), then NO SPAN BLOCKS may give you a little extra performance, though I suspect that in most cases the improvement will be minimal.

Allocations for areas have to do with how much space is reserved for data and keys. Rather than attempt to calculate these, let the RMS utilities set them, or use one of the optimization scripts in EDIT/FDL to set them.

Each key has it's own section for description. One of the fields in this is the name of the key. DTR does not put this information into the file: I highly recommend for documentation and maintenance purposes that you obtain an FDL description for all of your DTR data files, and that you fill in the name of the field into the FDL key name so that you will know which keys correspond to which field in your DTR application.

You can use this file to select some options that are also available in DTR, such as allowing or preventing duplicates and changes. You can also enable or disable data compression and key compression. As noted before, this may save disk space as the cost of performance. It is difficult to predict how much compression will be done on a file, so you will probably want to load the data, then ANALYZE the file and see what happened. There may be a performance trade off between compressing to save space, and the work needed to compress and decompress the data and keys. [See also the discussion transcribed below.]

FILL can be a very important parameter. If you are working with a file whose data is fairly static

(does not change often), you will probably want a fill factor of 100%, or very close, to save disk space and keep the data close together on disk for fast access. If you have a file which changes often, or to which data will be added, then you want to avoid the bucket splitting problem by using a lower fill factor during the initial load so there will be empty space scattered throughout the file to receive new records. The RMS utilities honor the fill factors to leave empty space in the file: DTR does not, so it will use the extra space for new records.

If the file has data in it, ANALYZE/RMS/FDL will also give you information about how much reclaimable space is in the file which is not being used, and how much compression is being done on the keys and data. This is a good indication of the state the file is in, and can also be used by EDIT/FDL optimization scripts to design a better file.

Built into the FDL editor are some scripts which work quite well in designing better files, especially if you analyze an existing file filled with data. Generally, a "flatter" file is one with better performance: the fewer the number of index levels, the fewer disk accesses will be needed to find a particular record, and this results in better performance. DIRECTORY/FULL on the VAX (DIR/ATT or DSP on the 11) gives information about the number of keys, Prolog type, number of blocks allocated, etc.

Example of reorganization.

We found that when a number of records had to be added to a large indexed file with many keys, it was better to add the new records to a separate sequential file, convert the large indexed file to a sequential file, append the file with the new records, then use the combined data to re-populate an indexed file. This process can be done in a batch mode, to save time and I/O processing. It avoids the problems of adding records to indexed files (such as bucket splitting), especially when the updating is done in "chunks".

We did some benchmarks with a file containing fixed length 110 byte long records, with 5 keys, containing a total of 1778 records. DATATRIEVE was used to write the new data to the end of the sequential file that contains the updates, and again to write the sequential records back in an indexed file. This was found to take much less time than to put records into an indexed file.

We also tested the use of the CONVERT utility to do the file conversions. This was found to take much less time and system resources: the improvement is much greater than that which can be obtained from typical "system tuning" efforts, which usually try to adjust system parameters for a 5% or so improvement. Adjusting the data file parameters can yield a much greater improvement.

Tests were performed on a stand-alone 11/780 (no other users), comparing DATATRIEVE with CONVERT to do the batch update. CPU usage was reduced by about 3 orders of magnitude, I/O operations were reduced from about 35,000 for DTR to 1700 for CONVERT, Elapsed time was reduced from 14 minutes to 2 minutes. The reduction in I/O operations is especially significant for many applications.

These tests were done using the same DATATRIEVE file definition. By using one of the FDL optimize scripts, the file can be further tuned to the particular application. You should load the data into a file first for best performance, and look at values for compression, etc. If compression shows up as a negative value, turn compression off. The scripts are nearly automatic in operation: simply select the optimize script. (Also look at the *VMS Guide to File Applications* manual: it may take a while to absorb all of the contents, but it's worth the effort.) We did a one pass optimization of our test file: not a lot of tuning, just some changes to bucket size, key compression, etc. When we ran the load test again, I/O operations were cut in half, Elapsed time was cut in half, and CPU time improved a bit. It took no more than 15 minutes to do the optimization with FDL.

General Hints.

Big updates will cost you, as you have to work with all of the indices: use CONVERT rather than DTR. Tune your files: a little effort here yields considerable benefits. Update in batch: you can off-load operations to times when the system is less used.

Some final examples.

I compared the time it takes to populate the YACHTS file, as is done during installation of DTR. These tests were done with the PDP-11, copying the installation verification procedure. An empty indexed file is created using DEFINE FILE (and I even improved the procedure by adding the ALLOCATION

clause), and DTR then reads the sequential file from the distribution kit and stores the indexed file using a FOR loop. This took 2 minutes and 46 seconds. I then create an identical empty file using DTR again (no optimization work done), and use IFL (the PDP-11's equivalent of CONVERT) to populate the indexed file and it takes 17 seconds. The reason for the difference is that DTR is a general query and report language whereas IFL is written for the sole purpose of populating indexed files.

Disk fragmentation: this is not directly an RMS problem, but it is something to know about. A fragmented disk causes performance degradation, and usually implies fragmented files as well. As you create and delete files, and extend files, your free space tends to be scattered about the disk in small chunks. All of the normal backup utilities (BACKUP, BRU, DSC) do disk compression as they copy disks. One way to find out if your disk is fragmented is a utility called FRAG, available through the DECUS library or SIG tapes for VMS and RSX. Because fragmentation happens gradually while the system is used, it can be a rather subtle performance degradation that may not be immediately obvious. Fragmentation is an important reason for making backup copies of your disks fairly often. If you have fixed media disks (Winchesters) and are backing up your disks to tape, you are not compressing the disks, which are probably getting more and more fragmented. You should either get extra disks (if you can) and copy disk to disk, or else you have to copy from disk to tape, then back from tape to disk to get the benefits of compression. If you have removable disks, you should copy from disk to disk, put the old disk on the shelf as a backup and run with the new disk. This not only gives you the advantages of disk compaction, it also tells you very quickly if the copy procedure worked, and if it didn't your original disk is safely on the shelf. (There are many people who backup to tape only, or backup to disk and put the new disk on the shelf, and it's only when a disaster occurs that they find out that their backup copies aren't usable.)

Questions and Answers.

Steve Hicks, Rockwell International.

Question: When you create a file with DTR, is it CONTIGUOUS or CONTIGUOUS_BEST_TRY?

Answer: sometimes.

Question: Will using FDL to set it CONTIGUOUS gain anything?

Answer: perhaps, but certainly you should get the FDL file and check.

Question: Can you change a Prolog-2 to a Prolog-3 with a set command in FDL?

Answer: you can specify it with the FDL editor.

Question: I have a file which which has fields which are not keys. Will they be compressed?

Answer: you can specify data compression independently of key data compression for this. (Joe:) Note that compression works on adjacent bytes with the same value. If your field is filled with different characters, you don't get compression. If your field is all blanks or zeros, or some characters followed by trailing blanks, for example, then you do get compression. You need strings longer than 4 or 5 bytes to get an advantage from compression. (Gary:) There is a system parameter to set Prolog-2 or Prolog-3 as the system wide default, but there is no system wide parameter to set data compression on or off as a default. (Unidentified comment:) There may be a speedup in looking up keys when compression is on. [Much of the later comments were off the microphone: apparently the user saw a considerable improvement.] (Gary:) We just ran into problems where we tried data key compression, and found the system was deleting records. There were about 60,000 records in the file, and CPU usage greatly increased. (Joe:) It depends on the data: you cannot make a general rule of thumb about files going faster or slower with key compression. The other issue has to do with the nature of the key. If adjacent keys have a lot of commonalty (for example, the keys are values such as 12345, 12346, 12347, 12348), then you get a lot of compression: if the keys are vary different, you get much less compression. A file which has a lot of compression may well see a performance improvement. [Audience comment not audible: apparently the key in question was a ZIP code.] A ZIP code would an ideal case where you have a lot of duplicate or similar keys, resulting in considerable compression. Other types of data might not work as well.

Warren Alkire, Abilene Christian University.

Question: Once you have your FDL file, do you ever need to edit it again (once it's optimized).

Answer: once you have the file in a state where you are satisfied with the performance, quit. Sometimes you can keep on tuning, other times FDL my indicate that this is the best you can get.

Question: I may convert the file, then a week or two later may need to convert it again. Do I use the same FDL file?

Answer: if the nature of the data you have has changed significantly, then maybe you want to optimize again. If the nature of the data hasn't changed, and you are just adding records, then you can use the same FDL file.

Rick Trane, (?) Engineers, Milwaukee.

Question: You stated that if you delete a record, it is physically still in the bucket, and is not removed until you do a reclaim on the file, is that correct?

Answer: some of the space does get reused. What happens is that there is a pointer in the bucket saying that there used to be a record here. If there is still enough free space in that bucket to hold a new record and a new pointer, then the space may be reused, but there is still an extra pointer from the original record. A lot depends on the size of the record: if the record is, for example, 480 bytes long and you have a bucket size of 1 which is 512 bytes, there is only room for one record and pointer. If you delete the record, there is not enough room for a record and two pointers, so the space does not get reused at all until the file is compressed. If, however, the records are only 20 bytes long, there is room for several records and pointers in a single bucket, and some space may be reused. (Joe:) If the file has a single primary key, under version 4 of VMS it can be deleted and reclaimed immediately: if there are secondary keys, it can't reclaim space, and you have to use CONVERT.

(Unidentified)

Question: Our problem is with a file that has alternate keys which allow duplicates. Any hints for access time?

Answer: you must understand that if a key is not unique and there are many duplicates (for example, if sex is a key in a personnel file and about 51% have the same key), you are essentially using half of a sequential file, and there isn't much that can be done for performance. The best performance occurs when keys are unique or nearly unique. [Remainder lost from end of tape.]

Procedure used to compare loading a file with DATATRIEVE and IFL.

This was placed in an indirect command procedure so no time would be lost typing in the commands. It was run on a PDP-11/70 using RSX and no other users.

Example of output from FRAG utility

Disk fragmentation statistics for DW1:

28-APR-86 09:18:40

Contiguous free blocks (Holes)

```
>PIP YACHT.DAT;*/DE
>DTR @DY
DEFINE FILE YACHTS
    KEY=TYPE(NO DUP),
    KEY=MODEL(DUP, NO CHANGE),
    ALLOCATION=30, SUPERSEDE
>TIM
09:02:04 02-DEC-82
>DTR @T1
READY YACHTS WRITE
READY YACHTS-SEQUENTIAL
FOR YACHTS-SEQUENTIAL STORE YACHTS
    USING BOAT=BOAT
>TIM
09:05:04 02-DEC-82
    [Elapsed time 2:46]
>PIP YACHT.DAT;*/DE
>DTR @DY
DEFINE FILE YACHTS
    KEY=TYPE(NO DUP),
    KEY=MODEL(DUP, NO CHANGE),
    ALLOCATION=30, SUPERSEDE
>TIM
09:05:04 02-DEC-82
IFL YACHT.DAT=YACHT.SEQ
    PRIMARY KEY:
    SORT HAS STARTED
    SORT MERGE PHASE HAS FINISHED
    ALTERNATE KEY(S)
    SORT MERGE PHASE HAS FINISHED
    ALTERNATE KEY(S) 1:
    NUMBER OF INPUT RECORDS: 113
    NUMBER OF OUTPUT RECORDS: 113
    NUMBER OF EXCEPTION RECORDS: 0
>TIM
09:05:21 02-DEC-82
    [Elapsed time 0:17]
>@ <EOF>
```

Hole Range	Frequency	Number Blocks
1 - 8	23	76
9 - 16	6	79
17 - 32	0	0
33 - 64	0	0
65 - 125	3	225
126 - 250	1	212
251 - 500	1	304
501 - 1000	0	0
1001 - 2000	0	0
2001 - 4000	1	3311
4001 - 8000	0	0
8001 - 16000	0	0
16001 and up	0	0
Largest free block		3311
Total free blocks		4207

EDUSIG

TEST GENERATION AND COURSE MANAGEMENT WITH DEC'S CML,

COMPUTER MANAGED LEARNING SOFTWARE

Claude M. Watson
Lansing Community College
Lansing, Michigan

ABSTRACT

This session will feature the Computer Based Training software developed by CBTS of Canada and marketed by Digital in the U.S. Test generation and course management are combined in a single integrated system. The power and flexibility will be discussed. Topics covered will be: features, description, success and popularity, equipment and support for variety of options, and training required for best use. This report is based on two years experience.

This session was presented by Claude M. Watson, Divisional Computer Coordinator, Arts & Sciences Division, Lansing Community College, Lansing, Michigan and described LCC's experience with an educational management software package.

Introduction to CML

The Computer Managed Learning (CML) system is a software package designed to manage learning activities in an educational environment. The system was developed and marketed by Computer Based Training Systems, Ltd, (CBTS) of Calgary, Alberta, Canada. The main features of the system are menu access to the various CML options, a test generating system, and a course management system. The CML system runs on VAX computers using the VAX editor EDT extensively and supports integration with DEC software such as CAS-DAL, REGIS, etc. While it is designed especially for independent study types of courses, its features can be adapted to manage traditional classes as well.

History of Computer Course Management at LCC

Prior to obtaining CML by CBTS, LCC faculty had developed extensive test generating systems for two independent study courses on a Hewlett Packard computer. These systems were complex and dependent on support from staff members with other responsibilities. When new computing facilities were obtained, support became increasingly difficult to maintain and the courses began to shift back to traditional classroom courses.

Introduction of CML by CBTS

In the fall of 1982, the college received a Title III grant to assist faculty in achieving computer literacy and to pilot both the training of staff and the introduction of the computer to the classroom. By the following fall, the CML package had been identified, evaluated and obtained. The CML software package was installed and a five day intensive training program was conducted by CBTS. Eight Computer Assisted Instruction (CAI) Coordinators and trainers participated. Following this training, a

pilot project was begun in 1984 with the Rocks and Stars course. Workshops were then developed for the training of other faculty.

The implementation of CML generally has followed three steps at LCC. The first is training, second is the development of test banks, and finally, course management is implemented. At this time, 27 courses have test banks in the development or implementation stages.

Features of CML

The menu overview, Figure 1, shows the key features of CML. Figure 2 shows the actual Main Menu 1 as it appears on the terminal screen. The general categories which the user may select, as shown in Figure 1, are:

(1) Item 2.0, Testbank. Test bank editing, test bank reports, and test generation are controlled by the Test Bank Menu 2.0 (Figure 3).

(2) Item 3.0, Course. Figure 4 is the Course Menu 3.0, which controls the creation and editing of a course, as well as choices to create student records.

(3) The remaining menu items 4.0 through 9.0 identify the support for other DEC products, course reports, and the student menu that provides the supporting structure for operating a course.

A very important feature of CML test banking is the classification scheme for cataloging questions (Figure 5). Figure 6 diagrams in three dimensions the "MMOQQ" coding for numbering the questions with: MM, module number up to 99, O, objective number up to 9, and QQ, question number up to 99. This gives a possible total number of questions of $99 \times 9 \times 99$, or 88,209 for each test bank.

In addition to identifying the questions with five digits, each question has a header, described in Figure 5, that is 14 more digits long. This provides a very flexible scheme for identifying key features for classifying questions, such as type, value, degree of difficulty, lock level and cognitive level. A variety of kinds of questions can be used, such as multiple choice, true-false, problems with randomly generated variables, essay and completion. Questions

can be classified to produce general tests for a course, pre-tests, tests by topic, by level of difficulty, or by instructor where more than one instructor is involved in teaching a course.

The Course Management portion of the CML system also makes use of the VAX editor EDT to allow the user to organize the sequence of course objectives, the testing methods, and tracking of the student's progress through a course. Figure 7 is a three dimensional view of the main parameters involved in "mapping" a course. A course can be composed of up to nine subjects with 99 testing points each. Each of the testing points can have up to 19 decision options. Independent study courses can be organized and managed in a variety of ways. For traditional courses, norm referencing is available.

Training and Training Materials

A system that allows so many options and the ability to meet so many educational challenges requires training for its users to be able to effectively take advantage of its potential. LCC's initial training for trainers was provided by CBTS at the time the software package was installed. One of the initial faculty trainees, in cooperation with LCC technical staff, began a pilot program which included establishment of a test bank. This program served as a model as training proceeded for other faculty. Training of additional staff was offered in various workshop or seminar formats in two categories: test banking and course mapping. Development of test banks is a natural first step, so two types of workshop formats were developed in this area. The most effective was two full, consecutive days of training. The second option was one hour per week over a five week period, with lab time provided additionally. Training for course management has been offered in the same choice of formats. Knowledge of or experience with test banking is a pre-requisite.

Materials developed for use in training workshops have also proved useful as aids to self-study.

Summary

The CML system provides a set of programs to support a wide variety of learning environments. Users can use a few or as many as they wish. Inherent in the system are the tools to analyze and improve the testing and the delivery of instructional materials. A tribute to the power and flexibility of the CML software has been the many ways and levels at which the various departments at LCC have implemented the system.

In order to provide these options and flexibility, the system is necessarily complex. To make the most effective use of CML, ongoing training is important. Effort and support are required. At LCC, faculty and students have been enthusiastic about the benefits obtained. CML is clearly superior to previous management systems developed in-house.


```

*****
* CML 2.7          MAIN MENU 1.0          cbts *
* ===== *
*                               PF2=HELP PF4=EXIT *
* Please select one of the following *
* or type Help for additional information: *
* * * * * * * * * * * * * * * * * * * * * * * *
* 1. EXIT          - Return to CML ID Menu or DCL *
* 2. TESTBANK     - Display Testbank Menu *
* 3. COURSE       - Display Course Creation Menu *
* 4. STUDENT      - Display Student Menu *
* 5. ANALYSIS     - Display Course Analysis Menu *
* 6. USER_PRIVILEGES - Display User Privileges Menu *
* 7. UTILITY      - Display Utility Menu *
* 8. NORM_REFERENCE - Display Norm reference Menu *
* 9. CBI          - Display CBI Development Menu *
* * * * * * * * * * * * * * * * * * * * * * * *
* Enter option number or name and press <RETURN>: _____ *
* * * * * * * * * * * * * * * * * * * * * * * *
*****

```

Figure 2. Main Menu 1.0

```

*****
* CML 2.7          TESTBANK MENU 2.0      cbts *
* ===== *
*                               PF2=HELP PF4=EXIT *
* Please select one of the following *
* or type Help for additional information: *
* * * * * * * * * * * * * * * * * * * * * * * *
* 1) EXIT          - Return to MAIN menu *
* 2) CREATE        - Create a new test bank *
* 3) EDIT          - Edit an existing test bank *
* 4) CHECK         - Check testbank for errors *
* 5) MATRIX        - Difficulty & cognitive level matrix *
* 6) CHARACTERISTICS- Question characteristics listing *
* 7) DOCUMENT      - Access to curriculum bank documents *
* 8) EXAM          - Draw exams from testbanks *
* 9) GUIDES        - Display module study guide menu *
* * * * * * * * * * * * * * * * * * * * * * * *
* Enter option number or name and press <RETURN>: _____ *
* * * * * * * * * * * * * * * * * * * * * * * *
*****

```

Figure 3. Testbank Menu 2.0

```

*****
* CML 2.7                COURSE MENU 3.0                cbts *
* =====*
*                               PF2=HELP PF4=EXIT        *
*   Please select one of the following                    *
*   or type Help for additional information:              *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   1) EXIT           - Return to MAIN menu              *
*   2) CREATE         - Create a new empty course file   *
*   3) EDIT           - Display a Course Edit Menu       *
*   4) ROSTER         - Enroll or remove students       *
*   5) SUPERVISE     - Instructor Reporting and Managing *
*   6) EXAM           - Generate single course exam      *
*   7) MULTI_EXAM    - Generate range of course exams    *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   Enter option number or name and press <RETURN>:_____ *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*****

*****
* CML 2.7                COURSE EDIT MENU 3.3            cbts *
* =====*
*                               PF2=HELP PF4=EXIT        *
*   Please select one of the following                    *
*   or type Help for additional information:              *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   1) EXIT          - Return to Course menu            *
*   2) EDIT          - Edits the Course File            *
*   3) SUBJECTS     - Give a list of Subjects          *
*   4) MAP           - Give a Map of a Subject          *
*   5) LIST          - Course Decission Listing         *
*   6) SUMMARY      - Give a Summary of Group Exams    *
*   7) RESOURCES    - Estimate of Terminal Resources   *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   Enter option number or name and press <RETURN>:_____ *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*****

```

Figure 4. Course Menu 3.0 and Course Edit Menu 3.3

Coding system - Test Questions

Question_Characteristic_Coding

Each question can be 100 lines long, 00-99 coded as LL appended to the location coding resulting in line coding of MMOQQLL. Line 00 of each question contains the characteristic coding as follows:

MMOQQ

There are a possible 99 modules.
Each module can have up to 9 objectives
with 99 questions for each objective.

MMOQQ00 IITBLLCVNDDFAnswer(s)
where II = instruction statement number 01 - 99
TT = question type = 01, 02, 03, 05, and 09
 = 01 for Multiple Choice
 = 02 for True/False
 = 03 for Short Answer (completion/matching/etc.)
 = 05 for Problem solving (self generating)
 = 09 for Assignment (non computer marked)
BB = Blank Lines left after question printing 01 - 99
LL = Lock Level (01, 02, 04, 08, 16, 32, 64) Total = 127
C = Cognitive level currently only lock level masks from 1
 to 9 can be used when setting up course decisions.
 categories: K, C, A, OS
VV = Question value 01-99
N = Number of answers expected 0 - 9
 = 1 for Multiple Choice and True-False
 = 1 to 9 for Short Answer and problem solving questions
 = 0 or 1 for Assignment questions
 if 0 the question is assumed answered when issued, other
 questions are marked normally.
 If 1, the question and any others on the exam can not
 be marked until the instructor receives the assignment
 and clears the student via the Main program.
DD = Degree of Difficulty 0 - 99
F = Future expansion fields
Answer = Correct answer expected.
 Multiple choice: 0 to 6 or A to F (must be capital letter)
 True False: T or F
 Short answer: each answer terminated by a ; (semicolon).
 Alternate correct answers separated by an &.
 Problem solving: no answer in answer field required.
 Assignment: Can be coded answer known only to instructor.

Figure 5. Testbank Classification Scheme

CML TESTBANK STRUCTURE

CBTS60

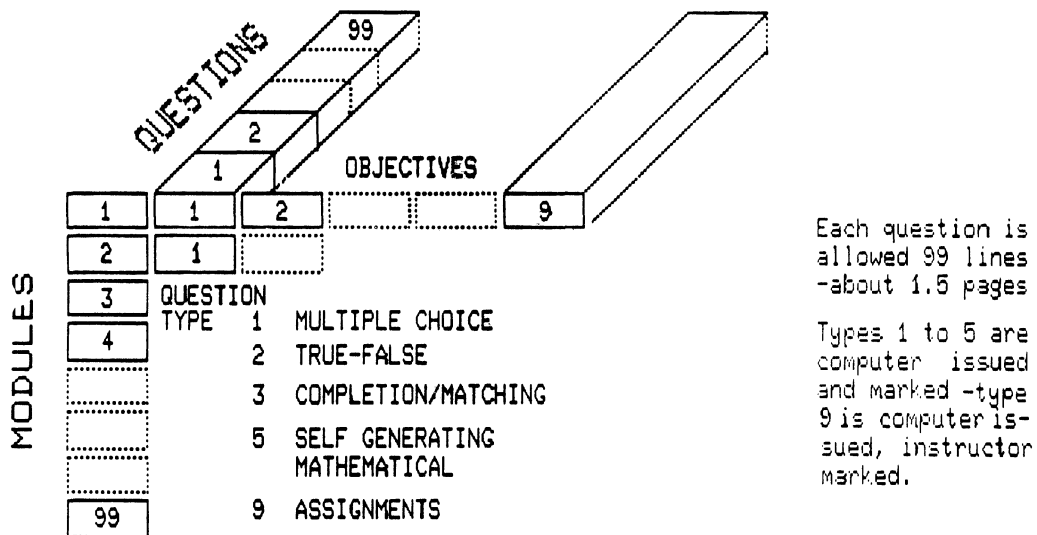


Figure 6. Three Dimensional Testbank Coding Diagram

CML COURSE STRUCTURE

CBTS59

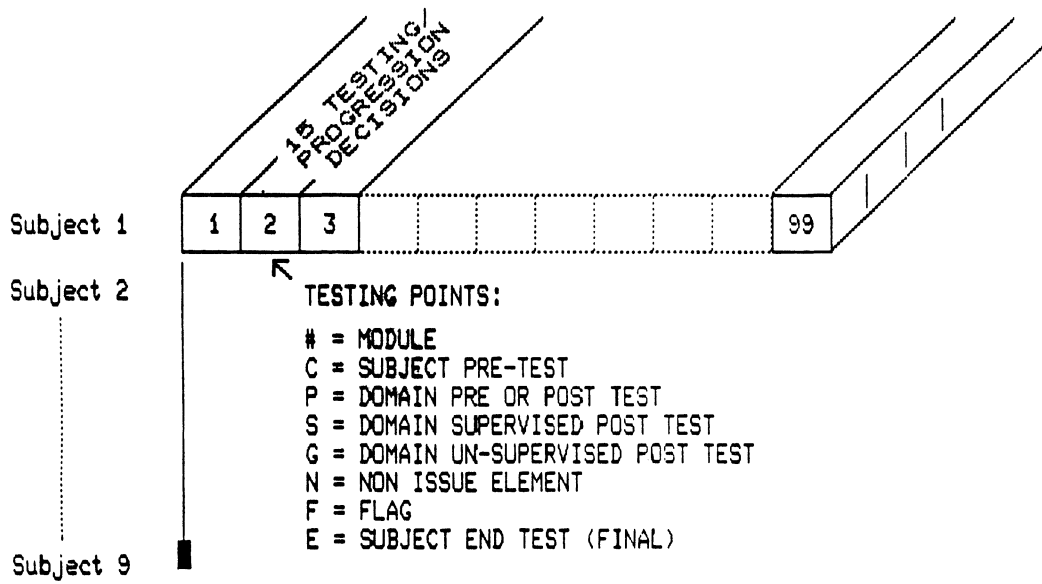


Figure 7. Three Dimensional Course Management Diagram

THE MANAGEMENT OF COMPUTER RESOURCES
WITH A WELL DESIGNED ACCOUNTING STRUCTURE

By Philip A. Dawdy, System Manager of a VAX 11/780
Lansing Community College
Lansing, Michigan 48901

ABSTRACT

This paper describes good system management techniques as well as specific accounting and file structures used to manage Lansing Community College computer resources.

OVERVIEW

LCC's VAX is primarily used for academic applications. All computer sites are different, and therefore the material presented is not to be construed as the only method for effectively managing a computer installation. The ideas are provided to convey concepts which you may or may not find applicable for your particular site.

The following is a summary of our VAX installation:

VAX 11/780 (w/FPA), 8 MB memory, (2) 500 MB hard disks, tape drive, 96 DZ-11 ports. The 96 devices consist of 46 VT100s, 7 VT125s, 13 VT241s, 14 GIGI terminals, 2 modems, 6 matrix printers, 6 laser printers, 1 line printer, and 1 plotter.

Software installed is VMS 4.3, FORTRAN, BASIC, PASCAL, 20/20 spreadsheet, SPSSX stat. package, CML (Computer Managed Learning), CAS (Courseware Authoring System), and GIGI software.

Over 1000 user accounts are authorized on our VAX. More than 50 percent of these are for computer science courses. During peak loads, 35-40 concurrent users is the maximum.

Students use the VAX for computer based education and for learning computer science. Faculty use it for course management and developing course materials. Staff and faculty develop lessons, drill and practice, and laboratory support. They also create procedures to facilitate user access to resources as well as run statistics on various courses.

IMPORTANT MANAGEMENT GOALS

The following goals are useful in all environments, but are considered important at our academic site:

- o Provide a user-oriented environment
- o Program automatic procedures
- o Document all developments
- o Provide technical support and training
- o Create common conventions (facilitates usage)
- o Provide a means for simple shared file access
- o Create well-organized file and account structures
- o Communicate with users and listen to requests
- o Inform users in advance of system changes

PRACTICES IMPLEMENTED TO FACILITATE USAGE

System-wide symbols are assigned to the most useful commands desired. User tasks are therefore simplified and made more efficient. Logical names are assigned with a common methodology and the more frequently used logicals are short -- thus easy to remember.

All commands that are added to VMS are documented. On-line help for each command is obtained via entering LCHELP. Documentation files are also available for the purpose of viewing or printing. By entering the command DOC, you can list or print any document in the LCC\$DOC directory. For the newcomer, the special file \$\$\$READ_FIRST.DOC provides a good starting place. \$OUTLINE.DOC is an outline of all documentation -- correlating the files with each.

A sample of our documentation files are: VAX_INTRO, VAX_ACCESS, VAX_HARDWARE, VAX_SOFTWARE, VAX_HELP, VAX_POLICIES, ACCOUNT_TYPES, ACCT_CLASS, DEVICES, DISK_STRUCTURE, COMMON_VMS_CMDS, LCC_LOGICALS, ENVIRONMENT, ENV_STRUCTURE, LIBRARY_USAGE.

ACCOUNT STRUCTURES AND CLASSIFICATIONS

Accounts are authorized by having the user fill out an "Account Authorization Request Form." This helps the system manager classify and file the account. The authorization procedure is automated and only requires five inputs per account. Entries include: "environment" (department or program), username, owner, password (defaults to username), and any additional accounting options.

At LCC there are five divisions, each division having several departments or programs. Under VMS V3, there was a severe limitation of file access via UIC's. The UIC system is much too restrictive and only allowed one level of hierarchy that places members into a specific group. This caused the division and its various departments to be placed into different groups. Computer science classes are each placed into different groups also.

The member number within a group determines another type of classification: group library, development leader, personal development, shared management, shared CML (Computer Managed Learning), shared CAI (Computer Assisted Instruction), class instructor, and class students.

At LCC there are three major types of accounts:

- o Personal Account -- private and permanent.
- o Shared Account -- usually captive and shared by many users. Four categories are used at LCC: CAI (Computer Assisted Instruction), CML (Computer Managed Learning), LIB (for library management -- obsolete), and class management.
- o Class Account -- private and temporary for instructors and students. Each student has their own account like the instructor. Since class accounts are never removed, all files are deleted and passwords are reset each term.

As was mentioned, under VMS V3 there was no way to create a hierarchy of environments that would place the division at the top, its departments at level two, and any department disciplines at level three. This desired scheme would allow accounts to access files at its own level or below.

With the new ACL (access control list) feature of VMS V4, the structure and access of files on the system could fit the desired hierarchical structure at LCC. This scheme has been implemented at LCC and has provided a much more flexible environment for every user while increasing the security as well.

Many users have wondered whether ACL's are a burden on the system. When few (less than ten) ACL entries are used, the extra overhead is very small -- as we have experienced. The use of special identifiers and wildcards helps decrease ACL entries. Most managers and DEC technical staff agree that there is little performance degradation for few ACL's.

ACCOUNT POLICIES

In addition to the normal policies of scheduling backup and assigning default protection schemes, LCC has many policies that are designed to help share the various computer resources equally. A detached program called WATCHDOG (nicknamed RABID_DOG) runs on the system to provide the following services:

- o Connect time is conditionally enforced when all terminals in a predefined area are active.
- o Idle (inactive) terminals are logged off after certain time periods. Users are given warning.
- o Under heavy loads: disconnected processes not running an image are logged off, interactive compiles and links are lowered priority, and GAMES account is not allowed access.
- o Special warnings are sent when CPU is low.

Even though computer science students can type listings directly in, the policies that follow help keep the temptation to copy other students programs low:

- o Mail cannot be received (however it can be sent).
- o Files cannot be copied from other student's.

Environment is defined as a common group of users that share most of the same resources. Files and logical names that reference files and devices are important resources to environments. There are three major classes of environments at LCC:

- 1) System (identifier SYS)
- 2) General (identifier GEN)
- 3) CBE (Computer Based Education). Environments are broken up by divisions and departments. A two or three character abbreviation is used to identify each environment. Names of logicals and root directories are derived from these short "identifiers" to maintain commonality.

The departments within a division are a sub-environment of the parent -- providing the hierarchy of file access as desired. At LCC, the Arts and Sciences division and Business division hold the following identifiers:

```
AS
  HUM COM MTH SCI SS  CPS
BUS
  MS  OS  MDC CAS HS  CJS AFS
```

DISK FILE STRUCTURE AND CORRESPONDING LOGICALS

Figure 1 illustrates the structure of environments for the user disk directories. The top-level directories represent the divisions. The directories at the second level represent the departments.

All "U\$" entries signify a user disk environment root. For example, U\$AS is the Arts and Sciences division main root. Each root directory contains four main subdirectories for the various types of accounts. Personal account directories are placed in the [.PER] subdirectory of the environment root.

Logicals U\$AS: and U\$AS_: point to the AS divisional roots. U\$AS_: is a root of the U\$AS directory. U\$AS: is a root of [.PER]. Therefore, account RED's personal directory is accessed via U\$AS:[RED].

Figure 2 illustrates the structure of the system disk shared library directories. All "L\$" entries signify a library for each environment. Again, the hierarchy is maintained for both structure & access. Logicals for the libraries are prefixed with "L\$". Therefore, L\$AS: is the library for the AS division.

To access your own environment library, the LIB: logical used. It is assigned automatically at login for all users. Therefore, L\$env: logicals are only needed for other environment libraries.

VMS does not allow you to access a subdirectory via a non-rooted logical that points to the parent directory. At LCC, we have set up a convention for easy access to subdirectories. By appending an underscore to a logical that points to a directory, you may then use the new logical to access its subdirectories. Therefore, LIB_: allows you to assess subdirectories in your default library.

User Disk Structure

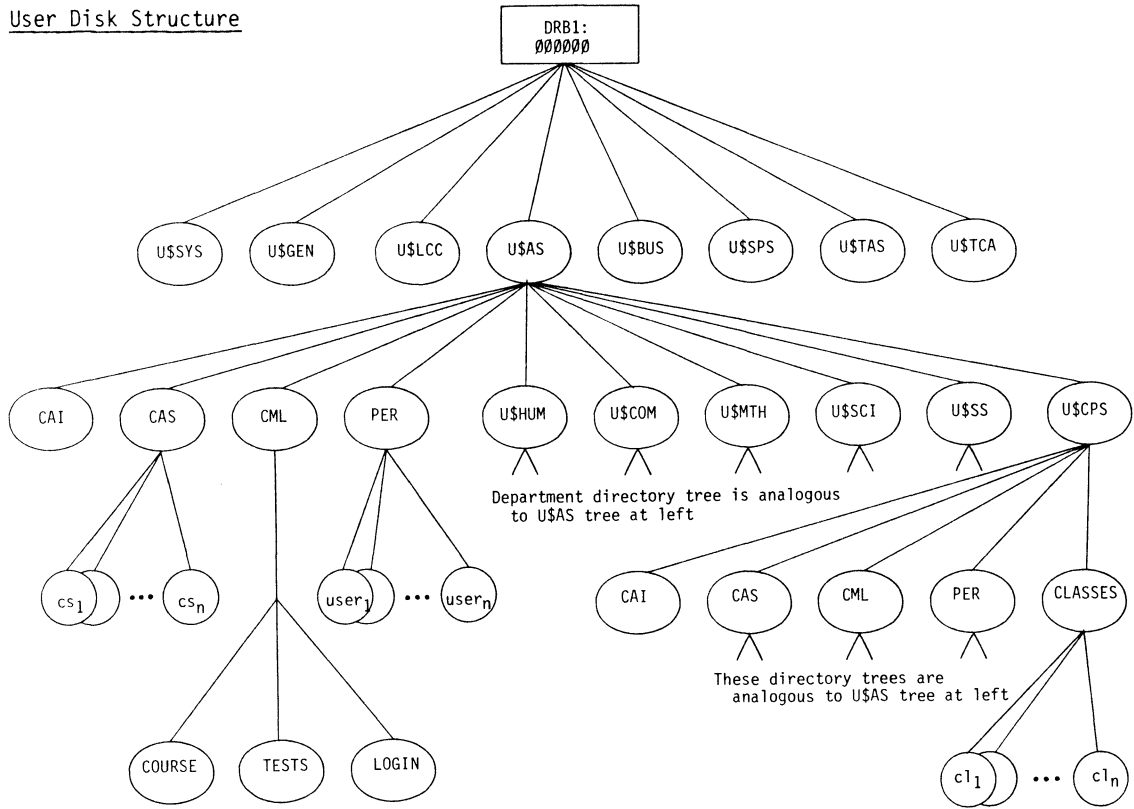


FIGURE 1

System Disk Library Structure

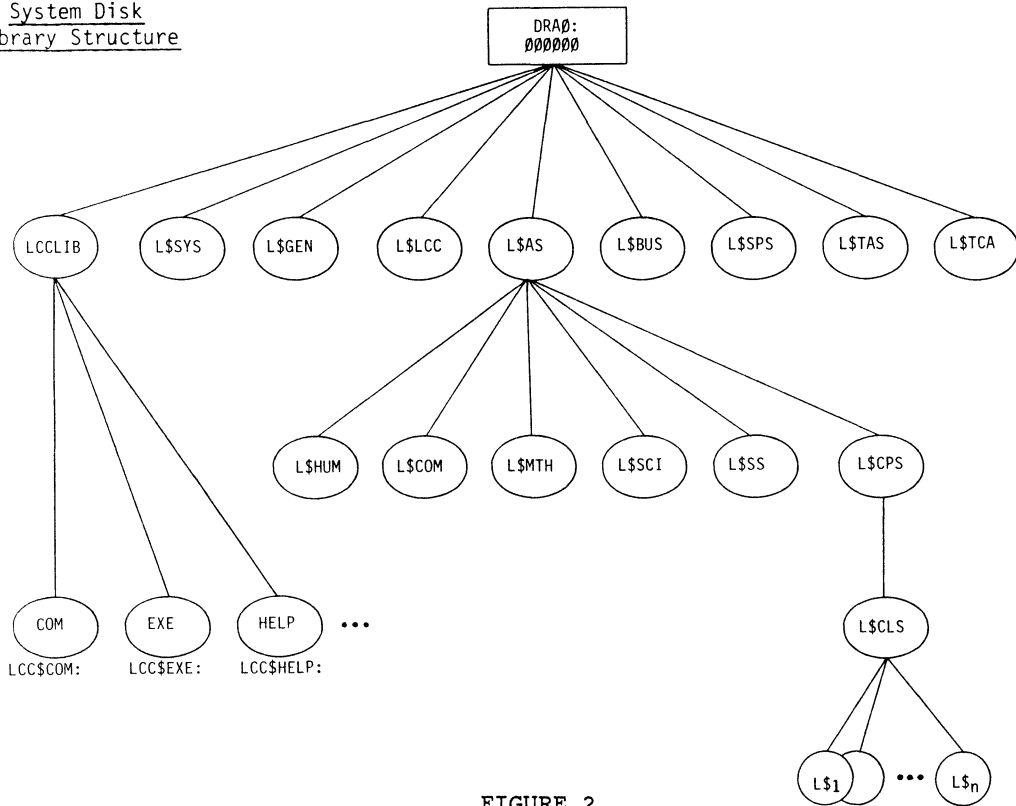


FIGURE 2

General-purpose directories are set up on the system disk called libraries to allow files to be shared between users. The divisional libraries are accessible by all users of the system. However, departmental libraries are only accessible by the individual department that owns it as well as its division.

To copy a file into your default library, use the command:

```
COPY file.ext LIB:
```

To copy a file from a subdirectory of your library into your current directory, use the command:

```
COPY LIB_[sub]file.ext *
```

The L\$env logicals are used to access libraries other than your default library.

Figure 3 illustrates the library access from a divisional account, departmental account, and CPS class accounts. The arrows between the account and the library shows the direction of access (read and write). Notice that the Communications department (COM) accounts may only read files from the divisional library. They have read or write access to their own library (L\$COM). The divisional accounts also have read or write access to any departmental library as well as their own divisional library.

The rationale for setting up the libraries in this manner is managerial in nature. Each library is maintained by key users (library managers) in each environment. Any user may write files into their own shared library, but they become the owner. Once a file has been determined as useful and permanent by the library managers, they may reassign ownership of the file to the library (via identifiers). Thus, you are not charged disk quota usage. The system charges it against the library identifiers.

Special identifiers, logical names, logical tables, and file ACL's are the key to file access for all environments at LCC. At system startup, top level environment logical tables are created for the entire divisional tree (see figure 5). At login, the appropriate logical table is linked and your job logicals are assigned automatically by the system-wide login program.

In addition to the four default logical tables (process, job, group, and system), two more tables are linked up via the LNM\$FILE_DEV logical (see figure 4): LNM\$SITE and LNM\$ENV. Instead of assigning system-wide logicals to the system table, they are assigned to LNM\$LCC which is assigned to LNM\$SITE. This keeps the system logical table and the site table uncluttered with each other's logicals.

The hierarchical file access for the libraries are dictated by the ACL's on the library directories. The following shows part of the AS library ACL's:

```
L$AS.DIR;1 4/4 L$AS (RWE,RWE,RWE,RE)
(ID=L$AS,OPTIONS=PROTECT,ACCESS=R+W+E+D+C)
(ID=L$AS,OPTIONS=DEFAULT+PROTECT,ACCESS=R+W+E+D+C)
(ID=[AS,*],ACCESS=R+W+E)
(DEFAULT_PROTECTION,S:RWED,O:RWED,G:RE,W:RE)
```

```
Directory SY$: [L$AS]
```

```
L$COM.DIR;1 4/4 L$AS (RWE,RWE,RWE,RE)
(ID=L$AS,OPTIONS=PROTECT,ACCESS=R+W+E+D+C)
(ID=[COM,*],ACCESS=R+W+E)
(ID=[AS,*],ACCESS=R+W+E)
(ID=L$AS,OPTIONS=DEFAULT+PROTECT,ACCESS=R+W+E+D+C)
(ID=[COM,*],OPTIONS=DEFAULT,ACCESS=R+E)
(ID=[AS,*],OPTIONS=DEFAULT,ACCESS=R+E)
```

```
LOGIN.COM;20 1/1 L$AS (RWED,RWED,RE,RE)
(ID=L$AS,OPTIONS=PROTECT,ACCESS=R+W+E+D+C)
```

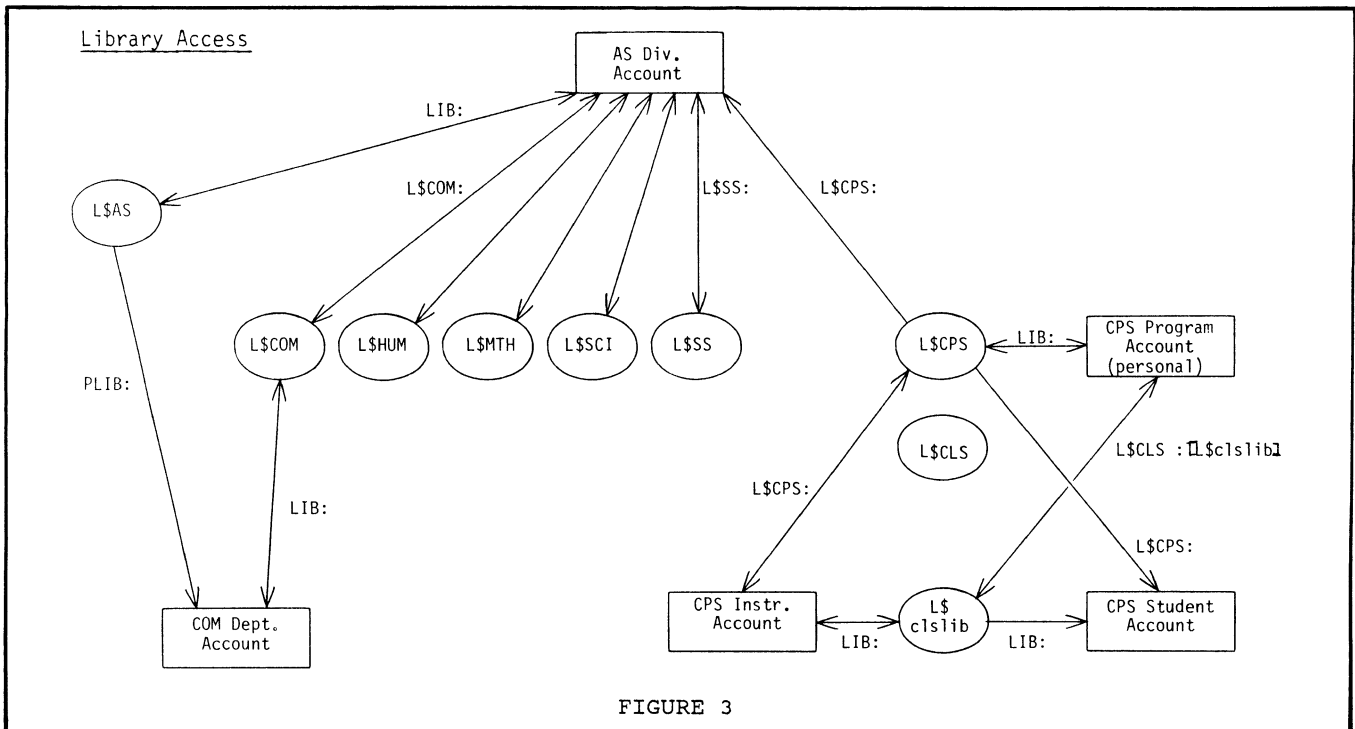


FIGURE 3

The divisions at LCC are fairly autonomous. However, sometimes it may be desirable to share or access files in divisions other than your default. Therefore, a special utility called ENVIRONMENT was created to provide you with easy access to external environments. It performs the following functions:

- 1) List all possible environment names:

```
ENV      (or ENV/LIST)
```
- 2) Set your default directory to another user's personal account home directory. For example:

```
ENV VAXUSER
```

sets your default to VAXUSER's home directory.

```
ENV/LOG AS
Default directory is U$AS_: [000000]
```

sets your default to the AS division's top root directory and logs the command.

- 3) Reassigns logicals LIB: and LIB_: in your job logical table to point to a different library. For example, to reassign your default library to be the Communications department library, use the command:

```
ENV/ASSIGN COM
```

The logicals would be assigned as follows:

```
LIB: = "L$COM:"      (Comm. library)
LIB_: = "L$COM_:"    (root of L$COM)
```

- 4) Adds or replaces a parent environment logical table to/with your current lookup table list. For example:

```
ENV/ADD BUS
```

links the Business division's logical table along with your default.

- 5) Sets environment related symbols for use in command procedures. The following command sets local symbols (for use in a command procedure) for the Communications department:

```
ENV/SYM COM
```

sets the following symbols:

```
ENV$ID=   "COM"
ENV$DIR=  "U$AS.U$COM"
LIB$DIR=  "L$AS.L$COM"
P_ENV$ID= "AS"
ENV$ALL=  "AS HUM COM MTH SCI SS CPS"
```

- 6) Shows the current settings of any of the above. For example:

```
ENV/SHOW=(PARENT,LOGICALS)
```

```
Environment COM has parent AS
```

```
Environment Logicals:
```

```
LIB: = SY$: [L$AS.L$COM]
LIB_: = DRAO: [L$AS.L$COM.]
```

FILE PROTECTION SCHEME

The default UIC protection for user files is the standard (S:RWED,O:RWED,G:REW). However, ACL's have been used to provide the special access methods described. Even the divisional logical tables are protected so that divisional managers are able to write to it while others can only read it. Until queue ACL's have been implemented, we have assigned area managers to be owner of their printer's queue.

As mentioned earlier, library managers are granted the right to manage their division's libraries. They are the only ones that have the right to assign library ownership to files placed in the libraries.

Computer class account directories have a special ACL setup that disallows students to access files from other student's directories. We implemented this by assigning ownership of their directory to the system manager. ACL's were placed on each student directory granting read, write, and execute access to the student as well as the instructor. Since students do not own their directory, they do not have "control" access that normally allows them to change their directory protection. This scheme has been very effective and secure. The directory ACL's are shown below for an LCC BASIC class (BASQ00 is the instructor):

```
BASQ00.DIR;1  3/3  [SYS,MANAGER]  (RWE,RWE,,)
              (ID=[CPS,*],ACCESS=R+W+E)
              (ID=[BASQ00],ACCESS=R+W+E)
BASQ01.DIR;1  3/3  [SYS,MANAGER]  (RWE,RWE,,)
              (ID=[BASQ01],ACCESS=R+W+E)
              (ID=[BASQ00],ACCESS=R+W+E)
BASQ02.DIR;1  3/3  [SYS,MANAGER]  (RWE,RWE,,)
              (ID=[BASQ02],ACCESS=R+W+E)
              (ID=[BASQ00],ACCESS=R+W+E)
.
.
.
BASQ41.DIR;1  3/3  [SYS,MANAGER]  (RWE,RWE,,)
              (ID=[BASQ41],ACCESS=R+W+E)
              (ID=[BASQ00],ACCESS=R+W+E)
```

SUMMARY

Creating a computing environment like the one that has been implemented at Lansing Community College has not been an easy task. It requires a system manager with VMS and programming expertise, as well as many months to design and implement all of the features. Delegating responsibility to users by granting them additional access rights relieves the system manager(s) of many duties and helps motivate the individuals.

Users have found our system to be easy to use and flexible while maintaining tight but not excessive security. Our environment promotes development, motivates the user, and allows the user to easily communicate with the system manager. Listening to users and their requests as well as implementing innovative ideas ensured the success of our site.

*

System Logical Table Directory

(LNM\$SYSTEM_DIRECTORY) [kernel] [shareable,directory]
 [Protection=(RWE,RWE,R,R)] [Owner={SYS,SYSTEM}]

Logical Tables:

[exec] "LNM\$AS"
 [exec] "LNM\$BUS"
 [exec] "LNM\$CML"
 [exec] "LNM\$GEN"
 [exec] "LNM\$LCC"
 [exec] "LNM\$SPS"
 [exec] "LNM\$SYS"
 [exec] "LNM\$TAS"
 [exec] "LNM\$TCA"

[kernel] "LNM\$SYSTEM_DIRECTORY"
 [kernel] "LNM\$SYSTEM_TABLE"

Logical Names Pointing to Logical Tables:

[super] "LNM\$DCL_LOGICAL" = "LNM\$FILE_DEV"
 [super] "LNM\$FILE_DEV" = "LNM\$PROCESS", "LNM\$JOB", "LNM\$GROUP",
 "LNM\$SITE", "LNM\$ENV", "LNM\$SYSTEM"
 [exec] "LNM\$AS" = "LNM\$GROUP_000110"
 [exec] "LNM\$COM" = "LNM\$GROUP_000112"
 [exec] "LNM\$SCI" = "LNM\$GROUP_000114"
 [exec] "LNM\$SS" = "LNM\$GROUP_000115"
 [exec] "LNM\$ENV" = "LNM\$SYS", "LNM\$AS", "LNM\$BUS", "LNM\$SPS",
 "LNM\$TAS", "LNM\$TCA", "LNM\$GEN"
 [exec] "LNM\$FILE_DEV" = "LNM\$SYSTEM", "LNM\$SITE", "LNM\$ENV"
 [exec] "LNM\$PERMANENT_MAILBOX" = "LNM\$SYSTEM", "LNM\$SITE", "LNM\$ENV"
 [exec] "LNM\$SITE" = "LNM\$LCC"
 [exec] "LNM\$SYSTEM" = "LNM\$SYSTEM_TABLE".
 [exec] "LOG\$SYSTEM" = "LNM\$SYSTEM", "LNM\$SITE", "LNM\$ENV"
 [kernel] "LNM\$DIRECTORIES" = "LNM\$PROCESS_DIRECTORY".,
 "LNM\$SYSTEM_DIRECTORY".
 [kernel] "LNM\$PERMANENT_MAILBOX" = "LNM\$SYSTEM"
 [kernel] "LNM\$TEMPORARY_MAILBOX" = "LNM\$JOB"
 [kernel] "LOG\$GROUP" = "LNM\$GROUP"
 [kernel] "LOG\$PROCESS" = "LNM\$PROCESS", "LNM\$JOB"
 [kernel] "LOG\$SYSTEM" = "LNM\$SYSTEM".
 [kernel] "TRNLOG\$GROUP_SYSTEM" = "LOG\$GROUP", "LOG\$SYSTEM"
 [kernel] "TRNLOG\$PROCESS_GROUP" = "LOG\$PROCESS", "LOG\$GROUP"
 [kernel] "TRNLOG\$PROCESS_GROUP_SYSTEM" = "LOG\$PROCESS", "LOG\$GROUP",
 "LOG\$SYSTEM"
 [kernel] "TRNLOG\$PROCESS_SYSTEM" = "LOG\$PROCESS", "LOG\$SYSTEM"

FIGURE 4

Arts and Science Division Environment Logical Table

(LNM\$AS) [exec] [shareable]
 [Protection=(RWED,RWED,R,R)] [Owner={SYS,SYSTEM}]

ACL= (IDENTIFIER=MGR\$AS,ACCESS=READ+WRITE)
 (IDENTIFIER=[AS,*],ACCESS=READ)

"L\$AS" [exec] = "SYS:[L\$AS]"
 "L\$AS_" [exec] = "DRAO:[L\$AS.]" [concealed,terminal]
 "L\$COM" [exec] = "L\$AS:[L\$COM]"
 "L\$COM_" [exec] = "DRAO:[L\$AS.L\$COM.]" [concealed,terminal]
 "L\$CLS_" [exec] = "DRAO:[L\$AS.L\$CPS.L\$CLS.]" [concealed,terminal]
 "L\$CPS" [exec] = "SYS:[L\$AS.L\$CPS]"
 "L\$CPS_" [exec] = "DRAO:[L\$AS.L\$CPS.]" [concealed,terminal]
 "L\$HUM" [exec] = "L\$AS:[L\$HUM]"
 "L\$HUM_" [exec] = "DRAO:[L\$AS.L\$HUM.]" [concealed,terminal]
 "L\$MTH" [exec] = "L\$AS:[L\$MTH]"
 "L\$MTH_" [exec] = "DRAO:[L\$AS.L\$MTH.]" [concealed,terminal]
 "L\$SCI" [exec] = "L\$AS:[L\$SCI]"
 "L\$SCI_" [exec] = "DRAO:[L\$AS.L\$SCI.]" [concealed,terminal]
 "L\$SS" [exec] = "L\$AS:[L\$SS]"
 "L\$SS_" [exec] = "DRAO:[L\$AS.L\$SS.]" [concealed,terminal]
 "LJ\$BOLD" [super] = "L\$AS:[LASER_FONTS]BOLD."
 "LJ\$COMPRESSED" [super] = "L\$AS:[LASER_FONTS]COMPRESSED."
 "LJ\$ELITE" [super] = "L\$AS:[LASER_FONTS]ELITE."
 "LJ\$ITALIC" [super] = "L\$AS:[LASER_FONTS]ITALIC."
 "LJ\$MATH" [super] = "L\$AS:[LASER_FONTS]MATH."
 "LJ\$PICA" [super] = "L\$AS:[LASER_FONTS]PICA."
 "PLIB" [exec] = "L\$AS:"
 "PLIB_" [exec] = "DRAO:[L\$AS.]" [concealed,terminal]
 "U\$AS" [exec] = "DRB1:[U\$AS.PER.]" [concealed,terminal]
 "U\$AS_" [exec] = "DRB1:[U\$AS.]" [concealed,terminal]
 "U\$COM" [exec] = "DRB1:[U\$AS.U\$COM.PER.]" [concealed,terminal]
 "U\$COM_" [exec] = "DRB1:[U\$AS.U\$COM.]" [concealed,terminal]
 "U\$CLS" [exec] = "DRB1:[U\$AS.U\$CPS.CLASSES.]" [concealed,terminal]
 "U\$CLS_" [exec] = "DRB1:[U\$AS.U\$CPS.CLASSES.]" [concealed,terminal]
 "U\$CPS" [exec] = "DRB1:[U\$AS.U\$CPS.PER.]" [concealed,terminal]
 "U\$CPS_" [exec] = "DRB1:[U\$AS.U\$CPS.]" [concealed,terminal]
 "U\$HUM" [exec] = "DRB1:[U\$AS.U\$HUM.PER.]" [concealed,terminal]
 "U\$HUM_" [exec] = "DRB1:[U\$AS.U\$HUM.]" [concealed,terminal]
 "U\$MTH" [exec] = "DRB1:[U\$AS.U\$MTH.PER.]" [concealed,terminal]
 "U\$MTH_" [exec] = "DRB1:[U\$AS.U\$MTH.]" [concealed,terminal]
 "U\$SCI" [exec] = "DRB1:[U\$AS.U\$SCI.PER.]" [concealed,terminal]
 "U\$SCI_" [exec] = "DRB1:[U\$AS.U\$SCI.]" [concealed,terminal]
 "U\$SS" [exec] = "DRB1:[U\$AS.U\$SS.PER.]" [concealed,terminal]
 "U\$SS_" [exec] = "DRB1:[U\$AS.U\$SS.]" [concealed,terminal]

FIGURE 5

Using VMS to Teach Operating Systems

Jerry Scott, Ph.D.
Spring Hill College
Mobile, Alabama 36608

This talk discusses teaching the standard ACM-oriented, undergraduate course in "Fundamentals of Operating Systems" using VMS and its many features as the standard example of an Operating System. Classroom laboratory exercises are designed to highlight the principles of Operating Systems Design. Some of the VMS features used include: DCL, the SHOW utility, Timers, Event Flags, and Mailboxes, the MONITOR utility, Process Scheduling and Memory Management Techniques, and Working Set Tuning Techniques.

1. Operating Systems course perspective

- o Student preparation
 - * Two Pascal Courses
 - * One Cobol Course
 - * One course in VAX macro
 - * One course in Data Structures
- o Problems in teaching OS -- students cannot modify VMS
- o Need a small OS that students can "Tweak" to see how OS performance can be changed

2. Classical Operating Systems

Problems covered in OS course

- o Higher language to machine translation capability
- o Concurrency -- PC versions vs. True VAX versions
 - * Readers and Writers
 - * Producers and Consumers
 - * Dining Philosophers
- o Process Deadlock Causes and Cures
- o Memory Management
- o Files Structures
- o Performance Measuring or Benchmarking

3. Current Text used -- Deitel.

Developed 100 page set of lecture notes to complement Deitel.

4. Most OS books either follow OS360 or Multics.

5. Most textbooks advocate "State of the Art" techniques as of late 70's

6. Why use VMS?

- o VMS supports many "State of the Art" features:
 - * DCL -- easy to learn vs. JCL
 - * Process scheduling is sophisticated and not batch oriented
 - * Balance Set Implementation
 - * Many utilities easy to access
 - * Excellent Monitor for investigating dynamic data structures VMS uses

7. Initial Classroom Exercises with VMS

- o Building an additional Utility Monitor all processes within your Group.
- o Using Monitor to discover Process ID's
- o Show System to see System Data Structures

- 8. Advanced Exercises with VMS -- VMS solutions to the classical OS problems.
 - o VMS language facilities --
 - * common Debugger
 - * common .obj format
 - * common linker
 - o Explanation of Systems Services Layer
 - o VAX calling standard
- 9. Solutions to concurrency and process coordination with VMS
 - o Using timers to activate suspended processes
 - o Using Event Flags and MailBoxes for Interprocess Communication
- 10. Performance Measurement Techniques
 - o Investigating the Balance Set and Working Set mechanisms used by VMS
 - o Using the System Monitor to size various working sets for VMS processes
- o Alan Watson's DECUS talk on "Turbo Tuning" using Working Set Parameters
- o Lab exercise with Working set parameters
 - * determine different working set sizes
 - * How to tune the WS parameters for our workload?
- 11. Future directions for the OS course with VMS
 - o Built a simple "partition oriented operating system" on IBM PC in Turbo Pascal
 - * Have students run this system and vary such things as
 - a) I/O timeout amounts
 - b) First Fit vs. Best Fit memory management
 - c) Quantum variation
- 12. Incorporate these items into VMS with above, lessening the number of written labs and increasing the number of programming labs



Student Information Systems

Warren Alkire
Abilene Christian University

ABSTRACT

Student Information Systems are described. Factors leading to the choice of a Student Information System are listed, as well as ideas on a successful implementation plan for a Student Information System.

To begin, some background on Abilene Christian University and its computing history should be given so the reader can compare their situation with our situation to see if there are similarities. ACU is a private, 4-year university of approximately 4500 students offering both baccalaureate degrees and graduate degrees up to the master's level. Around 1968 the university acquired an IBM 360 model 30. No EDP upgrades were made until the late 1970's when a PDP 11/70 was installed. On this several home grown systems were developed including registration, loan collection, billing, etc. At this point the school came to an impasse as the IBM was still doing 80% of the processing and there was no more machine capacity to develop interactive systems on the 11/70 to replace the IBM programs. There was also no physical room for another machine. The decision was made to do a quick convert to allow the 11/70 to run the IBM programs in batch mode temporarily and to buy two VAXes and "canned" software to replace the systems running on the IBM and 11/70. A VAX 11/780 and 11/750 were installed (taking up less room than the 360) and the Information Associates Student Information System (SIS) was purchased. ACU went live with the SIS system June 1, 1984.

What are the functions of a student information system? This may seem like a ridiculous question as the users attend their conferences, read their journals and are continuously telling the Data Processing department of all the functions they "need". There are some things that ACU felt were essential functions of administrative support software. First, the student recruiting office needs to track prospective students. This is one of the places where students first enter the database. Student recruiting will need to be able

to pull groups of prospective students for mass mailings, and should have the capability to do enrollment projections. The second office needing computer support is the Admissions office. This is the other place where students will enter the database. This office needs to do application processing, decision support and generate reports based on various attributes of the applicants.

The most obvious function of a student information system is the Registrar functions. This office will need computer support to enter students into their classes, produce transcripts, do class and room scheduling and enrollment reporting. Based on historical data, the registrar's office will want to be able to project total enrollment as well as class and room usage. One of the improvements that computers can offer over traditional methods is in the area of transcripts. With disk storage of a student's class history, transcripts can now be organized by subject or category rather than chronologically. This actually presents the information in a more useful format to both prospective employers and other institutions of higher learning.

Probably some of the most complex processing occurs in the Financial Aid office and therefore there is a great need for computer support. Some of the capabilities needed are support for need analysis tapes, document tracking, and decision support of aid packaging. Computer support of aid packaging allows the financial aid office to be more equitable in the distribution of funds. They should be able to do packaging models to see how their funds last and what adjustments in the packaging formula will lead to the

need of more students being met. This is in stark contrast to the traditional method of awarding packages to students in order of their need until the funds are depleted.

The business or bursar's office functions are something that could be included in a discussion of student information systems though they could equally well fit in the category of accounting software. Nevertheless, this office does have information about the student's account and will require computer support for student billing and possibly student loan collection depending on the set-up of the particular institution.

A final function that is particularly important to private schools such as ACU is the ability to feed information about students to the Alumni office as these students leave the institution. ACU's alumni are an important source of funds for the school so accurate information on student's and their parents is vital. Information about siblings is also useful for future student recruiting.

There are several factors to consider in the decision to buy versus in house development. The first factor is the time tradeoffs for in house development. The institution must consider the amount of time it will take to develop the needed capabilities as well as the personnel costs for capable analysts and programmers. Against this time consideration must be balanced the time and resources that will be needed for training in the use of a purchased package and conversion from whatever system is currently in place.

Another factor in the decision to buy is that someone else has already thought through many of the problems. Much of the groundwork and debate has already been done. Also, because a vendor must deal with multiple clients there are often problems solved in the software that your institution has not yet encountered. This is a two-edged sword. This is a help in that when the problem or a desire for these additional capabilities appears in the future, the software is already able to handle it. It also may lead to the consideration of problems which do affect the institution but have not yet been considered. This factor may be a hindrance in that the solutions to problems not affecting the institution and the unneeded capabilities may make the operation of the software more difficult than it needs to be for your particular institution.

One characteristic of software vendors is that they do not have to be concerned with the day to day operation of running a school. This allows them to concentrate on improving the tool rather than "fighting the fires" that occur when using the tool for administrative support. This can be detrimental if the vendor does not have a good perception of the true needs of an institution and solves nonexistent problems while leaving other pressing problems unresolved.

The best advice that can be shared from ACU's experience concerns who makes the decision on which software package to buy. That should be a user community decision. Administrative support is no longer a DP driven environment. Now the users tell DP when the various jobs will be run and DP is a servant to the other departments. An important aspect of this principle is that the decision should be made both by user VPs as well as clerk level users. ACU did not consult the clerk level users and this caused some problems during the installation of the software. The reason for letting the users make the decision about which package to buy is so they will be happy with the functions provided by the vendor chosen and will require a bare minimum of modifications to the package. The reasons for keeping modifications to a minimum will be discussed later.

While it is vital that the user community make the decision on which software package to buy, Data Processing needs to have some input into the decision. There are several technical considerations for DP to take into account. First, what mechanisms are available for conversion? What resources will be required to do the conversion from the present system to the purchased package? The system ACU purchased provided batch transactions that could be loaded with the information from the old systems. By using batch transactions, the interrelationships between the files are handled. This allowed ACU to go ahead with the conversion before the DP staff had learned the relationships and nuances of the system. Another thing to be considered is the ability of the current DP staff to support the new package. Will other manpower or machine resources need to be added? Also the DP department must consider the modifications to be made and the time it will take to have these modifications in place. Will the installation of the software be able to be complete before these modifications are complete? Finally, the data processing staff must look at the issue of integrating homegrown systems whose functions will not be replaced by the new software with the purchased package. ACU has a required daily chapel attendance Programs had been developed to track the attendance and these programs had to be interfaced with the bought package which provided the more traditional administrative support functions.

When the decision to purchase a package has been made, there is another decision to be made. What if any modifications will be made to the purchased package? Commercial packages by nature have to be generic, yet by this very nature they may not fill all the needs of a particular institution. This is why modification becomes an issue for almost every installation. In administrative support software, the specifications will change. The government seems to delight in changing the way financial aid administrators do business on a least an annual basis. Other things will change. SAT may change the layout and content of their magnetic tape records as they did this year. New needs and demands will arise for the software to fulfill. There are two roads which may be taken. One road is to consider the purchase a one time deal and do any required modifications in house. The other road is to chose a vendor who provides an update service, usually in the form of release maintenance. Extensive modifications to the delivered system will inhibit release installations. If the modifications are too extensive, they may prevent timely implementation of the releases. Therefore it is best to keep the modifications to an absolute minimum. The one time purchase strategy has the advantage of freeing an institution from the problems of release maintenance installation, but carries its own pitfalls. The institution will have to make specification changes itself. This can demand considerable resources in man hours which is not getting any cheaper. Most modifications will require someone with extensive experience in the workings of your software package. This makes high staff turnover rates very costly. Consider the example of the resources necessary for making specification changes in financial aid. As a bare minimum which may not be entirely adequate, and institution will need a super financial aid administrator who is responsible in all areas of the financial aid office. This person must be able to read and correctly interpret the government regulations. Second a programmer/analyst will be needed who is extremely well versed in financial aid administration almost to the point that the person could work effectively in the financial aid office at a moment's notice. Even with these two individuals, the institution does not have the viewpoints of other financial aid administrators in interpreting the government regulations. These other viewpoints from other situations can lead to a more correct understanding of the government's meaning in a particular regulation.

The final thing to be considered is the Student Information System Integration Committee. Most of the current packages are highly integrated. They make the business office dependent on the work of the registrar's office who in turn is relying on data from the Admissions office and so on. It is important that all components of an integrated system be brought up simultaneously. Those schools that are using the same package as ACU, who are having problems and have contacted ACU have generally gone live on some but not all the components of their student information system. This is usually a factor in the problems they are experiencing. ACU is becoming increasingly aware of the way that the actions of one office impacts the work of other offices. These interrelations need to be discussed and policies need to be made in light of those discussions. There also needs to be a reason for the integration committee to meet regularly. At ACU that reason is final enrollment which happens at the beginning of each term. This occurs four times a year. When the committee meets it not only discusses what if any changes need to be made in the way final enrollment is done and who will actually do the multitude of tasks that must be done each time, but it also discusses any problems that have arisen since the last meeting that concern integration of the software. Any potential changes in policy by one office are discussed here so that the other offices that will be affected can speak up before the policy change causes any problems.

A summary of the findings of Abilene Christian University in the installation of a student information system would include the following considerations. Let the user community make the decision about which software package will be purchased. Data processing should have some input. Keep modifications to the base system to a minimum. The fewer modifications, the easier it will be to keep the system current with regulation and specification changes. Form an ongoing integration committee that has some reason that causes them to meet on a regular basis. These factors will ease some of the problems in installing and using a student information system.

About the author: Warren Alkire is a programmer/analyst with Abilene Christian University. He has been with the university since July 1982 and was one of two programmers on the Student Information System implementation committee. He spent a year as the primary contact for the SIS system.



GRAPHICS APPLICATIONS SIG

**COMPUTER ASSISTED COURSE DEVELOPMENT
AND
INSTRUCTIONAL SYSTEM (CACDIS)**

A.L.Lakshminarasimhan
Associate Professor
Computer Science Dept.
Stevens Institute of Technology
Hoboken, N.J., 07030

ABSTRACT

This paper describes the design, development and implementation of a 'Computer Assisted Course Development and Instructional System' (CACDIS) on Digital PC 350. This system can be used by instructors to create Computer Aided Instructional (CAI) materials for a variety of courses. Students can access the CAI materials through this system for self study. Also the instructors can use the system as an aid in class room teaching. This system also provides an easy interface with graphics to create pictures. These pictures can be combined with textual information. The use and the capabilities of the system are illustrated with an example of a graphics course. The proposed system would enhance human-machine interaction both as instructional aid to students and also for the instructors for the generation of the course material.

INTRODUCTION:

In a major study on computers and their use in the educational environment, a Carnegie Commission noted that there are two problems of computers as instructional tools. Firstly, there is a lack of good learning materials in many areas. How do we generate such learning materials? Secondly, the non-availability of learning materials, to students on a large scale basis. With the advent of personal computers there has been a remarkable growth in their use for instruction and education for solving the above problems and as a consequence we are no longer limited by the availability of resources of a large time-sharing system. The use of microcomputers is a powerful medium for instruction. Thus the hardware needs of a typical computer-based tutorial or instruction is not excessive and is well within reach of students.

Historically, Computer Aided Instruction (CAI) has been used by industries to train their personnel on specific aspects of a job. In CAI the student trains himself by repeatedly going through a sequence of instructional material. Persons with varied learning abilities could progress at their own pace as they receive the feedback from the CAI. This process is difficult and almost impossible if an instructor were required to pay individual attention to each one of the students. Now CAI is being increasingly used in classroom education at schools and universities. The use of visual aids is certainly an asset, especially where there is a need to portray graphics information. Needless to say, even simple animation can be a very effective means of understanding complex ideas in Science and Engineering. The use of CAI with animation

makes the process of instruction very effective. However, the creation of the course material for CAI is rather tedious and time consuming. This paper describes a Computer Assisted Course Development and Instructional System (CACDIS). It is simple to use and suitable for course material development and to impart instruction.

The CACDIS system provides the user with two functions - the creation of a new course and accessing of an existing one. Creation of a new course involves three subfunctions; create the necessary menu using Frame Development Tool (FDT), create textual information and create graphical information. The menu frame enables easy interaction for students to traverse through different parts of the course material. This is created by using the FDT available on Digital PC 350. The instructor interactively creates pages of tutorial material by properly assigning chapter numbers, section numbers and page numbers. The system interactively reads this information and stores it as a record (with concatenation of chapter number, section number and page number as an index to that record). The facility includes provisions for easy modification and deletion of pages. One could also insert a page between two existing pages. During the creation of course material, the instructor can traverse through the course material so far developed by using 'NEXTSCREEN' and 'PREVSCREEN' keys. Also, the course material could be in any arbitrary order. The graphics generator of CACDIS enables creation of graphical

(pictorial) information. The instructor can use the core graphics language primitives interactively to draw pictures.

Students can access the relevant course material through CACDIS as follows. First the CACDIS displays the information regarding

the chapters of the course selected. The student/user can then make his selection. The system then displays the first page in a chosen chapter/section and the student can traverse through the course forwards or backwards by using NEXTSCREEN and PREVSCREEN keys. This feature of traversing through course material is also present during creation and modification of course material.

The CACDIS provides 'help' feature. Both instructors and students can query for operational details while they are either creating course material or accessing it.

The main features of the CACDIS system are the following:

- 1) Students constantly interact with the course material while learning.
- 2) A great deal of individualization of the course material is possible to suit the needs of the students.
- 3) An aid to students in problem solving skills through computer aided instruction.

PROBLEM DEFINITION

We are concerned with the design and development of a CACDIS system having the following capabilities.

a) An efficient tool for instructors to create course material consisting of textual information and graphical information.

b) A facility for comfortable learning of course material.

THE CACDIS SYSTEM

The CACDIS system is intended for two types of users. The 'instructor-user' - the person who intends to use the system and generates a computer based course material. The 'student-user' - who uses the system and learns a particular course.

The system design depicting various modules and their relationships is illustrated using the hierarchical diagram of fig. 1. The CACDIS is divided into two modules:

- 1) Module to create a new course.
- 2) Module to access an existing course.

The creation module is divided into

three sub modules:

- 1) Course structure
- 2) Menu and help frames through FDT
- 3) Textual/graphical information

The course material is input in terms of pages. The system is menu-driven with the 'help' feature at each stage of development of course material or retrieval of course material.

FILE SYSTEM

The CACDIS system requires creation, modification and access of pages containing CAI material. The file structure chosen should be capable of accessing the pages randomly as well as sequentially. Hence we have chosen index sequential access method provided by RMS on the Digital PC 350. In order to accommodate the facility of inserting pages between existing pages, the primary key of index sequential file is chosen as chapter number, section number, page number and additional page number.

PROGRAMMING DETAILS

All the modules, illustrated in Fig 1, are coded in PASCAL. In what follows, we describe the scenario of operation of the system.

The 'Instructor-user' creates the course by invoking the CACDIS system. A main menu, as shown in fig. 2, will be displayed on the screen. He may choose to create a new course by positioning cursor at 'Creating a new course' and pressing Do. Then the menu, as shown in fig. 3, will be displayed. He would create course structure which is planned earlier for computerising the instruction.

A. Creation of Textual Information

Textual information is stored in blocks of 1280 characters called 'pages'. Once we select option 'Create textual information' from the menu (see fig. 3), then a new menu as shown in fig. 4, pops up. This has four options which are described below:

(i) Create a new page

The system prompts for the chapter number, section number, and page number. If the page does not exist, then the system displays a blank page. The user may enter the textual information. The process is terminated by pressing the 'Do key'. However if the page is already created then a prompt to that effect is given to the user. An example for creating page 1 of chapter 1 section 1 is shown in fig. 5.

(ii) Page Modification

Once the user selects this option, the system prompts for the chapter, section and the page number. If the page exists then it is displayed as shown in fig. 6. If the user intends to modify this page, he could respond 'yes'. System then positions the cursor at the beginning of the page for modification. If the user responds with a 'n' then the modification is not done.

(iii) Page Insertion

An additional facility of inserting a page before an existing one can be done in two ways:

1) Two pages in the same chapter and section, with numbers say 1 and 2 are created by the user. If the user later needs to add another page between these two pages, then he can respond with a 'y'.

If the user's response is 'n', then the page traversing menu is displayed and the user may traverse the database. After each page is displayed the system enquires if the user intends to insert a page before the current one.

2) Suppose there are two pages existing in chapter three, section four-(pages five and ten with additional page number of zero). The page being currently displayed is page ten. The user prompts the system that he intends to insert a page before the current one. The system then allows the user to create a new page with identification of chapter three, section three, page number nine and additional page number zero.

(iv) Page Deletion

Existing pages in the data base may be deleted using this facility. The system prompts the user for the chapter number, section number and page number. The data base is checked for the existence of this page and it is displayed (see fig. 8). The system issues prompt to confirm deletion. The user may then delete this page by responding 'y'. If however, the user decides not to delete the page, the page traversing menu is displayed. For each page that is displayed the system prompts the user to see if he intends to delete the current page.

B. Creation of Graphical Information

This module provides the user with a facility for generation of picture and text for illustration. A large number of instructions chosen from core graphics language is implemented.

As in the case of text creation, the system prompts the user for the chapter, section number and page number. The system checks for the existence of the page and then prompts the user to that effect if the page does exist. If it does not exist then the screen is cleared and core graphics primitives are displayed. Additional core graphics language instruction can be seen by depressing 'NEXT SCREEN'. Further, the user can traverse back by pressing 'PREVSCREEN key' for earlier core graphics language instruction. The user enters the number corresponding to the instruction that is needed and the system prompts with the corresponding inputs needed. After the user gives inputs for the instruction, an effect of execution of instruction is shown interactively on the screen (see fig. 9)

C. Accessing the Course

Now the CACDIS system is ready for accessing course material. The student-user can go through series of chapter, section numbers and pages to learn about a specific topic (see fig.10)

CONCLUSION

CACDIS is a general purpose system with dual capability of creating course material for a specific subject and its use as Computer-Assisted instructional system for students. The CACDIS system, implemented on Digital PC 350, is powerful enough to be used for the development of computer based instructional system. It is simple and is completely menu driven. The users of CACDIS can learn the operation of system easily with the 'help' feature provided at each stage of operation. It is important that the present day computer aided instructional packages provide an easy interface for creation of pictures. The course material created for students may not only contain textual information but also pictorial drawings. Accordingly, CACDIS is provided with an easy interface to graphics. This does not need elaborate programming usually required and hence it is easy for user to create graphical images. Such a CAI material on CACDIS would significantly enhance understanding of course.

REFERENCES:

1. A.L.Lakshminarasimhan "Computer Assisted Course Development and Instructional System on Digital PC 350", Computer Science Dept. Report, Stevens Institute of Technology, 1986.

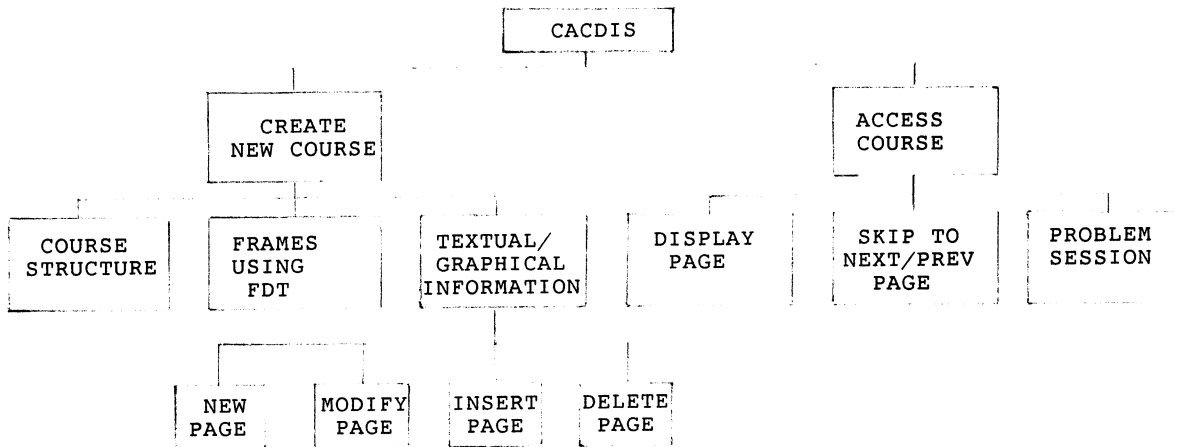


Fig 1: Hierarchical diagram of CACDIS system

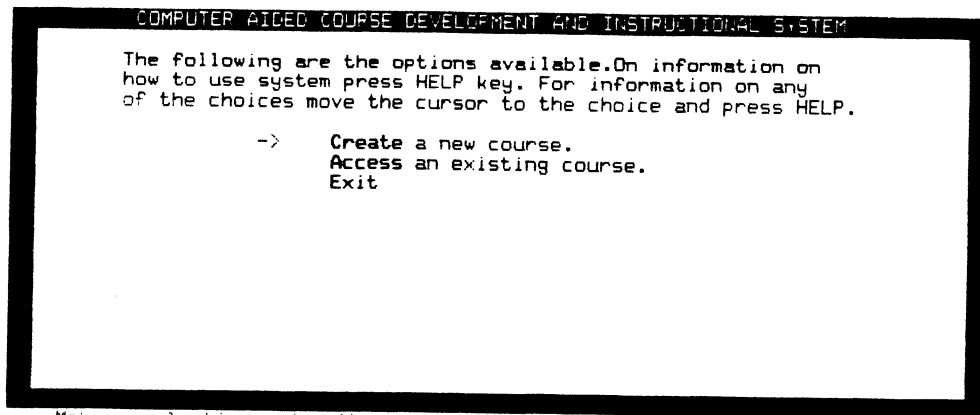


Fig 2

Make a selection using the arrow keys and press DO. Press EXIT to leave

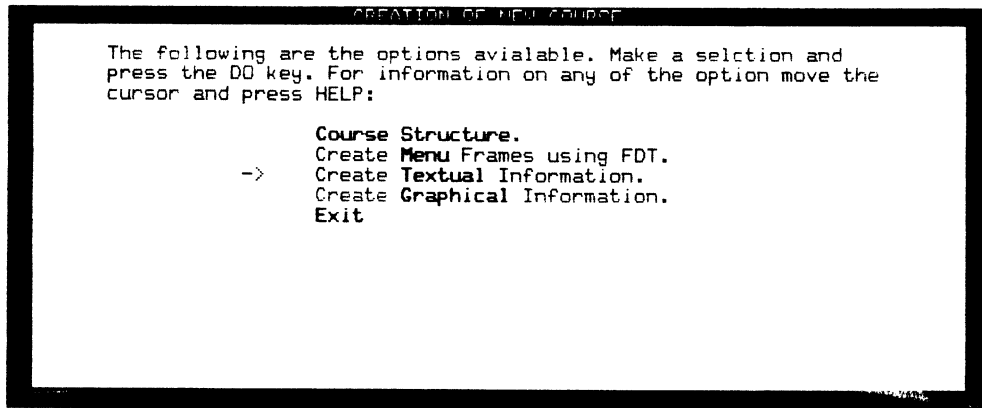


Fig 3 Press HELP for information. Press EXIT to leave the session.

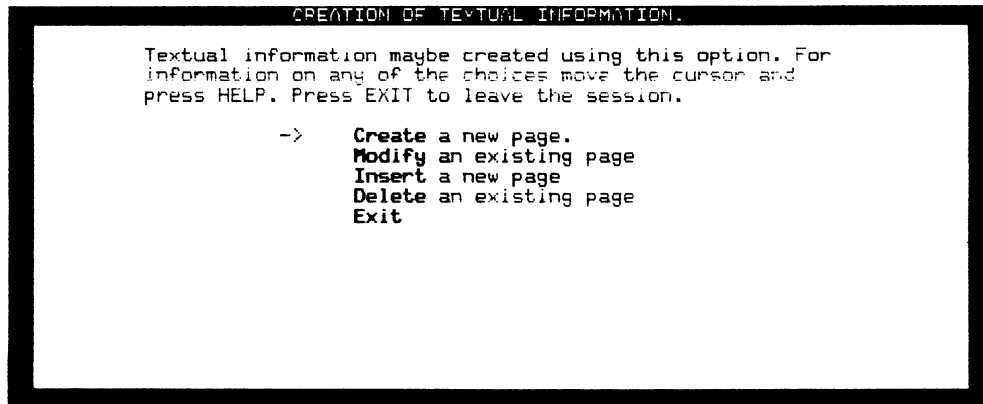


Fig 4 Make a selection using the arrow keys and press DO :

```

CHAPTER : 1      SECTION : 1      PAGE : 1      ADDL. PAGE NUMBER :
  
```

WELCOME TO THE COURSE
ON
COMPUTER GRAPHICS

This course introduces you to the basic concepts in computer graphics. It teaches you the basic skills that you need to start writing programs for computer graphics.

Select a topic and press do:

- LIST OF TOPICS
1. Introduction
 2. Computer Basics
 3. Basic interactive graphics programming
 4. Implementation of Basic graphic packages
 5. Geometrical Transformations

Contd.

Fig 5 Press NEXT SCREEN for next page Press PREV SCREEN for previous page Press HELP if needed Press EXIT to leave session

```

CHAPTER : 1      SECTION : 1      PAGE : 2      ADDL. PAGE NUMBER : 0
  
```

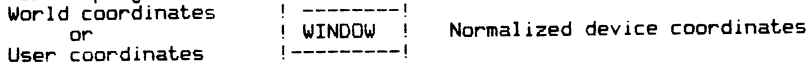
6. Viewing in three dimensions
7. Raster algorithms and software
8. Hidden surface elimination
9. Shading algorithms
10. Color Models.

Contd.

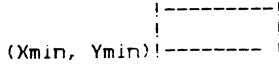
Fig 6 MODIFY THIS PAGE ? [Y/N/HELP]: Press HELP if needed

WINDOWS

Window statement in Core graphics language [C.G.L.] or Graphic Kernel System [G.K.S] is used to map device independent world coordinates to normalized device coordinates. Window selects the portion of world coordinate space for display.



The C.G.L or G.K.S. statement is WINDOW (Xmin, Ymin, Xmax, Ymax) where the coordinates are given by (Xmax, Ymax)



Contd.

Fig 7 INSERT BEFORE THIS PAGE ? [Y/N/HELP]: Press HELP if needed

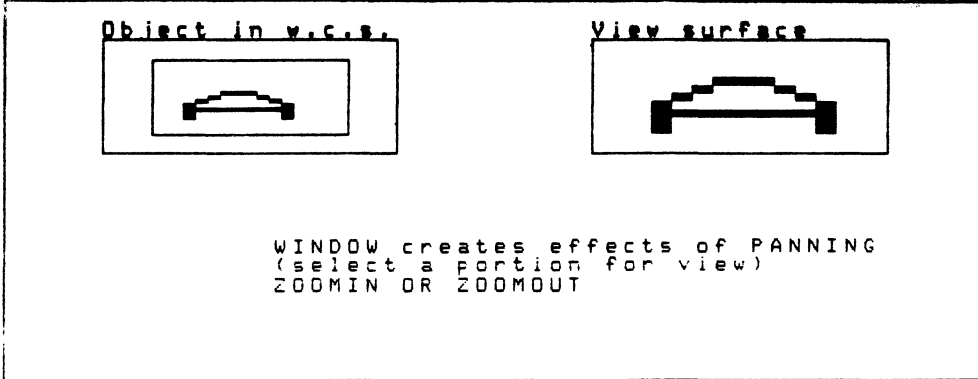
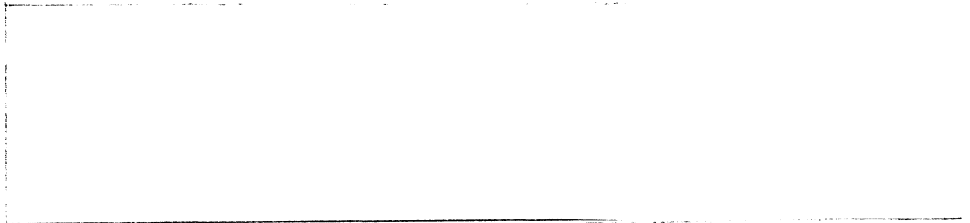


Fig 8 DELETE THIS PAGE ? [Y/N/HELP]: Press HELP if needed



TYPE THE NUMBER OF THE INSTRUCTION YOU WANT

(0)SET WINDOW (1)MOVE ABS (2)MOVE REL (3)LINE ABS (4)LINE REL
(5)SET VIEWPORT (6)RECTANGLE ABS (7)RECTANGLE REL (8)MARKER ABS
(9)MARKER REL (10)SET LINEWIDTH (11)SET CHARSIZE (12)SET CHARSPACE

Press EXIT when finished Press NEXTSCREEN PREVSCREEN to see more options

Fig 9

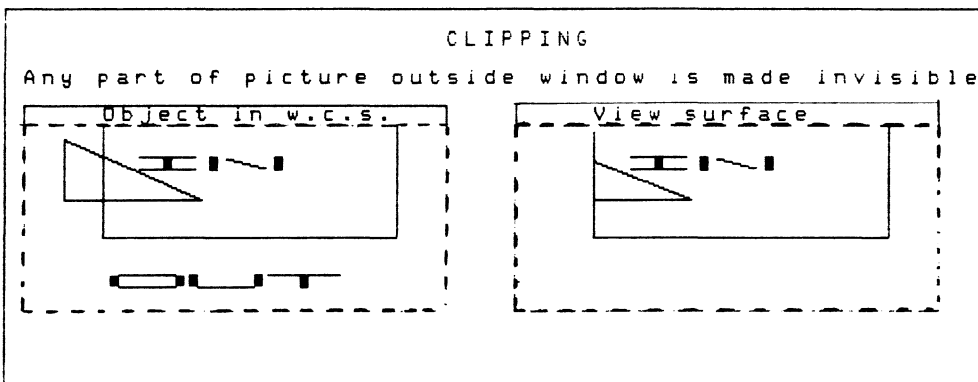


Fig 10 Press NEXT SCREEN for next page Press PREV SCREEN for previous page Press HELP if needed Press EXIT to leave session

INTERTASK COMMUNICATION

Ted Smith
Division of Medical Physics
Department of Radiation Therapy
Hospital of the University of Pennsylvania
Philadelphia, PA 19104

Memory Management

Introduction

Intertask communication is a powerful tool available to project designers and programmers. Large projects can be divided into a group of smaller and less complex tasks. Each task provides a subset of the functions required from the project. Through various methods, these tasks communicate information for processing and synchronization. Greater flexibility for future enhancements is achieved by updating only affected tasks or adding a new task. The task environment created by the computer hardware and operating system provide the mechanisms which enable, control and limit communication among tasks. The Radiation Therapy Patient Tracking System will illustrate some of the communication mechanisms provided on our system.

Hardware environment

Digital's PDP-11 family of minicomputers is among the most popular in use today [1]. However, the 16-bit word size of the PDP-11 restricts a task from explicitly referencing a memory address greater than 64Kb [1]. Using the memory management option, a task's logical 64Kb of address space [2] may be mapped to any physical memory address [2]. Memory management is used to divide a task into a maximum of eight segments. Each segment is independently mapped and assigned either Read-only (R/O) or Read-write (R/W) access for the task. Segments may even map to the same physical address space as other tasks if permitted by the operating system.

On the PDP-11, memory management is virtually transparent to the programmer who references task address space as logically contiguous and as starting at physical address zero (0). The Memory Management Unit (MMU) converts the task's 64Kb of logical address space into the actual physical addresses. The MMU hardware consists of eight 32-bit registers known as Addressing Page Registers (APR). Each APR can map from 64 bytes to 8192 bytes called a page. The MMU hardware divides physical memory into blocks of 64 bytes [3] and a page always begins on a 64 byte boundary. APR's are composed of two 16-bit registers called the Page Address Register (PAR) and the Page Descriptor Register (PDR). The PAR contains the base (starting) address of the page being mapped by the APR. The PDR contains the size of the page in blocks and the access rights to the page granted by the operating system (Figure 1).

Access to logical address space 50000 to 57777 is restricted because the page size is only 2kb in the PDR of APR 2. Conversion of a logical to physical address is accomplished by a mapping process. The following steps and examples demonstrate this process:

Figure 1: Example of Logical to Physical address mapping

Logical Address	APR	PAR	PDR	Physical Address
0 - 17777	0	1600	200	160000 - 177777
20000 - 37777	1	2000	200	200000 - 217777
40000 - 47777	2	2200	100	220000 - 227777
60000 - 77777	3	4000	200	400000 - 417777
100000 - 117777	4	4200	200	420000 - 437777
120000 - 137777	5	3300	200	330000 - 347777
140000 - 157777	6	13000	200	1300000 - 1317777
160000 - 177777	7	13200	200	1320000 - 1337777

Memory Mapping Process:

1. Bits 13-15 of the logical address determine which APR map the corresponding physical address.
2. The PAR contains the physical base address of the page mapped by the
3. Bits 6-12 are added to bits 0-6 of the PAR. The address of the physical memory block is the result.
4. Bits 0-5 of the logical address contain the block address offset resulting in the physical address. These bits are appended (on the right), to the result of step 3. The final result is the actual physical address.

Operating Environment

The Interactive Application System (IAS), Digital's largest operating system for the PDP-11, provides an ideal environment for intertask communication. The IAS executive contains a variety of mechanisms used for communication including address sharing, event flags and a communication node pool [4].

Address sharing between tasks is accomplished by having two or more tasks map the same physical address. This provides the programmer with the fastest method of transferring information between tasks. The IAS executive controls whether a task may map to another task's address space by assigning a protection mask to each segment of every task. Once access is granted, IAS loads the access rights (R/O or R/W) and page size into the PDR and the base address in the PAR of each APR that maps to the shared region of memory. The shared region then becomes a part of the address space of each task.

Examples of mapping process using data from Figure 1:

I. Logical address accessed = 023712 = 0010011111001010

```

                APR = 1 = 001
Bits 0-12 remain = 0011111001010          Step 1

                PAR1 = 002000 = 0000010000000000
Add bits 6-12:   0011111
Memory block address = 0000010000011111    Step 3

Append bits 0-5: 0000010000011111001010    Step 4

Physical address = 00203712 = 0000010000011111001010

```

II. Logical address accessed = 143076 = 1100011000111110

```

                APR = 6 = 110
Bits 0-12 remain = 0011000111110          Step 1

                PAR6 = 013000 = 0001011000000000
Add bits 6-12:   0011000
Memory block address = 0001011000011000    Step 3

Append bits 0-5: 0001011000011000111110    Step 4

Physical address = 01303076 = 0001011000011000111110

```

Event Flags

IAS provides event flags as a mechanism for tasks to detect and declare events (changes) in the operating environment. Sixty-four event flags are available in two groups of thirty-two flags: Local and Global. Each task has a unique set of local event flags numbered 1 thru 32 which cannot be accessed by other tasks in the system. Changing the state (setting or clearing) of local event flags will only affect the task and has no effect on other tasks in the system. Local event flags are commonly used to inform the task that an I/O request {2} has been completed. Common to all tasks in the system are a set of global event flags numbered 33 thru 64. Global event flags can be accessed by any task having proper privilege. Using a global event flag enables a task to inform one or more other tasks of an event. The programmer decides which event flag to associate with a specific event, so that a task can sense up to sixty-four different events. The IAS executive reserves local event flags 25 thru 32 of every task and global event flags 57 thru 64 for use by the operating system [5].

Node Pool

IAS allows up to 24Kb of main memory to be used as a system communications area called SCOM. System data structures such as the Send/Receive Queues and the Active Task List are located in SCOM [6]. Any remaining unused space in SCOM is made available to tasks as a node pool. Each node is eight words in length. A task may have up to 255 nodes in use at any time. Nodes are charged to the sending task until released when the receiving task accepts the message contained in the nodes.

Methods of Intertask Communication

Using one or a combination of communication mechanisms, IAS provides the programmer with five methods for intertask communication. Most of these communication methods are also available with other operating systems including RSX and VMS. In addition, IAS includes a group of FORTRAN callable subroutines allowing access to executive facilities directly from a FORTRAN program.

Methods of Intertask Communication:

- o Disk files
- o Subtasking
- o Chaining
- o SEND/RECEIVE messages
- o Shared Global Area

Disk Files

Disk files allow tasks to transfer large amounts of data. This is the simplest method of communication requiring none of the communication mechanisms previously described. Because of the disk access overhead, data files are beneficial only for large data transfers, memory constrained systems or batch oriented projects. A typical batch project would consist of two or more tasks that run sequentially. The first task creates the input data file later used by another task. Our department billing system is a good example of a batch system.

Our department billing system is composed of three phases: Charge item entry, Daily tape creation and reporting functions (Figure 2). During the day, as patients receive services provided by our department, the charges are entered into a file containing all unbilled charges. At the end of each day, the operator mounts and creates the "daily billing tape". The daily tape contains the unbilled charges sorted and grouped by various fields such as the patient's medical record number (ID) and status (inpatient/outpatient). The tape is then sent to central data processing where the actual patient billing occurs. Also, the charges written to tape are appended to a log file prior to the unbilled file being deleted. Finally, daily billing reports are prepared and placed into the administrator's account by a nightly batch job. Quarterly and other reports are possible using the charge log maintained by the billing system.

Subtasking

Subtasking is the ability of a task (parent) to run (spawn) another task (child). When a child is spawned, the parent task is suspended from further processing until the child completes. When spawning a child task, IAS allows one line of information (command line) to be passed from the parent to the child. This command line usually contains parameters informing the child of the operations to be completed for the parent. Subtasking enables a task to use system utilities such as SORT-11 and PIP, drastically reducing the coding of projects.

FIGURE 2: DEPARTMENT BILLING SYSTEM

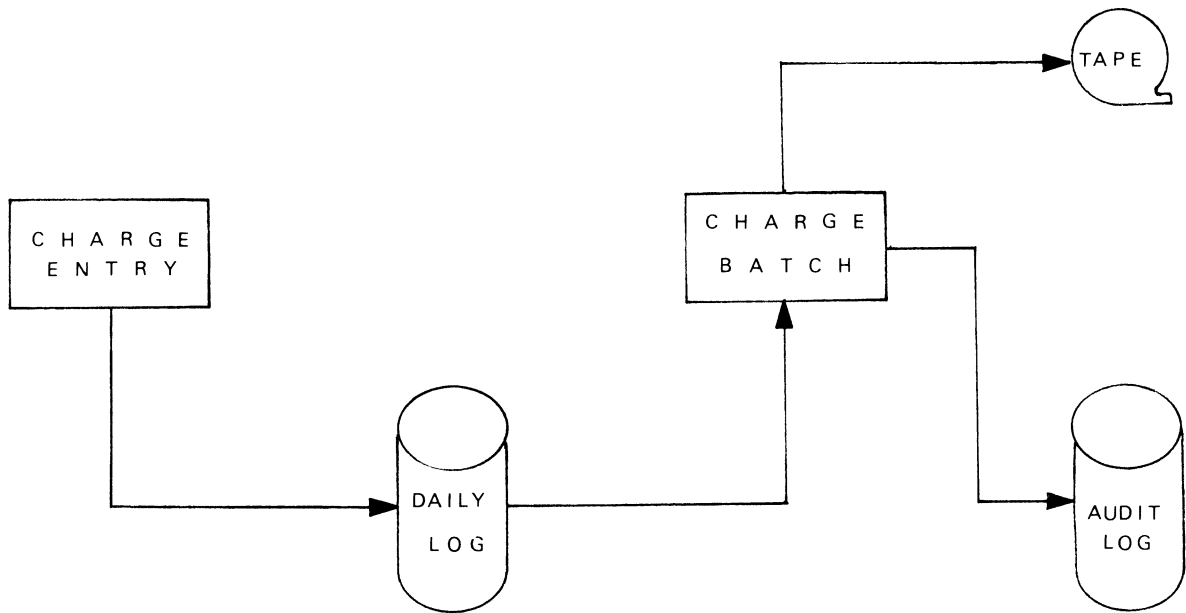
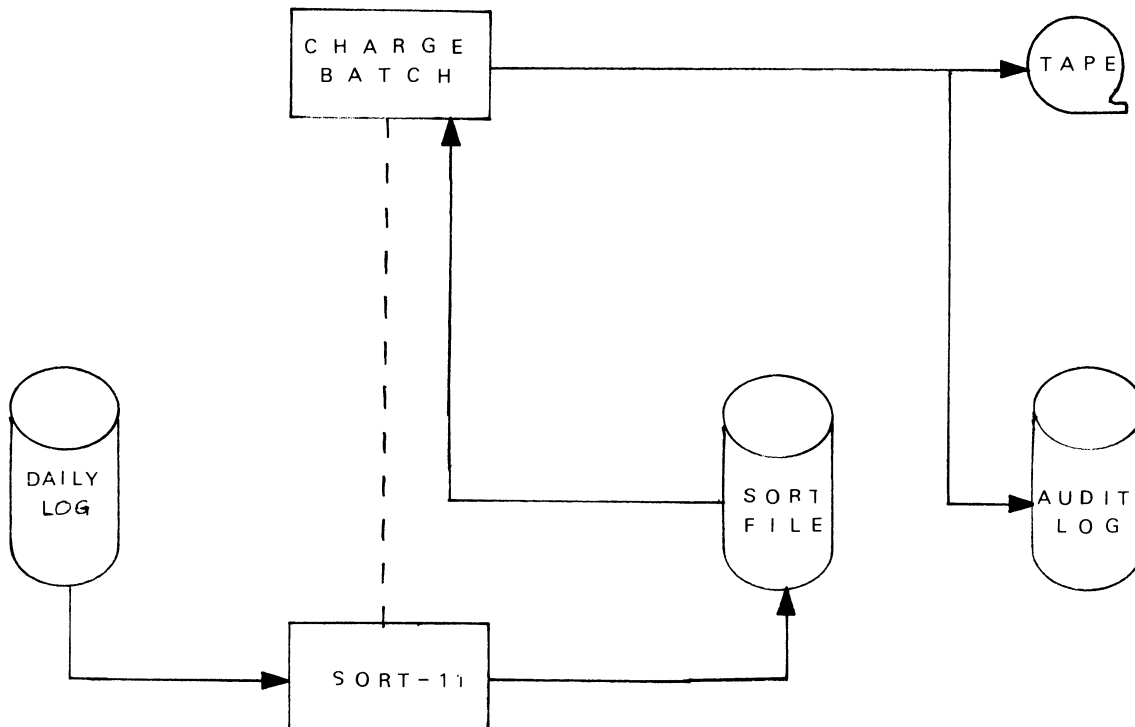


FIGURE 3 SUBTASKING SYSTEM UTILITIES



Again, the department billing system provides an excellent example of subtasking. During the daily billing tape generation phase, charges must be separated into batches of outpatients and inpatients. Each batch must then be sorted by medical record number. SORT-11 provides the capability of both selecting and sorting specific records from a file. Thus, to sort outpatients, we spawn SORT-11 and pass the name of the unbilled file, a sort specification file and the output file name. The sort specification file contains all the information for record selection, sort keys, and output file format. When completed, the sorted output is written to tape. PIP is then spawned to delete the sorted output file (Figure 3). This process is then repeated for inpatients. Here, subtasking eliminated the need to write and maintain a sorting algorithm reducing the task to a much smaller program.

Chaining

Chaining allows another task to be invoked when the current task completes. As with subtasking, a command line may be passed to the succeeding task. However, chaining differs from subtasking in that the initiating task exits as the successor is scheduled to run. Under IAS, chaining is available only in timesharing mode using the Timesharing Control Services (TCS) macro CHN\$T [7]. Tasks running in real time or multi-tasking modes can simulate chaining by using the RUN\$ [8] system directive before exiting. The RUN\$ directive does not permit passing a command line to the task being scheduled.

Send/Receive Messages

IAS provides the Send/Receive message utility allowing messages to be passed between tasks. Nodes from SCOM are used to buffer the Send/Receive messages. A message may contain up to 510 bytes of information. Tasks waiting to receive a message may request to be suspended, stopped or exit if no messages are received. Likewise, tasks sending a message may request that the receiver be resumed from a suspended or stopped state. This mechanism allows tasks to efficiently synchronize communications. An optional event flag can be set by a sending task as an alternate means of informing the receiver that a message was sent.

Send/Receive messages are most useful for brief messages rather than to transfer large amounts of information requiring multiple messages. Nodes comprising the message are charged against the sender's quota until released when the message is received. Sufficient nodes must be available or the message will not be sent. Although the messages are stored in memory, it will be copied several times to different locations before being received. First the message is assembled into a buffer in the sender's task address space. When the SEND directive is issued, the contents of the message buffer are copied into the node pool by the executive. The message is then copied from the node pool into the message buffer of the receiver. Finally, the receiver interprets the message.

Shared Global Areas

Shared global areas (SGA) are regions of memory shared by two or more tasks. SGA's may contain either data or instruction code. When used to contain data, an SGA provides the fastest mechanism for data transfers among tasks. Reduction in memory use is obtained by placing common routines into an SGA. Four types of SGA's are supported by IAS:

- o Resident Libraries
- o Common Areas
- o Installed Regions
- o Dynamic Regions

An SGA is built and installed in a manner similar to a normal task (Figure 4a). SGA's are taskbuilt with no taskheader, stack or units. A symbol table must be specified when taskbuilding. An SGA must be installed prior to installing any task that reference the SGA (Figure 4b).

Figure 4a: A FORTRAN example of a Shared Global Area

```
BLOCK DATA

INTEGER*2 NODES           ! Register: size of pool in nodes
INTEGER*2 VACANT          ! Register: address of first FREE node
INTEGER*2 POSTS           ! Register: address of first POST node
INTEGER*2 CASES           ! Register: address of first CASE node
INTEGER*2 TRKEFN          ! Register: Global Event flag
INTEGER*2 WAITRM          ! Register: Tracking console
INTEGER*2 TRKFLG(10)      ! Registers reserved for future use
INTEGER*2 POOL(16,511)    ! TRKCOM node pool

COMMON /TRKCOM/ NODES, VACANT, POSTS, CASES, TRKEFN, TRKFLG, WAITRM
COMMON /TRKCOM/ POOL

DATA NODES/511/
DATA TRKEFN/33/

END
```

Figure 4b: IAS DCL taskbuild command for a Shared Global Area

```
LINK/POSITION/SYMBOL/NOHEADER/OPTIONS TRKCOM
UIC=[11,120]
STACK=0
UNITS=0
/
```

Figure 4c: IAS DCL Mapping to a Shared Global Area

```
LINK/FULL_SEARCH/OVERLAY_DESCRIPTION:TRKHUP/READ_WRITE-
/MAP/TASK:TRKHUP/OPTIONS
ACTFIL=10
MAXBUF=80
UNITS=40
UIC=[11,120]
RESSGA=LB:[11,120]TRKCOM/RW
/
```

Mapping to an SGA requires at least one APR from the task's set of eight. An SGA size that is not an exact multiple of 8Kb will result in the task losing the ability to access addresses between the size of the SGA and the next higher 8Kb multiple address. This is a hardware restriction of the Memory Management Unit. Mapping an SGA larger than 8Kb can be accomplished by two methods. First, a task may allocate sufficient APR's until the entire SGA is mapped. This reduces the address space of the task available for other purposes by 8Kb for each APR allocated to the SGA (Figure 4c).

For some applications, mapping an entire SGA is not practical. Alternatively, a Task could logically divide the SGA into windows. Using the memory management directives, only one APR would be needed to access the SGA. However, only addresses mapped by the current window will be available to the task. The task would have to unmap the current window, freeing the APR, and map to another window to gain access to other addresses in the SGA.

Resident Libraries

Resident libraries allow routines common to many tasks to be placed into a region shared by those tasks. This reduces the actual physical memory needed to simultaneously run the tasks. Resident libraries are always read-only and never swapped out of memory. SYSRES and HANLIB are resident libraries in IAS containing common system routines used by many tasks.

Common Areas and Installed Regions

Unlike resident libraries, common areas and installed regions may be swapped out of memory and be mapped read-write by tasks. Common areas are swapped to the disk image file, thus the area is preserved in the disk image file until loaded back into memory. Installing the

common area will load the contents of the area when it was last swapped to the image file. Installed regions are swapped to the system swapfile leaving the disk image file intact. Installing an installed region will always load the initial contents of the region when taskbuilt.

Dynamic Regions

Similar to installed regions but created at run-time by the task, "dynamic regions" provide a task with the ability to send large amounts of data to another task. While other types of SGA's must have a unique name, a dynamic region provides the programmer with three naming mechanisms increasing the protection of the region.

- o The dynamic region may be nameless. To gain access, the region creator must use the send by reference directive to attach the receiving task to the region. This provides the creating task with full control of who may attach to the region.
- o A global name may be assigned to the region. Access to the region is possible by any task knowing the name of the region.
- o A terminal sensitive name, allowing only tasks running at the same terminal as the region creator to attach.

These mechanisms are in addition to the protection mask assigned to the region when created. Therefore, a task knowing the global name of a dynamic region can still be denied access if it fails to match the protection requirements of the region protection mask.

An Illustration

Our department's "Patient Tracking System" illustrates some uses for intertask communication. The patient tracking system is used to follow patients through the department and collect information about the services provided to them. At each service area, a VT220 terminal is installed to interact with the system. When the patient arrives at the reception desk, the receptionist notifies the tracking system. For each service the patient is scheduled, his name is placed on the screen of the VT220 at the corresponding service area. Selecting a patient for a service area locks out selection by other areas for that patient. When the service is completed, the patient is then removed from the service area and becomes available to the other areas. The tracking system is currently composed of an installed region and three tasks used to control the service area terminals, lookup patient information and utilities (Figure 5).

Installed Region

The installed region, TRKCOM, contains information for all of the service areas and patients in the system. TRKCOM is a 16Kb region and is mapped using two APRs. The region is divided into 16 registers and 511 nodes of 16 words. Like the system node pool (SCOM), the nodes in TRKCOM are used to communicate the current state of the tracking system among the tasks composing the system. These nodes are dynamically allocated and deallocated to form one of five record types:

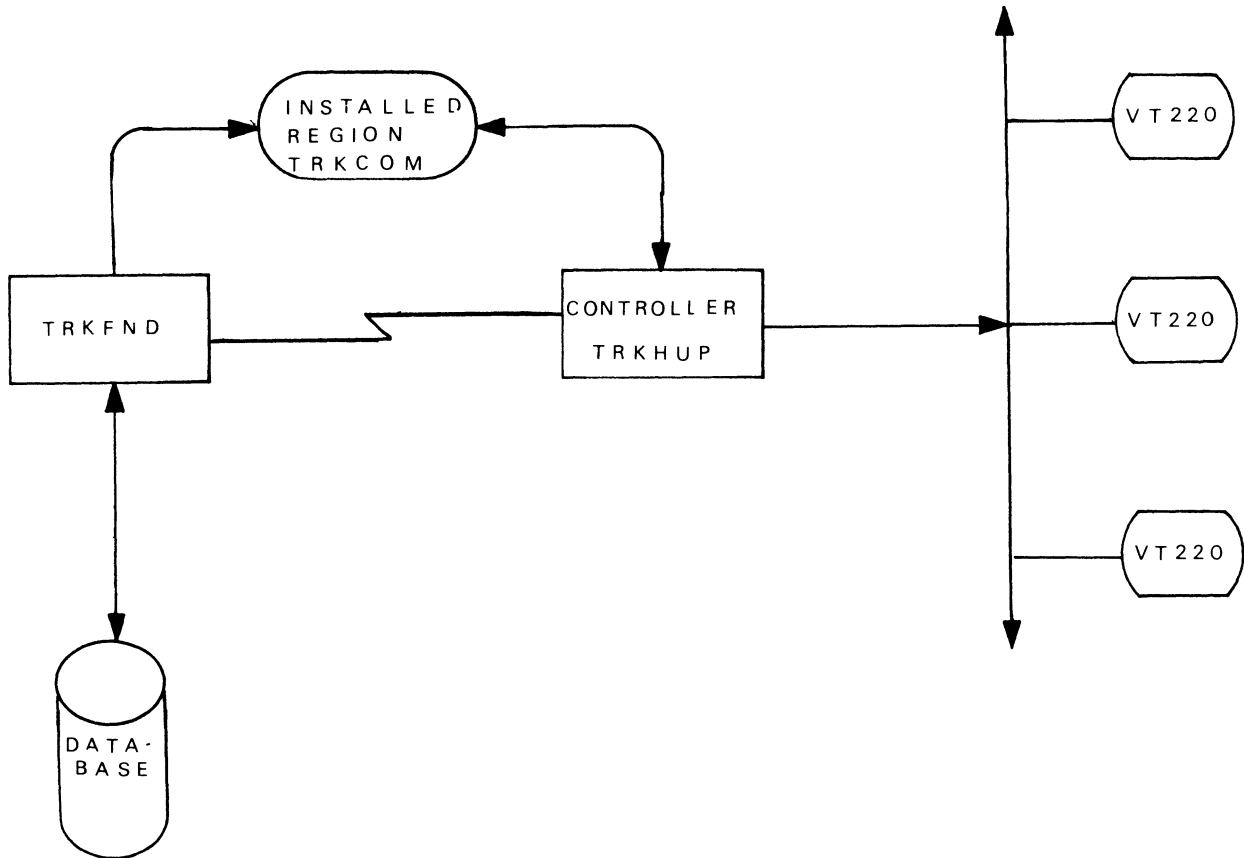
- o POST record: is assigned to each service area containing area specific information. POST records contain one node.
- o CASE record: contain demographic information (name, ID, etc.) for each patient currently in the tracking system. CASE records contain three nodes.
- o TASK record: contain information about the service scheduled for the patient. One task record is created for each service the patient is receiving. TASK records use one node.
- o RUMOR record: are used to transmit messages between service areas and use a variable number of nodes depending on the length of the message.
- o FREE record: mark unallocated nodes in the area and have a variable number of nodes assigned. Adjacent unallocated nodes are combined to form one FREE record.

Currently, only six of TRKCOM's sixteen registers are used by the tracking system. These registers contain:

- o Size of tracking system node pool (in nodes)
- o Address of the first FREE record
- o Address of the last POST record
- o Address of the last CASE record
- o Global Event Flag used by tracking system
- o Address of the reception service area

When installed, TRKCOM contains the size of the node pool, global event flag number and one FREE record of 511 nodes. All other registers are zero. The tracking controller task, TRKHUP, maintains the data and integrity of the region.

FIGURE 5: PATIENT TRACKING SYSTEM



Tracking Controller

TRKHUP, the controlling tracking system task performs three functions: initialization of the TRKCOM region, control of all service area terminals and requests for patient schedule information from the database interface. When TRKHUP is executed, the TRKCOM region is loaded with all service area information from a startup data file. The service terminals are cleared and painted with the main menu. The controller then assigns a local event flag and issues a QIO\$ [9] for each terminal. TRKHUP then awaits commands from the service areas using a "wait for logical OR of event flags" directive.

TRKHUP divides the service areas into reception and service posts. At the reception post, patients are entered into the system as they arrive in the department. The patient's appointments for the day are displayed at each post scheduled. Until the patient is selected by a post, he is displayed at the reception desk as waiting. A post may select a patient from the list of available patients displayed. Once selected, a message is sent to the post where the patient is waiting (usually the reception area), to send the patient to this post. Also, the patient is marked unavailable to any other post scheduled for the patient. When the service is completed, the technician or nurse removes the patient from the post, making the patient available for selection by another post. If this was the patient's last service for the day, the technician is informed and the patient is sent home. Otherwise, the technician will be prompted for the postname where the patient will be sent to wait for his next service. A message utility is also supported by TRKHUP, allowing posts to send messages to one another. This popular feature allows a post to send special instructions or requests to another post.

Patient Lookup

A patient database [10] is stored on disk and may require many disk accesses to locate a patient. To maintain a high response time, a separate task was written to search for patient information freeing TRKHUP from ever referencing a disk file after startup. SEND/RECEIVE messages and a global event flag are used to initiate and communicate with the database interface task, TRKFND.

When a patient arrives in the department, the receptionist informs the system of a new arrival. TRKHUP will then prompt the receptionist for a patient identifier. When either the name, social security number or ID of the patient is received, TRKHUP sends a message to TRKFND, and sets global event flag 33. TRKFND is a task which issues a wait for event flag directive and is suspended until event flag 33 is set. When TRKHUP sets flag 33, TRKFND is resumed and immediately clears flag 33 and fetches the message from the SEND/RECEIVE queue. The patient identifier is extracted and TRKFND searches the patient database for a matching patient file. If an exact match is found, TRKFND sends a message to TRKHUP containing today's schedule information for the patient. However, if an ambiguous identifier (such as a common surname) was received, TRKFND sends TRKHUP a message containing matching patient names and summaries. TRKHUP will then display the matching patients to the receptionist and prompt for a selection. If the receptionist selects a patient from the display, TRKHUP sends TRKFND the chosen identifier. If no patient was selected, TRKFND is sent a message asking for more matches. This process will continue until the patient is found in the database or no more matches remain. After TRKFND sends TRKHUP a response to a message, a wait for event flag is issued suspending TRKFND until the next patient arrives.

Tracking Utilities

A utility task, TRKUTL, was written as a programming aid for testing and debugging the tracking system. Options include:

- o Initializing the TRKCOM region
- o Dumping the TRKCOM region
- o Testing the patient search task
- o Stop the tracking system tasks

Normally, the utility task is only used to test new features of the tracking system. However, dumping the TRKCOM

region and testing TRKFND can be performed while the system is active without risk of corruption. Statistics of node pool usage can be kept for periods throughout the day enabling detection of periods of node depletion.

Future Enhancements

Additional features are now being designed into the tracking system. Some of the more interesting are described below with a brief explanation of how the features will be implemented:

- o Catching service information: When a patient is provided a with service, the technician enters the service information and TRKHUP sends the information to a task which updates the patient database.
- o Automated billing: Based on the treatment information entered, the database updater looks up the billing charge and sends a message to the billing system (previously described) to generate a charge record for the patient.
- o System viewing: A task attaching read-only to the TRKCOM region allowing a non-tracking user such as the chief technician to view the patient load at any service area and to send messages to the areas.

Summary

Using the communication mechanisms provided by IAS has improved the design of the Patient Tracking System. User response time was increased by dedicating a task (TRKFND) for database searches, freeing the controller (TRKHUP) from disk bound operations. Features such as realtime monitoring of activity and communicating with other projects, (billing system) increases project functionality. In addition, we are better able to meet the expanding needs of our department.

Footnotes

- {1} "Kb" a constant equivalent to 1,024 bytes of memory.
- {2} Typically, an I/O request is a read or write to a device such as a terminal, disk file or lineprinter.

References

1. Digital Equipment Corporation, PDP-11 Architecture Handbook, 1983, Preface p vii.
2. Digital Equipment Corporation, IAS Systems Directives Reference Manual, October 1978, pp 2-2, 2-3.
3. Digital Equipment Corporation, IAS Executive Facilities Reference Manual, October 1978, p 1-5.
4. Digital Equipment Corporation, IAS Performance and Tuning Guide, December 1980, p 2-6.
5. Digital Equipment Corporation, IAS Executive Facilities Reference Manual, October 1978, p 2-2.
6. Digital Equipment Corporation, IAS Executive Facilities Reference Manual, October 1978, pp 3-5 to 3-7.
7. Digital Equipment Corporation, IAS Guide to Writing Command Language Interpreters, December 1980. pp 8-8, 8-9
8. Digital Equipment Corporation, IAS Systems Directives Reference Manual, December 1980, pp 4-103 to 4-106.
9. Digital Equipment Corporation, IAS Systems Directives Reference Manual, December 1980, pp 4-84 to 4-88.
10. Curley, Robert F. and Smith, Theodore J., "A Radiotherapy Department Computer Database", Proceedings of the Eighth International Conference on the Use of Computers in Radiation Therapy, July 9-12, 1984, pp. 501-505.

LARGE SYSTEMS SIG

TOPS-20 Technical Update

Donald A Kassebaum
Computation Center
The University of Texas at Austin

Abstract

This session featured presentations from Digital on the status of TOPS-20 software.

Speakers

- Ernie Racine TOPS-20
- Mark Pratt DECmail/MS
- Dave Eklund FORTRAN

TOPS-20

Current Status

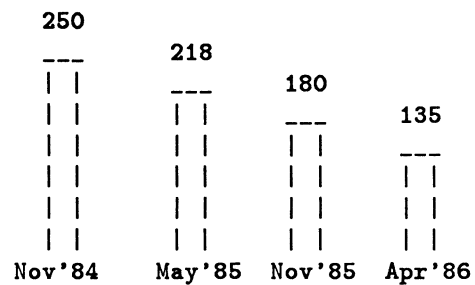
TOPS-20 Rel 6.1

- Field Image (KL)
 - Current "Off-the-Shelf" Version
 - Basis for Maintenance
 - (Support for 5.1 Ends September 1986)
- Reliability Is High
 - Long Uptimes
- Performance Is Improving
 - Enhancements Are On The Way
- Maintenance Efforts
 - Autopatch
 - SPRs

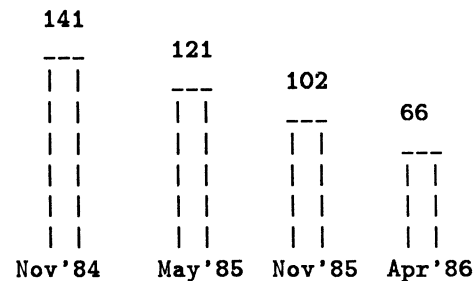
Autopatch Status

- Tape 12 (SDC Ship - April 1986)
 - First Release To Update TOPS-20 6.1
- Tape 13 (Expected SDC Ship - June 1986)
 - OFN Caching (performance enhancement)
 - Additional Edits

TOPS-20 SPR Backlog



TOPS-20 Customers Waiting



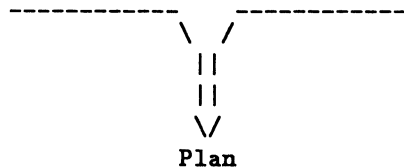
TOPS-20 - Future Directions

Release 7.0

- Goals
 - Complete Commitments
 - Improve RAMP
 - Minimize External Change
 - Minimize Impact of Upgrade
- Sources of Input
 - DECUS
 - Engineering
 - SPRs/QARs
 - Marketing

TOPS-20 Rel 7.0

Goals Priorities Ideas



• Expectations For

- Content
- Schedule

TOPS-20 DECmail/MS version 11 MS - MX

TOPS-20 DECmail/MS v11

benefits

- BUNDLED WITH TOPS-20
- PROVIDES SUPPORT TO -10's, -20's, VAXes
- MS - USER INTERFACE
- MX - MESSAGE SENDERS/RECEIVERS
 - REPLACES OLD UNSUPPORTED MAILERS
 - RELIABILITY
 - COOPERATES WITH ARPA MAILERS

Current status

- IN FIELD TEST ON TOPS-10 AND TOPS-20
- EXPECTED TO GO TO SDC BY END OF JULY
- DISTRIBUTED ON UPDATE TAPE

FORTRAN-20

Overview of V10.0 Features

FORTRAN-10/20 Version 10

- Field Image
 - Shipped in June, 1985
 - TOPS-10 and TOPS-20
 - KL, KS
- Autopatched Beginning With Tape 11
 - EXE File Replacement Option

FORTRAN-20 Version 10 New Features

- Full Language FORTRAN-77
 - Full level validation for All Configurations
- Extended Addressing
 - TOPS-20 KL Model B Only
- Portability Flagger
- Industry Standard Magtape
- G-Floating Support
 - TOPS-10 and TOPS-20
- VAX FORTRAN Features
- Debugger Enhancements

Portability Flagger

- Options
 - /FLAG:ANSI
 - /FLAG:VMS
 - /FLAG:ALL
- Compile Time Warnings
- Run Time Warnings

Supported Configurations

- TOPS-10
 - 7.02 or 7.03
 - KL or KS
 - LINK 5.1 or 6.0
- TOPS-20
 - 4.1, 5.1, or 6.1
 - KL or KS
 - LINK 6.0

Documentation

- Language Manual
- Pocket Guide
- FORTRAN-10/20 and VAX-11 FORTRAN Compatibility Manual
- Math Library Manual
- Installation Guide

Installation

- Leave FOROT7.EXE on System Indefinitely
- Install V10 Compiler, FORLIB, FOROTS, FORDDT
- Programs linked with V10 FORLIB use FORO10.EXE
- Old .EXE's Continue to Use FOROT7.EXE

Future Directions

Performance Improvements

- Object Code Improvements for Character Data
 - Single-character Assignments
 - Single-character Relationals

RMS

- Sequential, Relative, and ISAM File Organizations
- Invoked by New I/O Keywords
- I/O Keyword Compatibility with VMS FORTRAN
- Access to COBOL and BASIC+2 RMS files
- No Additional Overhead when RMS Not Explicitly Invoked

Network File Access

- Transparent Network File Access Between TOPS-20 and VMS for ASCII Data
- Remote I/O to RMS and Non-RMS Stream Files

VMS Symbols

- VMS-compatible Symbol Names Up to 31 Characters in Length
- Dollar Sign (\$) and Underscore (.) in Symbol Names

TOPS-10 Technical Update

Frank J. Francois
Federal Home Loan Bank Board
Washington, DC

Abstract

This article is a transcript of an audio tape made of this session at the Dallas symposium. Representatives from Digital Equipment Corporation were on hand to discuss the technical changes to the TOPS-10 operating system. Mark Pratt talked about DECmail MS. Dave Eckland discussed Fortran-10/20. Kimo Yap talked about TOPS-10 V7.03 performance. Bill Davenport discussed TOPS-10 futures. A question and answer period followed.

DECmail MS

The DECmail MS product is a mail system which has been available in various forms for a while now. It's been on TOPS-20 for quite a number of years, and it's been available on TOPS-10 for a few years as a product that was unbundled. We've done an awful lot of work on it. We now have a version that runs on both TOPS-10 and TOPS-20. It's a two-part mail system. There is a user interface and there is a system interface which actually sends and receives the mail. The mail system will let you send mail, it will let you read mail, it will let you move messages out to other files.

[MS is] a pretty powerful interface; you can do quite a bit of things with it. You can get a little summary listing of every message that has a particular subject or a particular date. There's probably a dozen to two dozen different options you can get for selecting mail messages.

As the TOPS statement says there, [MS/MX is] going to be bundled with TOPS-10 7.03. It's not on your SDC tapes because we're still in field test with the product, but it is going to be delivered with the operating system and be [present on] future update tapes.

It runs on both TOPS-10 and TOPS-20 and talks between the two of those. It also talks to VMS systems. It uses two different protocols to do this. Between TOPS-10 to TOPS-20 or TOPS-10 to TOPS-10 or TOPS-20 to TOPS-20, we use something called the Simple Mail Transfer Protocol (SMTP). If you are familiar with the Arpanet at all, it is a spec that is called RFC 822. The protocol that we use to talk to VMS systems is called Mail-11, which is the standard bundled mail system on VMS. In addition to that, there is some capability to talk to the ALL-IN-1 mail system using software on the VAX that is called

MR GATE. We speak directly with that to the Mail-11 protocol.

MS is the user interface and it is very powerful. MX is the system process that is the sender/receiver. MS is written totally in Macro and MX is written in a combination of Macro and Bliss-36. We are sending out the sources. In the case of MX, if you don't have Bliss-36 then we have provided some method for you to tailor MX, so that you can tailor the number of senders for TOPS-10, TOPS-20 or VMS.

By the way, [MX] communicates over DECnet. There has been no supported network mail on TOPS-10 before. There have been several unsupported mail systems out there, one of which was called NETMA. I don't believe it was even sent out to the field.

The reliability of the [MS/MX] product is pretty high. Currently you are able to have 512 messages in your mail text file; that will probably be extended to a very large number very soon. Since I can't speak about the TOPS-10 part of the reliability in comparison to what we had on the TOPS-20, [let me just say something about TOPS-20]. Some of the very old mailers that we had on the TOPS-20 side corrupted your mailed out file very easily. That's one of the things that we put into this product: the ability to recover from that [type of problem]. With just one mail system then we don't have the problems of contending mailers.

We are currently in field test on both TOPS-10 and TOPS-20. On the TOPS-10 side we have only been in field test for about three weeks. The reports that we get back [indicate] that it is fairly stable, in fact very stable. We have actually only had two QAR's reported against it and both of those have both been answered. We have no current outstanding problems with the mail system. We are expecting to end

the field test around the end of June. It will be going into the SDC on an update tape; this should happen the end of July. If you have any questions about the product, at the end of the session just let me know.

Fortran-10/20 Update

[This portion of the session is] an overview of the last release of Fortran-10/20 which went out almost a year ago and some of the things we have been thinking about for the next release.

Fortran V10

Version 10 is currently our field image Fortran. It was shipped in June of 1985 to both TOPS-10 and TOPS-20 for KL's and KS's. It is currently the only supported version of Fortran. We began autopatching it with tape 11 and we include both source updates on the autopatch tape as well as EXE file replacement.

We had quite a number of new features in Fortran version 10. It is a full Fortran 77 language [implementation with complete] level validation for all the configurations. We just recently got through revalidating the product at the full level.

There is a portability flagger that will allow you to discover those things which are either not ANSI compatible or not VMS compatible.

We also put in support for G-floating on both TOPS-10 and TOPS-20. We added some industry standard magtape features. We put in some compatibility features to allow us to be similar to VAX Fortran and we made some debugger enhancements. It was a major release.

Incidentally, it has been a very stable release over the past year. We've had very few bug reports from customers. We've had customers tell us that it is really one of the more stable versions of Fortran. If there are any of you that have not yet put it up, I strongly encourage you to do so. Our experience with it has been very positive.

The portability flagger in version 10 gives you three options. It allows you to determine those things which are not ANSI standard, those things which are not the same as on VMS, and you can get a collection on both. It gives you compile time warnings and, for those things that are not detectable at compile time, you may get some run time warnings.

These are the supporting configurations for TOPS-10. It runs under 7.02, and it will be supported under 7.03 on both the KL and the KS using LINK version 5.1. For TOPS-20 for those of you who also have TOPS-20 systems, it runs under 4.1, 5.1, 6.0 or 6.1, [on] KL's, KS's with LINK 6.0.

The documentation that we put out with version 10 was

fairly extensive. There's a large language manual, a pocket guide, the Fortran-10/20 VAX-11 Fortran compatibility manual showing you the differences that we know about between the various Fortran's. There is a large math library manual giving you some of the algorithms, some error analysis that we did, and there is an installation guide.

For those of you who haven't installed it, it's a rather simple matter. Version 10 and version 7 of Fortran can co-exist on the system. What we usually do is to put the new Fortran up on new for a while just in case and leave the old Fortran on SYS:. That allows your people to use either one for a short time to ensure that the new Fortran works properly.

Fortran Futures

We have a number of things in mind for the next version of Fortran. While it is not an announced product per se, we would like to give you some idea of the things we are thinking about and solicit things that you may desperately need. This will be probably one of the last times to give us input on what you would like to see on the next version.

The things we have in mind, however, are performance improvements and supporting [long?] symbols along with symbols for VMS.

In the area of performance improvements, we would like to improve the object code for character data, specifically for single character assignments and single character relational. We believe that the speed-up can be substantial for these very common cases. We would also like to put in support for VMS symbols, to be VMS compatible, up to 31 characters in length, dollar sign and underscoring included. You will still get warnings if you are using the flagger about variable names that contain more than six characters.

TOPS-10 V7.03 Performance

I have the impression that everybody has seen all the 7.03 slides before, so I thought I would talk about something different. This be the theory aspects of the performance issues.

We haven't had time to get a really good feel for exactly what's going to happen with 7.03 performance. We did a number of things to try and counteract places where we knew we were going to have an effect [on performance] with some of the new features we were adding in 7.03. Normally during the course of our field tests, we get some data from the field test sites. Western Michigan is one that in particular we did get performance data from. We got a little bit from them but not as much as we would like to get [in order] to say anything more definite about performance. We also get impressions from the field test sites, WMU specifically, because they have a particular

package that they run every time and usually we try to compare it against the previous release.

Performance, as always, is composed of overhead, throughput and responsiveness, and sometimes you end up trading off different things. We have not really heard of a problem with throughput from the field test sites. We have had a few complaints about overhead on all CPU's, and some complaints about responsiveness, but these haven't been quantified yet.

What I would like to talk about, as I said, are the things I know we worked to improve on.

In the area of Ethernet and DECnet the main killer we found early during our development was routing overhead. Toward the end, Bill did a heroic amount of work to try and cut down that amount of overhead by eliminating one of the routing vectors and postponing the updates until they were absolutely necessary. Basically, what this can do is cut down the number of nodes that you recompute. It makes the assumption that only one or two nodes change when you get a routing update instead of [assuming that] all of the nodes change. It only tries to affect the few nodes that are changing, and also eliminates unnecessary updates. For our configuration, we were running around 90% overhead on our POLICY CPU's sometimes. The Enet [DEC's internal DECnet] configuration is a little bit unusual and I'll explain about that in a minute. With the changes we lost about 30%-40% of our overhead on the boot CPU. As the note says, that [result] was [with] the Enet configuration; your mileage may vary. The main difference with the Enet configuration is that we are configured with a very large number of routing nodes, and I speak about that on the last slide when I talk about things you should watch out for with 7.03. A large number of routing nodes can make your overhead go up tremendously. The other thing [unusual] about Enet is most people do not have 5,000 or plus node networks.

In the area of virtual memory, with the larger address spaces possible with 7.03, we did a number of things to try and improve the virtual memory performance. There is a small [over]head on the CORE UUO that we found out with Western Michigan University. We are hoping that that is not a heavily trafficed area of code and we are also putting in some work to try and speed it up.

For the virtual memory we implemented paging queues. In the past when the page fault handler said to page out a page or when the user said page out a page, it was written immediately to the swapping space. That is not necessarily the case in 7.03. Pages go on to a number of queues; they migrate into one of two "in core queues". Following that, they will move to an "in progress queue" when a monitor decides it needs to swap out the queue, and then it will move to an "out" queue. As long as it is in one of those three queues, if the user faults for it in the time before it disappears off the out queue and gets recycled as a free page, the page fault will not generate an I/O. A

secondary advantage of this is that pages that get paged out get to be collected as contiguous chunks of space on the swapper. The swapper handles large amounts of swapping much more efficiently than individual pages. The benefit you will get out of this even if you do not use virtual memory per se is that IPCF pages which the monitor has been in the habit of paging out at the drop of a hat will collect on the paging queues instead of immediately going out to the swap space.

Another thing we have changed with 7.03 is to increase the swap space to 256K pages per unit, [with] the same limit of 8 units. There is a dependency there, however, on the cluster size: the swap file must fit between the bat blocks.

An additional change we made to improve end performance is to move the page fault handler into the monitor and hopefully write a better one. There is no change in the user mode any more for that. User PFH's will still work with non-extended programs. We have attempted to keep the mechanism alive, but we do recommend you use the monitor PFH if possible. Your user PFH will not get merged into your image from a file, the PFH must be set up when the page fault occurs.

There have also been some improvements to the SAVE/GET code to do larger IOP's and increase the virtual SAVE/GET cases.

Early on in 7.03 we picked up some scheduler improvements from Western Michigan. What these implement are one high segment retention time, so that if you have applications that do a considerable number of GETSEG's, you can keep these segments around on the swap space more easily if you tend to be recycling them a lot.

[We] also [got] a free core goal. You always know that it is an improvement that if you can do the swapping when you are idle and not when you have to swap something. The free core goal is one way that Western Michigan has tried to quantify when the system is idle and what you should actually be shooting for. It allows you to set a goal so that if the swapper is idle it will attempt to swap out long term waiters to meet the free core goal. These things are affected by schedule set parameter; the default is the same as the 7.02 behavior.

[Here are some] things you should watch out for with 7.03. The memory requirements [are] bigger and better, but mostly bigger. The minimum configuration for the KL's is 768K now. Insufficient memory can negate the effect of the paging queues. The pages will only stay around in memory as long as there is sufficient memory to hold them. If you do not have that memory you will have the added overhead of actually putting the pages on the queue, and you won't get the benefits of either the lack of fragmentation because it will be going out in one or two page chunks. Basically, you will not have any of the improvements.

Second thing to watch out for in 7.03 are routing wars, this is particularly true if you do run with a lot of routing

nodes on your network. You should try to run with as few as possible. First of all, each routing node generates a lot of traffic for all the routing updates. And second of all, if somebody does happen to have a bug and starts disagreeing about what the routing vector is to a particular node they will exchange messages with each other as they argue about who was right as to how far away a certain node is. This is also possible on 7.02. However, on 7.03 [with] Ethernet, the messages can go many times faster so you can chew up a lot more CPU resources doing that. If you have any questions about this, we will be taking them at the end or you can talk to me at the booth. If somebody wants to know about the features that are actually in 7.03 you can also talk to me about that.

TOPS-10 Futures

What I would like to talk about is what we are thinking about for 7.04, our next release. We are currently in a very early stage in the development cycle. In fact, we're in pre-internal phase 0. Phase 0 is the point where we actually sit down with product management and hash out the different features which are going to be in 7.04. We are not to that point yet. In fact, internally in the development group we are just now starting to sit down and write down our ideas about what we believe belongs in that release and what we believe the customers actually need for that release. In terms of input, the session later today at 3:00 is a very good time to let us know what you need for that release; it is a perfect time in the cycle.

Our goal for 7.04, the next release, is stability. We are going to be working to make this product a very stable release, very reliable, maintainable, all the good things that you have heard in the past with the RAMP philosophy. So for goals and particular things like reliability we are going to be doing a lot of concentrating on SPR's. We are going to be answering SPR's, we are going to be looking through the SPR's to find those particular sections of code or those particular pieces of the product that do not function very well or have more problems than other sections. We are going to concentrate on those particular things. Hopefully, we will be able to reduce the sticky points we have had over the last years. Along with SPR's in general, there are a lot of problems we know of that exist in some of the releases that we are going to try and address, things that haven't been SPR'd that have just been lying around.

Maintainability-wise we're going to be doing as much code clean up in the monitor as possible and in the utilities. The idea being to make the flow code very easy to read, very understandable, [and to allow there] less chance of any bugs actually being in the code because we will be doing some inspection of that.

We are going to be doing some work to complete things that are in and implemented in 7.02, 7.03 past monitors, that for some reason or other that didn't quite get finished.

One particular thing that comes to mind is the QUEUE UUU. There is some small missing pieces of the QUEUE UUU that you can not do, [such as the requirement] that you know you must use IPCF's to actually have happen. So we will be working on areas like that to just basically stabilize the code and complete the ne functionality that we started in the past.

Other things we are thinking about in terms of making the product more maintainable for customers and in fact for ourselves internally is that if there there are any utilities or any functions that more than one program might be doing, we are going to try coalesce these things. At the moment, I don't know of any specific examples, but if we can find a function that is being done very similarly by two programs, we are going to make some attempt to put those programs together into one.

The other big area that we are going to be investigating is performance. One of the things that I would like to see happen (in fact, I was the major force behind getting it on our list) is some work being done to actually go in under induced loads and identify specific hot spots in the code, so that we can go in and address specific areas that under some circumstances we know will give bad performance. One of the things that was done in 7.03 was the routing vector fixes. That in particular under some circumstances (like our configuration) has very dramatic effects; on smaller configurations it might not be as much a win. There are other circumstances I know everybody's load is different so that different sites have different things that they do a lot of, so they'll have different areas that we can concentrate on.

As I say, since we are very early in the process right now, just starting to think about what 7.04 entails, there is nothing specific I can tell you about what we are thinking about, but what I would encourage you to do is, given some of the things that we are interested in doing and think that we believe need to be addressed, to come to us at the 3:00 session this afternoon and let us know what it is you need for us to accomplish this goal from your perspective. If anybody has any specific questions on specific features, I am more than willing to talk about them either off-line or at the booth.

Question & Answer Period

Q. What plans do you have for ANF-10?

A. At the moment, we have no developments planned for ANF-10 but we expect to have it continue working in the same fashion that it works right now.

Q. Is there any hope of getting extended addressing and Fortran on the TOPS-10?

A. Probably not.

A. (Kimo) For the futures [portion of the talk], I neglected to mention that there is a full documentation set down in the booth, in the documentation area.

Q. Right now we are running 7.01A, and we have had people looking at going to 7.02. One of the issues is, that our accounting system still uses FACT files even though we have gone to Galaxy. It is feasible and have any of the test sites of 7.03 jumped directly from 7.01A to 7.03? There would be pros and cons of doing that.

A. There is a program on the tools tape that will convert your usage files back into FACT files for your accounting.

Q. (Bob Derchie, PRC) We are also going from 7.01A to 7.03 and I think the main question we have is—what were the problems of going from 7.01A to 7.02 and are we going to have the same problems going from 7.01A to 7.03?

A. Perhaps some of the people who went to 7.02 would be better able to answer from the problems they had.

Q. (Brent Stern - University of Ontario) I have a question from another site who is interested in PUSH and POP. Is there a convenient way of communicating between processes that use that utility? Funny space shared for example.

A. Funny space tends to go downward but not upward. IPCF between different contacts works.

Q. (John Edgcomb) We finally got a VAX. The VAX compatibility is an area that we should be concerned with. There are two specific areas that I like, one is the Fortran and the other is the monitor. [What I would like to see in] Fortran [is] the variable field format [that allows you to] specify at one time how wide a field is, that can be put on with an angle bracket in the format.

A. I don't believe it is on our list of things at this point. Why don't you catch me later and give me a fuller description of what you would like.

Q. The second thing is in the monitor and that is the recall command. I don't like typing, and when I make a mistake and have to retype the whole line to interchange two letters. I would like to have the recall command preferably with full line editing, but if not, then just the ability to go back a couple of commands and re-execute it would be heaven.

A. (Kimo) I didn't bring the noted side from the VAX group. Does anyone have an answer for this other gentlemen by the way? Did you want to talk to them off-line about the problems going from 7.01A to 7.02? I guess there weren't any problems.

Q. (Stan Baer, Harvard Business School) On the DECmail MS, it sounds like it may be the solution to a problem that recently came up. We had some faculty members who are interested now that we are on BITNET on our IBM

mainframe. Interested in having some way of moving stuff from the DEC-10 over out through BITNET. If we got a MicroVAX with j-net on it in between, and we got MS on the DEC-10 they could send things to the vax. Could it be sent in such a way that it could kind of keep going and go out through BITNET?

A. We do a few special things. Like I said, if the mail will be sent to the VAX using the Mail-11 protocol, you have the ability to format a message within quotes. Now if you can route that on the VMS side to the special mailer to go out to the j-net or the BITNET, you would probably be able to do that. There is some software called mr gate that runs on the VAX, and I'm not sure what they do to talk to other mail systems, but that is the route we have gone to get mail into ALL-IN-1 mail. There is some special other mailers that I have seen on VMS which interface directly to the arpEnet. Nothing that will go directly from the DEC-10 to another network, but if you can get it to the VAX and you have the VAX software to do that, I suspect you can do it.

Q. I was willing to get a VAX to do it, but my preference would be, is there any way to get things directly from the DEC-10 out to any other network?

A. Do you have the Bliss-36 compiler?

Q. No.

A. It's probably going to be a problem for you then, because most of the MS is written in Bliss, the scheduler for it is, many of the utilities. The individual drivers though, that we have are written in Macro. For SMTP and for VMS mail, they're both in Macro. I don't know what we would have to do to allow you to add another driver without modifying the utilities. Most of the I/O is done in Bliss code, and usually there are common drivers for that. You might have to write a special driver and then be able to interface it that way. It's possible but I can't guarantee it.

Q. (Frank Francois, FHLBB) On the MS mail it is bundled in 7.03, but can you run that newer package in 7.02?

A. No it can't. It is only supported under 7.03 for TOPS-10 and 6.1 for TOPS-20.

We do a lot of things with the new Galaxy, especially having to do with account validation and things like that.

Q. (Brent Stern) I am running 7.02 with autopatch tape 8 installed. I have 4 other autopatch tapes sitting on my shelf. I am not going to be going to 7.03 because I am just so busy with so many other things and we see the end of the line. Is it worth my while to put in the effort of upgrading to the 12th autopatch tape, or is there a 13th coming? Am I going to see better stability then I've got now, which quite frankly is pretty good? Are there some really good reasons for doing that? Am I going to get better support?

A. (Kimo) Well, 7.02 support will end on the usual date that it ends, 6 months after the last customer ship. If you don't see any problems and you don't have any plans on upgrading anyway, I can't think of anything really obvious, any particular reason why you would want to go, other than the support issue which like I said, 6 months after the last customer ship for 7.03 it's going to be a new point anyway.

Q. My understanding is that 7.03 requires Galaxy V5, is that the right way around?

A. Sort of in the same way that 7.02 required Galaxy V4.

If you don't want the new features, officially the only supported position is Galaxy V5. If you don't really want the new features, you can get away without running Galaxy V5.

Q. So all the Galaxy V4 EXE's will run on 7.03?

A. I believe that is true. Like 7.02, Galaxy V2 would run with 7.02.

Q. (Chuck Bacon, NIH) Is there any likelihood that either mode 2 or mode 3 files will be supported for disks?

A. (Kimo) What do you want them to be?

Q. So I can open a file in mode 2 and have 8 bit bytes to and from the disk. Right now if I open in mode 2, I get 36 bit bytes.

Mode 2 is pim, that's only technically legal for terminals.

Increasingly, we are getting binary files stored in 8 or 32 bit modes, and we are trying to discourage people from writing binary data in 36 bit mode and instead encouraging them to treat all binary data as 32 or 16 or 8 bit data. this is working, KERMIT works wonderfully with 8 bit data, it also works with 36 bit data but you have to tell it it's text and nobody understands the data when it gets to the other end. It's beside the point. If an organized blessed mode existed for 8 bits, I think people would really like that. I know you open and enter the file and then you go and set the bite pointer and then off you go.

A. (Kimo) We'll note that, if it was it probably would not be one of the existing modes, it would probably be a new data mode.

Q. All we've had to do to every monitor we ever had was to, there is some bit you flake someplace that says is this a legal mode and off you go and it works. and there is someplace where you have to change the count from multiply it by 4, divide it by 4 or do one of those things. For some reason or other our fix got it so it writes nicely on the disk but when you read it back, it still comes back funny, that's our fault. But we would like to see the company bless it.

Q. How much trouble should I expect retrofitting FACT files throughout and not having anything to do with usage in 7.03?

A. (Kimo) You can skip the mail system. There are a lot of reasons why you might want the rest of the accounting for 7.03. Encrypted passwords, password expiration, minimum length all those good things you won't get without going to the Galaxy 5 accounting. In terms of just your accounting data, the utilities that's on the tools tape will convert the accounting data from usage format to FACT format which you can then run from your downline accounting program if that is what you want.

A. MS is not with the 7.03 FDC tapes, it will be sent out on a later update tape.

TOPS-10 V7.03 Users Panel

Frank J. Francois
Federal Home Loan Bank Board
Washington, DC

Abstract

Representatives from TOPS-10 V7.03 field test sites discussed their experience with the new release of the operating system. Ralph Bradshaw represented Johnson & Johnson, and Robert McQueen represented Stevens Institute of Technology.

Johnson & Johnson

Johnson & Johnson became a field test site for 7.03 because an impending move required them to have Ethernet running on the TOPS-10 system. TOPS-10 7.03 with DECnet Phase IV provided Ethernet support on the DEC-1095 with the NI.

Johnson & Johnson 7.03 Findings

- J&J used two terminal servers. They ran dedicated service on the terminal servers. They encountered a problem using V2 of the terminal server software with a 7 bit host, dedicated service and autobaud. The terminal server beat itself to death never servicing the line. This resulted in a 1% throughput. Stevens' took away the autobaud as a work around to this ts problem.
- With dedicated service you also lose sight of the host. You can connect to terminal server and connect to host (if host isn't there you get no indication of a host being down). No message such as "host not available" comes back to the terminal.
- No disconnect from terminal server on logout. Host drops you to the terminal server *or* you do a carriage return and you end back up to the host—host drops you and you are back to the terminal server. Version 2 of the terminal server doesn't let you know what port you are on. The terminal server's next software release is supposed to fix the problem.
- The terminal server makes flow control much more complex. There is flow control between terminal server and host and terminal and terminal server. If you have page size set to 24 lines and the DEC-10 stops the output, there is no way to restart the DEC-10 output unless you have the unpause bit set up.

When you try to restart the DEC-10 flowing by hitting a Ctrl/Q - the terminal server doesn't pass it along to the DEC-10. You can, with the unpause-bit, set up any character to become a Ctrl/Q when it hits the DEC-10. Johnson & Johnson generally uses a Ctrl/A as the alternate flow control character.

- security and accounting was a major change.
 - Password encryption is available.
 - Password expiration in accounting daemon now available. There is a new program available called NEWACT allows it to set the expiration time. This program generally only run once so be sure you have the correct parameters.
 - The conversion to the new accounting program is straight forward but it does take some studying of the documentation to insure a complete conversion.
- The DECnet File Access Listner behaves differently; FAL in 7.03 is now multi-threaded.
- Galaxy V4 to Galaxy V5 transition went smoothly. Operators had little trouble adjusting to the new Galaxy.
- J&J is experiencing good up time now; 7.03 is reliable.
- A problem with corrupted disks (UFD) was hardware-related and did not necessarily have any thing to do with 7.03. Some sites did experience UFD corruptions.

Stevens Institute of Technology

Stevens' configuration included 1090 with MG20, NIA 20, MCA 25, ANF network, DN 200, and a DN20. They have been running with an Ethernet for several years. 50 PRO 350 IP, VAXcluster and MicroVAX II on same network.

Stevens Institute 7.03 Findings

- NI-based DECnet works fine. Stevens runs multiple DECnet areas. There was a problem with the DN20 being Phase III, so they will have to swap that out.
- Other 7.03 stuff works fine. Alternate context is really great. Extended address user mode has a few bugs.
- Galaxy. The multiple driver in line printer support is great. Can write your own module for foreign equipment without a lot of changes to the DEC sources.
- Accounting. There are new password and security features. The ability to put distribution location on output is a real plus.
- CTERM doesn't work with RSX and PRO 350 systems.
- The DEC-2020 had performance problems and is no longer running 7.03. Cobol-based and Scope applications suffered.

Question & Answer Period

Q. (Ralph Smith) Can you give me an idea on size of system and users response?

A. (Stevens) 1090 1 meg, 40-50 jobs slows down system. 2020 monitor grew about 20 pages.

A. (Johnson & Johnson) 1095 2 meg CPU power is used up before you run out of memory.

LCG Software Products Update

Carla J. Rissmeyer
Computation Center
The University of Texas at Austin

Abstract

Dave Braithwaite and Susan Porada of Digital Equipment Corporation presented this update on the Large Computer Group's software products. Mr. Braithwaite talked about engineering's strategy and work in progress, while Ms. Porada discussed the directions of the documentation group.

ENGINEERING

Product Strategy Since 1983

- Expand the TOPS environment
 1. CI, HSC disks, CFS
 2. MG memory, MCA-25
 3. improved performance and reliability
- Interconnect with other DEC products
 1. Ethernet: gateways, LAT
 2. DECnet Phase IV
- Provide migration path
 1. integration tools
 2. documentation
 3. Datatrieve
 4. language compatibility

Current Happenings

- TOPS-20 7.03 to SDC
- MS-10/20 in field test
- TOPS-20 6.1 shipped

Work in Progress

- Microcode improvements
- TOPS-20 7.04
- TOPS-20 V7
- Fortran V11
- DECmail/MS-10/20
- DIU
- Reduction of SPR backlog

Strategy for LCG Engineering

- Migration of engineers to cluster projects
- Add large systems features to VMS
 1. DX20 for VMS
 2. VAXcluster console
- Future possibilities
 1. very large disk farms
 2. cluster management
 3. high availability
- Continued focus on needs of large systems environment

DOCUMENTATION

Development Releases

- TOPS-10 7.03 in SDC
- DECmail/MS in field test
- DIU/RMS-20 in late summer
- Fortran V11

Completed Integration Documents

- Network compatibility
- Operating system compatibility
- Language compatibility

Documentation Stabilization Goals

- Get up to field image level
- Ensure adequacy
- Provide soft copy

New Projects

- VAXcluster console
- VAXcluster applications guide
- VAXcluster systems handbook
- VAXcluster systems QUORUM magazine

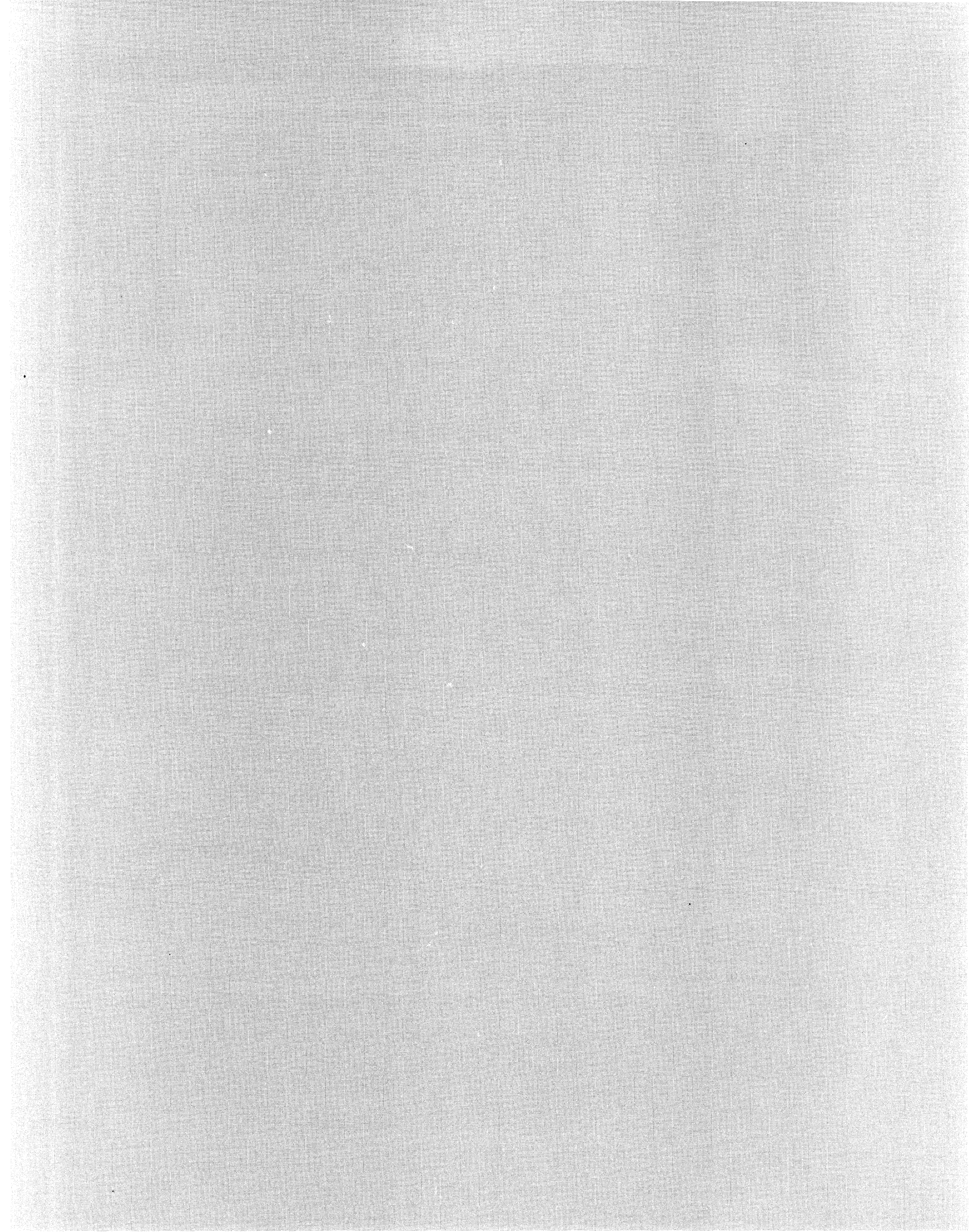
COMMENTS FROM THE Q&A PERIOD

Concerns were raised about the quality of the TOPS-20 V6.1 manuals. They were not typeset.

Users requested that documentation for the final releases of the TOPS operating systems be provided in both hard and machine-readable (DSR format) copy. This would enable users to maintain documentation easily.

When TOPS-20 development ends, users ask that the documentation be as complete as possible, including MONSYM, MACSYM and ACTSYM listings, as well as Bug lists and monitor tables.

LANGUAGES AND TOOLS SIG



Typesetting Articles for the DECUS Proceedings with L^AT_EX

Barbara N. Beeton
American Mathematical Society
Providence, Rhode Island

Abstract

The *DECUS Proceedings* have traditionally been published from copy supplied by the authors, prepared according to rules devised for typewritten material. The power of the computer typesetting language T_EX,¹ through the macro package L^AT_EX, has now been applied to this task, and a formatting package, named DEPROC, has been submitted to the DECUS Program Library for use by authors who have access to a working T_EX system. (The T_EX program and related software, created by Donald Knuth of Stanford, are in the public domain.)

This paper presents the important features of the L^AT_EX implementation of DEPROC and, through examples, shows how it is to be used. Use of DEPROC, which is encouraged, will produce the author's work, nicely typeset, in the standard *Proceedings* format. There is a general description of how the package works and of the mechanical requirements for camera copy of *Proceedings* articles, which will be created on the author's local output device.

No prior knowledge of T_EX or L^AT_EX is required, but authors using DEPROC will be expected to learn some rudiments, especially if their papers contain special notation or formats such as tables.

The *DECUS Proceedings*, like the conference proceedings of many other organizations, is rushed to publication as quickly as possible so that the material will reach the conference participants and other interested readers before its value is diminished by time. Reproducing author-prepared copy eliminates the considerable bother and expense of typesetting, proofreading and corrections. The published document should be compact, uniform in appearance, and readable, regardless of the kind or quality of printing device available to the author. For these reasons, instructions to authors have heretofore assumed that nothing more elaborate is available than an ordinary typewriter or dot matrix printer.

To enforce uniformity, the author is provided with "model paper", on which are printed (in non-reproducing ink) column and page borders, alignment marks, and instructions for placement of title, author, and the other parts of a proceedings article. The dimensions of the model paper are almost always larger than those of the published Proceedings — this permits more text to be packed onto each page, and also improves its appearance or "quality" when photographically reduced, smoothing out the rough edges of letters and symbols generated by a typewriter, dot-matrix printer or other "low-resolution" device.

Within the past few years, advances in laser-printer technology have made good-quality output accessible to a

growing number of users, through a widening selection of low-cost output systems based on print engines with 300 dot-per-inch resolution and (relatively) easy-to-use interfaces. Such devices have been attached to most kinds of DEC computers, and drivers now exist to print the output from such programs as Scribe,² T_EX and Troff. Most low-end laser printers cannot use paper wider than 8 1/2", however, so even if both a good composition program and output printer had been available, until now an author would have been discouraged from using them for mechanical reasons.

The editor of the *DECUS Proceedings* has now agreed to accept typeset copy printed on such a system at 100% on 8 1/2 × 11" paper, provided it conforms to the published format. This article (which has itself been produced by the technique it describes) introduces a package, DEPROC, designed to prepare *Proceedings* articles using L^AT_EX.

What is T_EX? What is L^AT_EX?

T_EX is a public-domain typesetting language created by Donald Knuth of Stanford University. His original aim was to typeset his own books, in particular *The Art of Computer Programming* [ACP], with a quality equal to that produced by the best traditional composition methods. The technical content of these books assured that full

¹T_EX is a trademark of the American Mathematical Society.

²Scribe is a trademark of Unilogic Ltd.

	DECSytem-10	DECSYSTEM-20	VAX (Unix)	VAX (VMS)
Allied Linotronic L100, L300P			Textset	Textset
Apple LaserWriter			Carleton University Textset †	Textset †
Autologic APS-5, Micro-5		Textset	Textset	Intergraph † Textset
Canon			Canon	
Compugraphic 8400, 8600				Kellerman & Smith
DEC LN01			Univ of Washington	Louisiana State U
DEC LN03				DEC Kellerman & Smith
Imagen	Stanford Vanderbilt	SRI Columbia	Univ of Maryland	Kellerman & Smith †
QMS Lasergrafix			Textset Univ of Washington	GA Technologies Texas A&M Textset
Symbolics		Univ of Washington	Univ of Washington	UMass
Talaris	Talaris †	Talaris †	Talaris †	Talaris †
Xerox Dover		Carnegie-Mellon U	Stanford	
Xerox 2700		Ohio State U	Ohio State U	
Xerox 9700	Univ of Delaware		Univ of Delaware	ACC Textset

† Graphics supported

Figure 1: Computer/output device combinations with T_EX interfaces

Information regarding the interfaces shown here can be obtained from the individual listed below for the site. This table and the names of the site contacts were provided by the T_EX Users Group.

ACC (Advanced Computer Communications)
Diane Cast, 805-963-9431

Canon (Tokyo) Masaaki Nagashima, (03) 758-2111

Carleton University Neil Holtz, 613-231-7145

Carnegie-Mellon University Howard Gayle,
412-578-3042

Columbia University Frank da Cruz, 212-280-5126

DEC (Digital Equipment Corp) John Sauter,
603-881-2301

GA Technologies Phil Andrews, 619-455-4583

Intergraph Mike Cunningham, 205-772-2000

Kellerman & Smith Barry Smith, 503-222-4234

Louisiana State University Neil Stoltzfus,
504-388-1570

Ohio State University John Gourlay, 614-422-6653

SRI

Stanford

Talaris Sonny Burkett, 619-587-0787

Texas A&M Bart Childs, 409-845-5470

Textset Bruce Baker, 313-996-3566

University of Delaware Daniel Grim, 302-451-1990

University of Maryland Chris Torek, 301-454-7690

University of Massachusetts Gary Wallace,
413-545-4296

University of Washington Pierre MacKay,
206-543-2386

Vanderbilt University H. Denson Burnum,
615-322-2357

attention was given to the niceties of formatting mathematical expressions, as well as to the structures of documents commonly encountered in technical publishing.

\TeX deals with low-level concepts familiar to typesetters — type size, leading, interword spacing and kerning. It does not incorporate directly the structures an author encounters when writing a paper — title, figure references, bibliographic entries. However, \TeX is essentially a macro compiler, and provides a full vocabulary of low-level functions that can be manipulated by knowledgeable users to create higher-level packages to support the casual user.

One such macro package is \LaTeX . \LaTeX [LT] is a powerful document formatter, providing the capability to format books and reports, with functionality similar to that provided by Scribe.

The DEPROC macro package

In order to use this implementation of the DEPROC macro package, the author of a *DECUS Proceedings* article must have available a working \LaTeX system, which presupposes a working \TeX system. \TeX has been implemented on VAXes and DECsystem-10s and -20s under the standard operating systems. There is also a good selection of output devices available, capable of production output of quality suitable for the *Proceedings*; Table 1 shows the computer/output device combinations known to the \TeX Users Group. (\TeX has not, however, been implemented on PDP-11s, since it requires a larger address space than is supported on those machines.)

\LaTeX may not be available to all \TeX users. (\TeX is a very large program by itself, and routinely adding a large macro package can put unwelcome strain on an already overloaded machine. Some system administrators prefer not to give their users that opportunity.) An earlier implementation of DEPROC does not require \LaTeX , but only \TeX itself; it was described in [DP], and the supporting files are on the Fall '85 DECUS Program Library tapes for Languages & Tools, Large Systems, and VAX.

The present \LaTeX -based version of this macro package is called DEPROC.STY, for “*DECUS Proceedings* style file”. It is an ordinary ASCII file, and has been submitted to the Spring 86 DECUS Program Library for the same systems listed above.

Some preliminary \TeX technical information

An author who intends to use the \LaTeX version of DEPROC should preferably have used \LaTeX already. Nonetheless, a few basic concepts are worth repeating. (\LaTeX is identical to \TeX in many ways. The following discussion will specify \LaTeX only when there is a difference.)

Spacing

\TeX uses different spacing rules in text (paragraphs) and math. Paragraphs are set so that interword spacing is as

uniform as possible. Wider spaces are set after punctuation that indicates the ends of sentences (period, ! and ?). Within math, the best traditions for arranging symbols in two dimensions, including proper spacing, are observed. Thus input spacing is largely ignored, except for its functions of separating words and marking the boundaries of certain kinds of expressions. \TeX considers multiple spaces in an input file to be equivalent to a single space. The carriage return (CR) and the tab character (TAB) are equivalent to ordinary spaces, except in special environments (noted below). And all spaces at the beginning of any line are ignored.

Paragraph breaks

A blank line in the input file indicates a paragraph break. A line is blank if it contains only a (CR) or spaces and a (CR). Multiple blank lines are equivalent to a single blank line. (A paragraph can also be indicated by \par; terms beginning with \ are described below.)

Comments

A comment may be entered on any line; a comment begins with a % sign:

```
% This line contains nothing but a comment.
\newcommand{\cs}{...}% explanatory comment
... Smythe % ***** check spelling *****
```

\TeX will ignore the % and everything following it, including the (CR). Thus, the space ordinarily indicated by the (CR) will be suppressed, and if a space is really wanted between the last item before a comment and the first item on the next line, it must be input before the %. Conversely, if no space is wanted between the last item on a line and the first item on the next, a % can be used to suppress it intentionally.

Control sequences, also called macros

A “control sequence” *cs* is an instruction for \TeX to perform some action or to produce a particular symbol. A *cs* begins with a backslash, \. There are two types of *cs*-es:

- A “control word” consists of \ followed by one or more letters. It is terminated by any non-letter, including a space; multiple spaces follow the usual compression rule, so a special technique (see next paragraph) is required to create an output space after a control word. \TeX is an example of a control word; it produces the \TeX logo.
- A “control symbol” consists of \ followed by exactly one non-letter. Since its length is known, no special terminator is required. \& is a control symbol to produce an &. _ (\ followed by a space) is an explicit space, to be used where an output space should follow an element input as a control word.

New **cs**-es can be defined within a document to make input easier or clearer. A few principles governing **cs** names should be observed carefully.

- Case matters; `\csname` is not the same as `\Csname` or `\CSName`. Try to pick a name that means something to you, and is easy to type.
- Don't try to redefine an existing **cs** name unless you really know what you're doing; results, as they say, "may be unpredictable".
- Never define or redefine any **cs** whose name begins with `'\end'`.

To define a new command,

```
\newcommand{\csname}{...something...}
```

If the name has been used before, **L^AT_EX** will stop and report an error. If you are really adamant about re-using this name,

```
\renewcommand{\csname}{...something...}
```

will assign it the new meaning.

The control symbols `\0, ..., \9` always start out undefined, so they are available for transient use without checking.

A **cs** with arguments is defined by

```
\newcommand{\csname}[2]{...#1...{#2}...}
```

with the number of arguments given in brackets as shown; for details, see [LT].

Math

Mathematical expressions are input between `\(...\)`. Display math is begun and ended with `\[...\]`. For details of math input, see [LT].

Starting a *DECUS Proceedings* article

The first step in preparing an article is to create a file. The first two lines in this file should be

```
\documentstyle{deproc}
\begin{document}
```

This will cause the formatting definitions to be loaded when the file is input to **L^AT_EX**.

Next, enter the "top matter". This consists of such things as the title of the article, the author(s) and their addresses, and the abstract.

Title and authors

For an article with a short title and one author, the input looks like this:

```
\title{A One-Line Title}
\author{Author Name\
  Author's Organization\
  City, State}
```

The double backslashes `\\` indicate line breaks. This technique is also used to break up long titles:

```
\title{Here We Have a Particularly
  Long Title\\That Can't Possibly
  Fit on a Single Line}
```

This will be set (in a boldface font slightly larger than text size) as

**Here We Have a Particularly Long Title
That Can't Possibly Fit on a Single Line**

Notice that the way the lines are broken in the input file is not how they appear in the output — only `\\` matters to **T_EX**. Actually, **T_EX** will break long titles into lines short enough to fit on the page, but a multi-line title usually makes more sense to the reader if the author decides where the line breaks should occur.

For multiple authors, the same `\author` tag is used with `\and` or `\And`:

```
\author{First Author
  \and
  Second Author\\
  Common Organization\\
  City, State}
```

or

```
\author{First Author\\
  First Organization\\
  City, State
  \And
  Second Author\\
  Second Organization\\
  City, State}
```

and so forth, which will appear thus in the output:

First Author and Second Author
Common Organization
City, State

or

First Author
First Organization
City, State

Second Author
Second Organization
City, State

Authors' names (the first line, and the first line after `\And`) are printed in boldface; if an author name is to appear on any other line, begin that line with `\bf` (the **T_EX** instruction for boldface type).

The title and author of the present paper look like this in the file:

```
\title{Typesetting Articles for the DECUS
Proceedings with \LaTeX}
\author{Barbara N. Beeton\
\ANS\
Providence, Rhode Island}
```

One item to look at here is `\ANS`, which becomes American Mathematical Society in the output. This is an example of a “local definition”, something that is not likely to be useful to anyone else, but can save the author a lot of time correcting typing errors. Local definitions that are used throughout an article are best input right after specifying the document style:

```
\documentstyle{deproc}
\newcommand{\ANS}{American
Mathematical Society}
...
\begin{document}
```

Abstract

The abstract is the final part of the top matter.

```
\begin{abstract}
This is a short summary of what
the article is about.
\end{abstract}
```

The heading “**Abstract**” is provided automatically; don’t input it. The abstract may contain more than one paragraph. Paragraphs are separated by a blank line or by `\par`, as usual.

The top matter is now complete. The body of the article follows.

```
\maketitle
(Text of footnotes to the top matter is given here)

This is the first sentence of article text.
...
\end{document}
```

The body of the article

An article can start out with text or with a heading. Three levels of headings are provided by `DEPROC`:

```
\section{Section heading}
\subsection{Subsection heading}
\subsubsection{Subsubsection heading}
```

These produce headings (with extra space above and below, not shown here) in the following styles:

```
Section heading
Subsection heading
Subsubsection heading
```

The first paragraph following a heading will not be indented in the default style. Other paragraphs will be indented a standard amount. To suppress indentation on a single paragraph, precede it by `\noindent`.

Footnotes

A footnote consists of two parts, the mark and the text. These are usually entered as a unit³:

```
... as a unit\footnote{Like this.}:
```

This is equivalent to the two statements⁴

```
... two statements\footnotemark
\footnotetext{Or this.}
```

The two-statement form must be used for footnotes in the title or abstract and in “boxed” environments (which will not be explained here; see [LT] for details). In such cases, the `\footnotetext` should be specified as soon as possible after completion of the special environment.

Footnotes are automatically numbered sequentially starting with 1. Numbers may also be given explicitly, between [...] following the `\footnote...` command. In most contexts, this is optional, but for footnotes in abstracts or in “boxed” environments, the number *must* be given for the `\footnotetext`; the first footnote in this article was produced by the following:

```
\maketitle
\footnotetext[1]{\TeX\ is a trademark
of the \ANS.}
```

Footnote numbers can be reset if necessary by

```
\resetcounter{footnote}{integer}
```

Quotations

Short quotations, of less than a paragraph, are set with

```
\begin{quote}
If you can't fix it, ... {\em Button}
\end{quote}
```

and look like this:

If you can't fix it, call it a feature. *Button*

For longer quotations, use

```
\begin{quotation}
...
\end{quotation}
```

in a similar manner, separating paragraphs with blank lines as usual.

Lists

Itemized and enumerated lists occur in many *DECUS Proceedings* articles. `LATEX` provides automatic counters and up to four levels of nesting. Here is a short example of a two-level itemized list.

³Like this.

⁴Or this.

-
- Small figures which can be set in place, i.e., in the same relative position where they occur in the input file
 - One-column figures to be set at the top or bottom of the first available column
 - Double-column figures to be set at the top or bottom of the first available page
 - Full-page figures

Figure 2: Possible figure formats

```

\begin{itemize}
  \item first item
  \item second item
\begin{itemize}
  \item new level
  \item one more
\end{itemize}
  \item back a level
\end{itemize}

```

Here's what the output looks like, after padding out the text a bit to show how longer items look.

- The first item in this list isn't particularly interesting, but it has to be long enough to make two lines.
- The second item isn't either.
 - Even going to a new level doesn't add very much excitement to this exercise.
 - We'll do one more at this level.
- Then we'll go back a level to finish things off.

If `{enumerate}` is specified instead of `{itemize}`, the items will be numbered — 1, 2, ... at the first level, a, b, ... at the second level. If the default labels aren't what you want, an overriding label may be specified, for example, `[--]` (used in Figure 2). Each item comprises one paragraph; an unlabeled paragraph can be produced by specifying an empty label. Extra space above and below a list is provided automatically.

Figures

Figures come in the sizes, shapes and page locations listed in Figure 2. Not all these formats are supported yet by DEPROC. In particular, two-column figures cannot be placed at the bottom of text pages.

One-column figures

To get a single-column figure, enter

```

\begin{figure}[loc]
  content of figure

```

```

\caption{Caption text}
\label{label}
\end{figure}

```

loc is the location where the figure is to be placed, specified by one to four letters (in the order in which you find the possible locations suitable), as follows:

h here, at the position in the text where the figure is input.

t at the top of a text page.

b at the bottom of a text page.

p on a page of "floats".

The default *loc* is **tbp**; note that figures will not be placed in-line unless **h** is indicated explicitly. For additional details, see [LT].

Figures are numbered automatically. The optional *label* is provided to permit references in the text to the figure:

```

... shown in Figure~\ref{label}

```

To reserve space for figures which will be prepared separately, use the command `\vspace{dimension}`. Some space is automatically skipped above and below a figure, and also between the figure and the caption, so the dimension given with `\vspace` should be precisely the height of the item to be pasted in.

Two-column figures

Double-column figures can be placed either at the top of a text page or on a separate page of "floats". The command syntax is the same as for one-column figures, except that the "star" notation `\begin{figure*}... \end{figure*}` is used.

Full-page figures

Full-page figures are a special case of two-column figures, specified by `\begin{figure*}[p]... \end{figure*}`. A full-page figure will be placed on the first available page. If space is to be reserved for insertion of a separately-prepared figure, use the `\vspace` command with a suitably large dimension; `8.5in` should be sufficient.

Tables

L^AT_EX contains powerful table-formatting capabilities. However, since their use is specialized and the rules somewhat complex, specifics are not presented here; see [LT] for details.

Verbatim

Verbatim items are printed in so-called “typewriter” style, using TeX’s `\tt` font. In-text verbatim items are preceded by the command `\verb` and enclosed within a pair of identical characters which do not occur within the verbatim string (except a space, a letter, or `*`), for example, vertical bars `| . . . |` or ditto marks `" . . . "`. Blocks of verbatim code are delimited by

```
\begin{verbatim}
...
\end{verbatim}
```

`\begin{verbatim}` and `\end{verbatim}` should be on lines by themselves. Within verbatim mode, `\CR`s are obeyed as line breaks, not spaces. An input line that is too long for the current column width will be broken at a space if possible, and the remainder of the line hanging indented on the next output line; since this may change the meaning of the verbatim passage, such passages should be checked with special care in the output. Overlong lines also frequently result in overfull `\hboxes`, which are listed in the transcript file. To mark overfull `\hboxes` clearly on the printed output (with black boxes: ■), specify

```
\documentstyle [draft]{deproc}
```

at the beginning of your input file. The `draft` option can easily be removed when you are ready to prepare the final copy.

A passage occurring between `\begin{verbatim} . . . \end{verbatim}` is treated as a unit by L^AT_EX — if it is too long for the vertical space available, it either will be carried over as a unit to the next column or page, or will result in an overfull `\vbox`, which will be noted only in the transcript of the L^AT_EX run. In such a case, the best remedy is to break the passage in two, by inserting another `\end{verbatim} \begin{verbatim}`.

Verbatim mode is suitable for program listings, indicating keyboarding instructions, file names, and similar uses.

References, bibliography

References in text to items in the bibliography are input as

```
\cite{label}
\cite[text]{label}
```

where the label is the same as that specified for the item in the bibliography. For a label “ABC”, the reference will be rendered [ABC]. See [LT, pp. 73, 189] for further details.

References may also be made to figures, tables, or anything else for which you have established a label:

```
\ref{label}
\pageref{label}
```

`\pageref` is not immediately useful — the page numbers generated by TeX will not be the same as those assigned when the collection is put together in the editorial office. This feature would become useful if it ever becomes possible to submit articles electronically, as L^AT_EX input files.

Before you start to input the reference list, some housekeeping is required — you must decide what you want the list to look like. This is what the input looks like for one of the items in the reference list at the end of this article:

```
\bibitem[TB]{TB} Knuth, Donald E., \TB,
Addison-Wesley and \AMS, 1984.
```

(`\TB` and `\AMS` are among the local definitions for this article.) Default output looks like this:

```
[TB] Knuth, Donald E., The TeXbook, Addison-
Wesley and American Mathematical Society, 1984.
```

Labels may also be omitted, or numbered sequentially. Begin the reference list with this command:

```
\begin{thebibliography}{widest label}
```

This will generate the **References** section heading, and establish the item indentation, using the width of the specified label.

If you do not wish to use labels, substitute `\omit` for the widest label. (The `{ . . . }` are still required in the context of `\bibitem`.)

```
\bibitem{} Knuth, Donald E., . . .
```

will result in

```
Knuth, Donald E., The TeXbook, Addison-Wesley
and American Mathematical Society, 1984.
```

Note that `\cite` cannot be used if references are unlabeled.

If your labels are numeric, no labels need be entered — items will automatically be numbered sequentially. However, the widest label must be specified at the beginning of `thebibliography` environment. The input

```
\begin{thebibliography}{99}
\bibitem{TB} Knuth, Donald E., . . .
```

will now look like this:

```
[2] Knuth, Donald E., The TeXbook, Addison-Wesley
and American Mathematical Society, 1984.
```

Caveats

DEPROC and this article were created on a DECSYSTEM-20 at the American Mathematical Society, running L^AT_EX version 2.08 under TeX version 1.3. The AMS installation is standard in all ways.

With one exception, none of the changes to the TeX program since version 1.0 should have any noticeable effect on an article produced with DEPROC. The exception is

large, complex tables—tables incorporating many boxes and rules require large amounts of $\text{T}_{\text{E}}\text{X}$ memory. Memory management was radically changed in version 1.3 to make more memory available to the user without actually changing the physical memory allotment. (Otherwise, if you run out of memory, the most likely cause is an input error.)

Although thorough testing has been attempted, no one outside the AMS has tried to use **DEPROC** yet, so bugs are sure to be found. In fact, the version of **DEPROC** first placed in the Program Library should best be considered a beta test version. If you find a bug, please communicate it to the author, accompanied by an example which demonstrates the bug as simply as possible. Suggestions for improvements are also welcome. Send everything to

Barbara Beeton
American Mathematical Society
P.O. Box 6248
Providence, RI 02940

References

- [DP] Beeton, Barbara, Typesetting articles for the DE-CUS Proceedings with $\text{T}_{\text{E}}\text{X}$, *Proceedings of the Digital Equipment Computer Users Society*, USA Spring 1985, 349–356.
- [ACP] Knuth, Donald E., *The Art of Computer Programming*, Addison-Wesley, Vol. 2, second edition, 1981.
- [TB] Knuth, Donald E., *The $\text{T}_{\text{E}}\text{X}$ book*, Addison-Wesley and American Mathematical Society, 1984.
- [LT] Lamport, Leslie, *L A T E X, A document preparation system*, Addison-Wesley, 1985.
- [TD] Southall, Richard, First principles of typographic design for document production, *TUGboat* Vol. 5 (1984), No. 2, 79–90; Corrigenda, Vol. 6 (1985), No. 1, p. 6.
- [TUB] *TUGboat, the Newsletter of the $\text{T}_{\text{E}}\text{X}$ Users Group*, $\text{T}_{\text{E}}\text{X}$ Users Group, c/o American Mathematical Society, P.O. Box 9506, Providence, RI, 02940.

An Introduction to T_EX and L^AT_EX

Samuel B. Whidden
American Mathematical Society

LT012, DECUS Symposium, Dallas, Texas

Abstract

This is the first of two sessions in which examples of L^AT_EX usage are given. The author hopes to demonstrate that L^AT_EX, a document formatter based on the T_EX typesetting system, is both powerful and easy to use. This session briefly describes the origins of T_EX, and L^AT_EX's relation to T_EX, and then presents examples of the author's use of L^AT_EX to do some fairly common and useful things.

This session is intended to introduce new users to typesetting with L^AT_EX by offering a few examples of L^AT_EX's use. Experienced users of L^AT_EX won't learn much from this presentation, but those who haven't been exposed to it will get an idea of what it can do, and with what degree of difficulty.

L^AT_EX is a system of typesetting macros and definitions built on top of Donald Knuth's T_EX typesetting language. T_EX is a powerful language that offers nearly unlimited flexibility in composing and typesetting difficult material. It was devised originally to typeset mathematics and allows very complex mathematical formulations to be set precisely as the mathematician desires.

My first illustration, Figure 1, is the title page of Chapter One of The T_EXbook, by Donald Knuth. The T_EXbook is the principal document describing T_EX. It's a book you'll need if you intend to use T_EX. The three letters of the T_EX logo are typeset as you see them here by T_EX. These are the Greek letters Tau Epsilon Chi, pronounced 'techhh'. They form, to quote Michael Spivak in The Joy

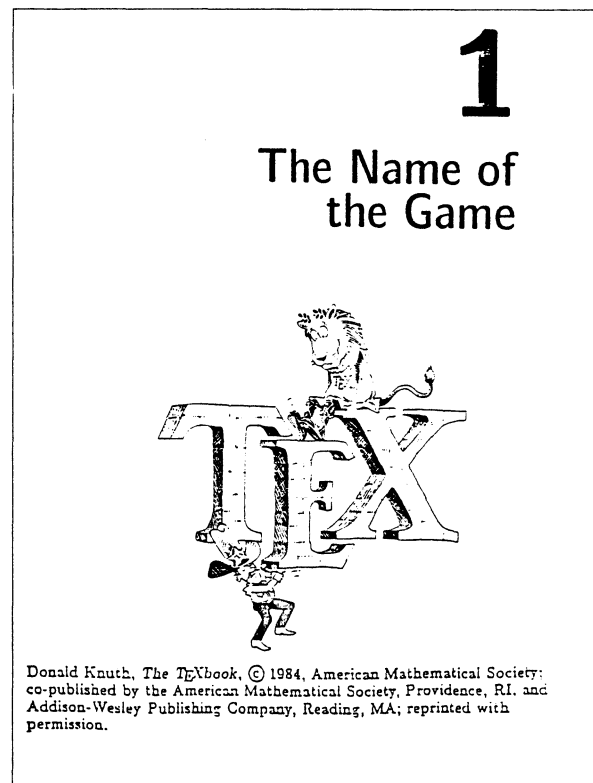


Figure 1: T_EX

of \TeX “the beginning of the Greek word that means art, a word that is also the root of English terms like technology. This name emphasizes two basic features of \TeX : it is a computer system for typesetting technical text... and it is a system for producing beautiful text.”

Figure 2 illustrates \TeX 's mathematical typesetting ability. The top half shows material typeset as final copy, on an Alphatype CRS photo-typesetter with a very high resolution of some 5000 dots per inch. The lower half shows the same material set for proofreading on a low-resolution, 200-dot-per-inch Varian printer/plotter. Examined with a magnifier, these two samples reveal a very different print quality, but note that their typeset format is identical. Element positioning and linebreaking are the same in the final copy as they are in the proofreading copy. \TeX 's output is designed to be device independent to permit exactly this identity of formatting, allowing multiple, low cost proofreading runs which correctly reflect the appearance of the final, expensively-produced copy.

But \TeX is used for many other typesetting jobs besides mathematics. Any difficult and complex typesetting requirement is a candidate for \TeX . Figure 3 shows a page from a manual in the documentation set for VMS Version 4. The version 4 documentation was produced by DEC using an internally-developed typesetting system based on \TeX . Here the various fonts, typestyles, and page formats are produced by \TeX commands. The system is called SDML, for Standard Digital Markup Language, and can output either runoff files, for quick proofreading using low-cost printers, or \TeX files for final copy like this.

A companion program to \TeX is METAFONT. In order to accomplish beautiful typesetting, it's obviously necessary to have beautiful fonts of type. METAFONT creates such fonts, for use by \TeX . First, a type designer draws the characters and symbols. Next, a METAFONT programmer, someone who knows the METAFONT language created by Knuth, writes a rather simple program encoding the character for METAFONT. The programmer codes entire families of characters, adding parameters to allow METAFONT to produce variant versions of the fonts like bold, slanted, or bold slanted. Then METAFONT reads these programs, outputting two kinds of font files. One kind, the .TFM file, simply contains what are called the “ \TeX Font Metrics”—the size and position details of each character. \TeX as shown in Figure 4, reads these .TFM files when it composes a document.

METAFONT's second kind of output, the .GF files, contain the actual raster images of the characters. Both the

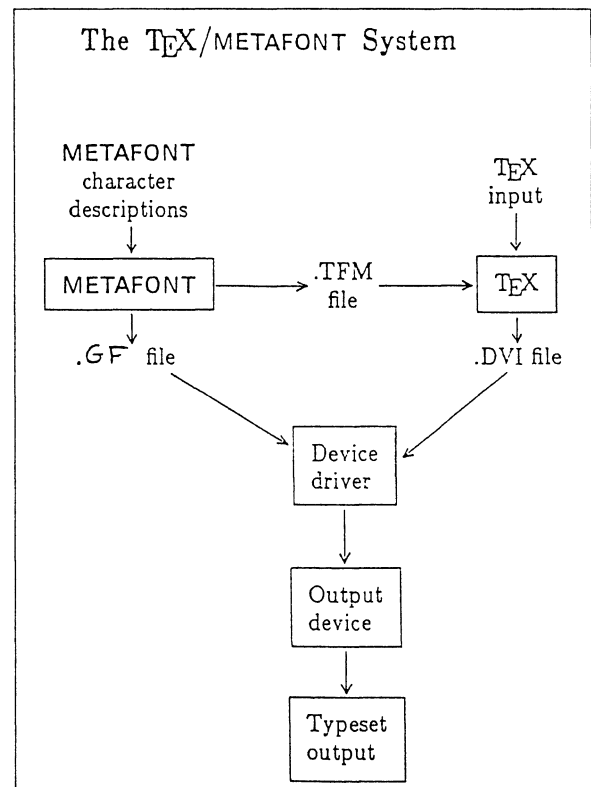


Figure 4: The \TeX -METAFONT System

where $m_n := m_n(k, l)$ is the associated *multiplier* which is also algebraic in k and l . Indeed Jacobi's differential equation

$$(2) \quad n \frac{dk}{dl} = \frac{kk'^2}{ll'^2} \left(\frac{K}{L} \right)^2 = \frac{kk'^2}{ll'^2} m_n^{-2}$$

shows immediately the algebraic nature of m_n , given that for Φ_n . In terms of the nome q one has

$$(3) \quad q := e^{-\pi K'/K} \quad \text{and} \quad q^n = e^{-\pi L'/L}$$

and the standard product relationship

$$(4) \quad \frac{q^{1/12} \prod_{k=1}^{\infty} (1 - q^{2k})}{q^{n/12} \prod_{k=1}^{\infty} (1 - q^{2kn})} = \left(\frac{kk'}{ll'} \right)^{1/6} \sqrt{\frac{K}{L}}.$$

Logarithmic differentiation of (4) combined with application of (2) and $qdk/dq = 2kk'^2 K^2/\pi^2$ produce

$$(5) \quad nP(q^n) - P(q) = (4KL/\pi^2)R_n(k, l),$$

JON BORWEIN

where $m_n := m_n(k, l)$ is the associated *multiplier* which is also algebraic in k and l . Indeed Jacobi's differential equation

$$(2) \quad n \frac{dk}{dl} = \frac{kk'^2}{ll'^2} \left(\frac{K}{L} \right)^2 = \frac{kk'^2}{ll'^2} m_n^{-2}$$

shows immediately the algebraic nature of m_n , given that for Φ_n . In terms of the nome q one has

$$(3) \quad q := e^{-\pi K'/K} \quad \text{and} \quad q^n = e^{-\pi L'/L}$$

and the standard product relationship

$$(4) \quad \frac{q^{1/12} \prod_{k=1}^{\infty} (1 - q^{2k})}{q^{n/12} \prod_{k=1}^{\infty} (1 - q^{2kn})} = \left(\frac{kk'}{ll'} \right)^{1/6} \sqrt{\frac{K}{L}}.$$

Logarithmic differentiation of (4) combined with application of (2) and $qdk/dq = 2kk'^2 K^2/\pi^2$ produce

$$(5) \quad nP(q^n) - P(q) = (4KL/\pi^2)R_n(k, l),$$

Figure 2: Math typeset by \TeX

Overview of the VAX Text Processing Utility

- The last two lines on the screen are reserved for error and informational messages from VAXTPU.
- The EDT prompt "Command:" is replaced with "TPU Command:", and the prompt appears on the third line from the bottom of the screen.

1.4.1 EDIT/TPU Command Qualifiers

You can add qualifiers to the EDIT/TPU command. VAXTPU qualifiers control such items as recovery from an interrupted session and the initialization files that provide the interface with which you access VAXTPU. Qualifiers for EDIT/TPU are listed in Table 1-1.

Table 1-1 Qualifiers to the DCL command EDIT/TPU

QUALIFIER	DEFAULT
/[NO]COMMAND	/COMMAND=TPUINI
/[NO]DISPLAY	/DISPLAY
/[NO]JOURNAL	/JOURNAL
/[NO]OUTPUT	/OUTPUT
/[NO]RECOVER	/NORECOVER
/[NO]READ_ONLY	/NOREAD_ONLY
/[NO]SECTION	/SECTION=TPUSECINI

For full descriptions of the VAXTPU command qualifiers, see Section 6.

1.4.2 Initialization Files

There are two kinds of initialization files that can create or customize a VAXTPU interface: command files and section files.

A command file is a VAXTPU source code file that has a file type TPU. It is used with the VAXTPU qualifier /COMMAND=file-spec. By default, no command file is read when you invoke VAXTPU. You must specify /COMMAND=file-spec, if you want to include a command file.

A section file is the compiled form of VAXTPU source code. It is a binary file that has a GBL file type. It is used with the qualifier /SECTION=file-spec. By default, the section file that creates the EVE interface is read when you invoke VAXTPU. You must specify a different section file (for example, /SECTION=my_section_file) or /NOSECTION if you do not want to use the EVE interface.

Note: When you invoke VAXTPU with the /NOSECTION qualifier, no binary file is read to provide an interface for VAXTPU. Even the RETURN and DELETE keys are not defined. Use /NOSECTION when you are creating a new section file and do not want the procedures, variables, and definitions from an existing section file to be included. See Sections 5 and 6 for more information on /NOSECTION.

Figure 3: VMS Documentation typeset by \TeX

.DVI file, output by $\text{T}_{\text{E}}\text{X}$, and the .GF file are device-independent. Both are read by translators that convert them into device-specific formats for input to driver programs for particular output devices.

If you are a member of TUG, the $\text{T}_{\text{E}}\text{X}$ Users Group, you receive a publication called TUGboat which provides current technical and user information to the $\text{T}_{\text{E}}\text{X}$ community. Figure 5 is a page from TUGboat giving the most recent information on output devices for which $\text{T}_{\text{E}}\text{X}$ drivers exist.

But we don't always need as powerful and complex a tool as $\text{T}_{\text{E}}\text{X}$. For simple jobs, like a letter, a simpler tool would be useful. In a delightful and very positive review of $\text{T}_{\text{E}}\text{X}$ in the *American Mathematical Monthly* (The Mathematical Association of America, April, 1986), Herbert S. Wilf, of the University of PA, says

Is it all perfect, then? Well, not quite. $\text{T}_{\text{E}}\text{X}$ is a non-user-friendly program. It belongs to the 'what-you-see-isn't-a-bit-like-what-you'll-get' school of programs. The learning process was very painful for me. For about one month, using the system several hours per week, I can recall no session in which I wasn't totally surprised by something that printed out. After three months I got moderately proficient at it, and now I can type from my head into $\text{T}_{\text{E}}\text{X}$ fairly fluently...Despite my limitations...it is clear to me that $\text{T}_{\text{E}}\text{X}$ represents a quantum jump in our ability to convey thoughts to paper...

Leslie Lamport, then of SRI in Palo Alto and now at DEC's Systems Research Center, also in Palo Alto, recognized not only the need for a simpler version of $\text{T}_{\text{E}}\text{X}$, but also the potential that $\text{T}_{\text{E}}\text{X}$ itself offered, through its extremely powerful macro facility, for creating such a tool. The result was \LaTeX , a tool second only to $\text{T}_{\text{E}}\text{X}$ itself in power and flexibility, but far simpler and easier to learn and use.

Figure 6, for example, shows a simple business letter written in \LaTeX . Every character on the page was produced by \LaTeX , including the letterhead and the personalization block.

I offer this and some following real examples, not to teach you all you need to know about typesetting with \LaTeX , but to show you that it's relatively easy to use and that you can produce very attractive, professional-looking results with it with not much effort.

```
\documentstyle[12pt]{letter}

\topmargin=-.5in    \textwidth=6in
\oddsidemargin=.2in \pagestyle{headings}

\begin{document}
\raggedright
```

Figure 7: \LaTeX Letter—Houskeeping

To go beyond the examples I give here and in my follow-on session, read the \LaTeX manual. It's very readable and easy to use, and it's available, like the $\text{T}_{\text{E}}\text{X}$ book, from the $\text{T}_{\text{E}}\text{X}$ Users Group. Its first half consists of a well-written users guide, covering matters of gradually increasing complexity, and its second half is a concise reference manual, suitable for quick lookup of any \LaTeX question.

Figure 7 shows the first few lines of the \LaTeX input file that created the letter in the previous illustration. $\text{T}_{\text{E}}\text{X}$ and \LaTeX commands begin with a “\”, and the first command \LaTeX expects to see is one telling it what kind of document it's going to work on. In this case, we're using the “letter” style, and this command gets from the \LaTeX database the “letter” style file of macro definitions to use in processing our document.

“ \documentstyle ” is a macro command. Any required arguments of macro commands are enclosed in “{ }” braces. Any optional arguments precede the required arguments, and are enclosed in “[]” brackets. The “ \documentstyle ” command can take several optional arguments—for example, when the documentstyle is “article”, the “[twocolumn]” optional argument typesets the document in double-column pages. Here, we've called for a fairly large type size, 12-point, because the letter is short. 11-point is another option, and the default is 10-point.

The next few commands set up the physical position of the typeset area on the page. The “ \topmargin ” and “ \oddsidemargin ” commands adjust the margins of the page a little upward and rightward from the default position. “ $\text{\pagestyle{headings}}$ ” tells \LaTeX to use its standard headings (which consist of the addressee's name and the date) on pages of the letter after the first.

Next we tell \LaTeX we're ready to begin actual document

	DEC 10	DEC 20	VAX (Unix)	VAX (VMS)
Apple LaserWriter			Carleton Univ; Textset†	Textset†
Autologic APS-5/Micro-5		Textset	Textset	Intergraph†; Textset
Canon			Canon	
DEC LN01			Univ of Washington	Louisiana State U
DEC LN03				Procyon; Distrib Tape
Imagen	Stanford; Vanderbilt	SRI; Columbia	Univ of Maryland	Kellerman & Smith†
PostScript printers			Textset	Textset
QMS Lasergrafix			Textset; Univ of Washington	GA Technologies; Texas A&M; Textset
screen preview				Univ of Adelaide
Symbolics		Univ of Washington	Univ of Washington	Univ of Massachusetts
Talaris	Talaris†	Talaris†	Talaris†	Talaris†
Xerox Dover		Carnegie-Mellon U	Stanford	
Xerox 9700	Univ of Delaware		Univ of Delaware	ACC; Textset

Table 1. Selected Output Devices Interfaced to DEC Computers

Information regarding the interfaces shown in this table can be obtained from the individual listed below for the site. This table and the names of the site contacts were provided by the T_EX Users Group.

ACC (Advanced Computer Communications): Diane Cast, 720 Santa Barbara St., Santa Barbara, CA 93101, 805-963-9431

Canon: Masaaki Nagashima, Office Systems Center, 30-2 Shimomaruko 3-chome Ohtaku, Tokyo 146, Japan, (03)758-2111

Carleton University: Neil Holtz, 613-231-7145

Carnegie-Mellon U: Howard Gayle, 412-578-3042

Columbia: Frank da Cruz, 212-280-5126

Distrib Tape: VAX/VMS tape available from Maria Code, 408-735-8006

GA Technologies: Phil Andrews, 619-455-4583

Intergraph: Mike Cunningham, 205-772-2000

Kellerman & Smith: Barry Smith, 503-222-4234

Louisiana State U: Neal Stoltzfus, 504-388-1570

SRI:

Stanford:

Talaris: Sonny Burkett, 619-587-0787

Texas A&M: Bart Childs, 409-845-5470

Textset: Bruce Baker, 313-996-3566

Univ of Delaware: Daniel Grim, 302-451-1990

Univ of Maryland: Chris Torek, 301-454-7690

Univ of Massachusetts: Gary Wallace, 413-545-4296

Univ of Washington: Pierre MacKay, 206-543-2386

Vanderbilt: H. Denson Burnum, 615-322-2357

† graphics supported

Figure 5: Current Output Devices

AMERICAN MATHEMATICAL SOCIETY
Post Office Box 6248, Providence, Rhode Island 02940
{201 Charles Street at Randall Square — (401)272-9500}

Samuel B. Whidden
Director of Computer Services

April 14, 1986

Joe Angelico
U. S. Coast Guard
500 Camp St
New Orleans LA 70130

Dear Joe:

I look forward to working with you again in the DECUS Store at the Anaheim meeting.

I've used \LaTeX to typeset our price list this time. I think you'll agree it's a lot better looking than our old one.

With Best Regards,

Samuel B. Whidden
Director, Computer Services

cc: Judy Arsenault

encl: Price List

Figure 6: A \LaTeX Letter

```

\begin{letter}{Joe Angelico\\
U. S. Coast Guard\\
500 Camp St\\New Orleans LA 70130}

\address{\null}

\begin{center}
{\large AMERICAN MATHEMATICAL SOCIETY}\\
{\bf Post Office Box 6248,
Providence, Rhode Island 02940}\\
{\small \{201 Charles Street
at Randall Square --- (401)272-9500\}}
\end{center}

```

Figure 8: L^AT_EX Letter—Beginning Matter

typesetting and that the right margin need not be justified; if we were to omit the “\raggedright” command, the right margin would be justified and T_EX would automatically hyphenate words where necessary. L^AT_EX is a structured language; there will be an “\end{document}” command later to tell L^AT_EX that processing is complete.

You see three kinds of commands here: dimension-setting commands, like “\textwidth=6in”, that adjust the value of variables like the length a line of type may have; environment-establishing commands like “\pagestyle{headings}”, which give L^AT_EX some of the conditions under which it does its processing; and switches, like “\raggedright”, which turn certain typesetting modes on or off. Among the latter, commands affecting type size and style occur frequently.

These few commands are all we use to tell L^AT_EX how we want our letter physically set up. We wouldn’t even need all of these if we were content with all of L^AT_EX’s defaults.

The commands in Figure 8 begin the material specific to this letter. The command “\begin{letter}” tells L^AT_EX to start collecting the information it needs to typeset this letter. “\begin{letter}” will be matched by “\end{letter}”, which could in turn be followed by another “begin{letter}”.

“\begin{letter}” takes a second required parameter which is the name and address of the recipient, with address lines separated by “\\” commands. L^AT_EX stores this name and address away for future reference.

```

\vspace{3ex}

\parbox{30em}{\scriptsize Samuel B. Whidden\\
Director of Computer Services}

\signature{Samuel B. Whidden\\
Director, Computer Services}

```

Figure 9: L^AT_EX Letter—Personalization Block & Signature

Before it begins writing the letter, L^AT_EX wants to know our return address, to put at the top of the letter. But when L^AT_EX comes to print this return address, it puts it against the right margin near the top of the letter, a default style I didn’t want to use here. The “letter” documentstyle requires an “\address”, so I gave it a “\null” argument to keep it happy, and created my own letterhead in the following commands.

I started a “center” environment, into which I put my office address. For the Society name I switched to the “\large” type size, but limited the scope of that command by enclosing it with the company name in braces. Within the center environment, lines are broken with the “\\” command. I wanted the next line in boldface type, so I enclosed it in braces along with the “\bf” command. I followed that line with one giving our street address and phone number in the “\small” typesize (note that there’s no double backslash after “Street”, so there’s no linebreak there). To get braces to print on this line, I typed “\{” and “\}”. The command “\{” prints a “{”

Another look at Figure 6 shows the result.

In Figure 9, the “\vspace” command moves L^AT_EX down the page by three times the height of a the letter “x”, below the letterhead. I next create a “box” containing my name and title in the size of type normally used for sub- and superscripts. L^AT_EX typesets this box where it finds it. Finally, I tell it what to use for a signature block. In letter style, L^AT_EX recognizes the “\signature” command and stores away the data in the argument for later use.

In Figure 10 we come to the body of the letter. L^AT_EX recognizes the “\opening” command and does several things. First, it digs out the return “\address” it stored away earlier. It has only “\null” for this, since I wanted

```
\opening{Dear Joe:}
```

I look forward to working with you again
in the DECUS Store at the Anaheim meeting.

I've used `\LaTeX` to typeset
our price list this time.
I think you'll agree it's a lot
better looking than our old one.

```
\closing{With Best Regards,}
```

Figure 10: \LaTeX Letter—Body

```
\cc{Judy Arsenault  
}
```

```
\encl{Price List  
}
```

```
\ps
```

```
\end{letter}  
\end{document}
```

Figure 11: \LaTeX Letter—Bottom Matter

to create my own letterhead, so nothing much happens. Then it sets today's date against the righthand margin. Then it retrieves the recipient's name and address block, which it typesets. Finally, it extracts the argument of the "`\opening`" command, in this case, "Dear Joe", and typesets it. All the elements it has remembered, it will continue to remember and use for subsequent letters in this run, if there are any, until you change them by entering them again with new values.

I type in what I want to say to the DECUS Store Manager as ordinary text in paragraph form, and then I end with the "`\closing`" command. \LaTeX recognizes the closing command just as it does the "`\opening`" command. It typesets the argument, then retrieves the signature block it stored away earlier, and typesets that.

In the LETTER context, \LaTeX understands the commands "`\cc`", "`\encl`", and "`\ps`" (see Figure 11), which

```
\storeitem{39}{3-Ring Binder}{8.00}
```

Figure 14: Store Input Sample

we can use if we want to. We tell it we're ending both the letter and the typesetting run, so it finishes up and outputs our letter. We make it do the typesetting by giving the command, at the operating system level, "`\LaTeX joe.tex`", assuming the file that contains our letter is named "joe.tex", and assuming that our systems people have installed \TeX and \LaTeX on our machine. The result will be "joe.dvi", a device-independent file, which must be translated by a driver to drive the output device at our installation.

In that letter, I told Joe I had typeset the Store Price List. The Price List is shown in Figure 12. It's a table with multiple rows, columns, and headers. The first column contains the store's control number and the second the name of the item. The third and fourth columns are blanks to be filled in by the customer, and the last is the price. All the columns repeat on the right hand side of the sheet. Note that in some cases, the name of the item may go to more than one line, and that a header crosses the entire form. Also note that there's a small commercial for \LaTeX tucked away at the bottom.

In fact, it's two tables, both set with the same \LaTeX macros, using different input files. The large table contains most of the items sold in the store, while the smaller one, Figure 13, contains just session notes, which are sold first at the registration counter, and later at the store (the 3-ring session notes binder appears in both lists).

Judy Arsenault, the DECUS staff member responsible for the store, sends me both lists over DCS. I put them in alphabetical order and embed them in \LaTeX commands, as in Figure 14. The command name is "`\storeitem`" and it takes three arguments: the control number, the item name, and the price. The file of all the items is read into a \LaTeX file that contains, among other things, a definition of the "`\storeitem`" macro.

Figure 15 shows the beginning set-up commands from that file. The "article" documentstyle is the standard for most \LaTeX documents. "`\textwidth`" tells \LaTeX to use just about the full width of the 8 1/2 inch paper, and "`\textheight`" says to use more of the page vertically than the default would allow. "`\arrayrulewidth`" sets the thickness of the rules in the table; a 1-point set-

DECUS STORE, Spring Symposium, April 28 - May 2, 1986, Dallas, TX.									
Code	Description	Color, Size, etc.	Qty	Price \$	Code	Description	Color, Size, etc.	Qty	Price \$
39	3-Ring Binder			8.00	18	Ruler			4.00
01	Backpack			7.00	19	Scissors			6.00
02	Bag, Bike			3.00	11	Secretary, Mini (pocket address book)			7.00
05	Bag, Canvas			8.00	21	Shoe Laces			2.00
04	Business Card Case			4.00	34	Stick Pin, DTR SIG			4.00
06	Coffee Warmer Plate			5.00	40	Stick Pin, VAX Chip, L&T SIG			5.00
07	Dart Board			3.00	37	T-Shirt, L&T SIG			9.00
31	Flashlight, (RSX)			5.00	48	T-Shirt, Large Systems			9.00
08	Flippy Flyer			2.00	62	T-Shirt, Library			9.00
50	License Plate Frame, (BA)			3.00	43	T-Shirt, PC SIG, (Adult)			9.00
33	Magnet, DTR SIG			1.00	44	T-Shirt, PC SIG, (Youth)			8.00
09	Markers, Transparency			1.00	55	T-Shirt, Site SIG (Adult)			9.00
12	Money Clip			6.00	56	T-Shirt, Site SIG (Youth)			8.00
03	Mug, Boot			6.00	26	T-Shirt, VAX SIG, (Adult)			9.00
28	Mug, VAX			9.00	27	T-Shirt, VAX SIG, (Youth)			8.00
22	Notebook, Spiral			1.00	65	Tape, DM-111			75.00
46	PC Floppy (DECmate)			15.00	63	Tape, PRO-123			75.00
47	PC Floppy (PRO)			15.00	64	Tape, PRO-124			45.00
45	PC Floppy (Rainbow)			45.00	66	Tape, VAX-150			45.00
13	Pen Set			6.00	41	Template			4.00
20	Pen, Schaeffer			12.00	40	Tie Tack, VAX Chip, L&T SIG			5.00
14	Pen/HiLighter Combination			2.00	29	Tie Tack, VAX SIG			9.00
10	Pencil, Mechanical			1.00	53	Tie, Texas style, (AI)			6.00
15	Portfolio, Large			13.00	23	Transparencies (set of 3)			1.00
16	Portfolio, Small			3.00	24	Travel Alarm w/pen			15.00
68	Proceedings			15.00	69	Utility Keypad Blanks, (L&T)			2.00
17	Refill Pads (8 1/2 x 11)			2.00	38	Utility Keypad Layouts, (L&T)			5.00

Table by L*TeX

April 14, 1986

Figure 12: The Store Price List

Session Notes

DECUS STORE, Spring Symposium, April 28 – May 2, 1986, Dallas, TX.

Code	Description	Color, Size, etc.	Qty	Price \$	Code	Description	Color, Size, etc.	Qty	Price \$
39	3-Ring Binder			8.00	35	Networks			13.00
50	Artificial Intelligence			11.00	59	Office Automation			15.00
47	Business Applications			15.00	42	Personal Computer			8.00
49	Commercial Languages			11.00	60	Refereed Papers Journal			8.00
56	DAARC			9.00	30	RSX			11.00
58	Data Management			15.00	52	Site Management			8.00
32	DATATRIEVE/4GL			10.00	57	UNISIG			15.00
55	Graphics Applications			8.00	25	VAX Systems			15.00
36	Languages & Tools			13.00					

Table by L^AT_EX

April 14, 1986

Figure 13: The Session Notes Price List

```

\documentstyle{article}

\textwidth=8in      \textheight=9.5in

\topmargin=-1in    \oddsidemargin=-.7in

\arrayrulewidth=1pt

\renewcommand{\arraystretch}{1.5}

\count20=0          \pagestyle{empty}

```

Figure 15: Store List—Houskeeping

```

\newcommand{\storeitem}[3]{#1&\parbox{1.7in}%
{\raggedright#2}&&&#3%

\ifnum\count20=0\global\count20=1%

\newcommand{\finish}{&}\else\global\count20=0%

\newcommand{\finish}{\\hline}\fi\finish}

```

Figure 16: The Store Item Input Command

ting (about 1/72nd of an inch), makes them nice and dark.

“`\arraystretch`” governs the vertical distance between rows of the table. That command has already been set by \LaTeX , but we can “renew” the setting by using the “`\renewcommand`” command, and here we tell \LaTeX to make that vertical distance one and a half times what it otherwise would.

We want “`\pagestyle`” to be empty—we don’t need page numbers or running heads. We’re going to use one of \TeX ’s already-defined internal registers, “`\count20`”, as an on-off switch in our macro definition, so here in the “preamble” we initialize it to zero.

The next thing we come to in the file is the definition of the “`\storeitem`” macro, Figure 16. It’s just defined at this point, not used. It will be used later when it is invoked by the “`\storeitem`” command while it is building

our table.

“`\newcommand`” tells \LaTeX we’re defining a new command. The first argument gives the new command’s name. The next argument is an optional one. It tells whether our new command will have any required arguments, and, if so, how many. Recall that there will be three arguments to “`\storeitem`”: the control number, the item name, and the price (Figure 14).

The second required argument of “`\newcommand`” is the string which defines the new command. For “`\storeitem`”, the definition begins with the # symbol, which tells \LaTeX to substitute here the first argument given in the call, in this case the control number, 39. So, when “`\storeitem`” is called during table construction, the control number will be set in the first column of the table (we define the table itself later).

The next character in the defining string is &, which says “skip to the next column in the table”. For what to put in that next column, \LaTeX finds that we want to construct a “paragraph box”, that is, a “box” of text, in this case 1.7 inches wide, the width of the item-name column of the table, within which text is set in paragraph mode, with automatic line breaking. If an item name is shorter than 1.7 inches, as most of them are, it will simply be set as is. If it’s longer, \LaTeX automatically breaks it into multiple lines, increasing the height of that row in the table. The contents of the “`\parbox`” are specified by the #2, which tells \LaTeX to substitute here the second argument in the calling string, namely the item name. “`\raggedright`” tells \LaTeX the item name needn’t be justified and hyphenated at the right margin.

\TeX uses the % as a comment-character—anything following a % on a line is ignored, including the “`return`”. Returns are normally converted to spaces by \TeX , and spaces inside macro definitions can have unfortunate effects. It’s usually best to comment out the return with a %, except in circumstances where you know a space will just be thrown away (as following a command word like “`\line`”).

Now we come to three column-skip symbols, getting us past the two blank table columns and into the price column. The #3 simply picks up the third argument in the calling string, the price, and dumps it here. And the macro definition is complete.

Almost.

Recall that the table has a right side and a left side. Those two sides differ only in what \LaTeX needs to do

```

\newcommand{\storehead}{
\multicolumn{1}{c|}{Description}&
\multicolumn{1}{c|}{\parbox{35pt}%
{Color, \Size, etc.}}&
\multicolumn{1}{c|}{Qty}&
\multicolumn{1}{c|}{\parbox{30pt}%
{Price\\\$}}

```

Figure 17: Partial Headers

to finish off the macro, after inserting the price. If it's working on the left side, it must finish by crossing over to the right side so as to be in position to start the next item. If it's working on the right side, it must end by dropping down a row, drawing a horizontal line across the table, and positioning itself at the left margin to be ready for the next item.

\TeX provides some conditional statements for testing conditions like this, one of which is used here. It begins with “\ifnum”, the start of a numeric test. The end-if statement in \TeX is—what else?—“\fi”. You'll find it toward the end of the macro definition. Everything between “\ifnum” and “\fi” is a conditional test which defines one of two versions, separated by “\else”, of the macro “\finish”. Which version of “\finish” is defined, the left side version or the right side version, depends on the setting of the switch “\count20”, which the test itself toggles back and forth. Once “\finish” is properly defined, it's called and executed and the call to “\storeitem” is done. The left-side definition of “\finish” contains only a column-skip symbol (&) (plus a call to the “\rule” macro, which is here just a device to insure sufficient vertical separation between rows, and may not even really be necessary). The right hand version of “\finish” issues the double-backslash linebreak command, plus the “\hline” command to draw a horizontal line under the row. The switch, “\count20” must be set “\global”ly so its value will persist after the call to “\storeitem” is closed.

in Figure 17, another command is defined. You'll recognize part of its content as most of the column headers from the table: “Description”, “Color, Size, etc.”, “Qty”, and “Price”. They appear on both halves of the table, so

```

\newcommand{\storetable}[2]{
\begin{table}
\begin{center}{\Huge #2}\vspace{.5in}\end{center}
\small
\begin{tabular}{|r|l|c|c|r|l|c|c|r|}| \hline
\multicolumn{10}{|c|}{\Large DECUS STORE,
Spring Symposium,
April 28 -- May 2, 1986, Dallas, TX.}%
\rule[-.9em]{Opt}{2.9em} \\\hline
\multicolumn{1}{|c|}{\rule[-1.2em]{Opt}{3em}Code}&
\storehead&
\multicolumn{1}{c|}{Code}&
\storehead \\\hline
\input{#1.txt}
\end{tabular}
\par\hspace*{5em}{\tiny Table byL\hspace{-.1em}}%
\raisebox{1ex}[2ex]
{a}\TeX\hfill \today \hspace*{8em}\null}
\end{table}}

```

Figure 18: The Store Table

I defined them separately. We'll see how this command, “\storehead” is used, and what its definition means, when we examine the actual table definition.

Figure 18 gives the definition of the table itself. The definition is put into a macro, called “\storetable”, so it can be invoked once for the main store list, and again for the session notes list. Note that the definition of “\storetable” calls for two arguments to be expected in the calling string. The first argument will govern which table is being set. The second is a title phrase which can be used to identify the table.

The table definition is set within a “table” environment, delimited by “\begin{table}” and “\end{table}”. The principal effect of this environment is to allow \LaTeX to choose an appropriate page, or position within a page, at which to set it.

To set the title above the table, we establish a “center” environment (“\begin{center}... \end{center}”), within which we typeset the title (argument #2) in \LaTeX 's “\huge” typesize, followed by a half inch of vertical space. The scope of “\huge” is delimited by surrounding braces.

Next we set “\small” as \LaTeX 's default type size inside the “table” environment.

The significant environment for our purposes is the “`tabular`” environment inside the “`\begin{tabular}`” and “`\end{tabular}`” statements. The second argument of the “`\begin{tabular}`” command is a series of characters that describe to \LaTeX the columns in the table. Each letter “`r`”, “`l`”, or “`c`” identifies a column within which text is to be set right or left justified, or centered. Between these code letters are vertical bars, telling \LaTeX whether to insert a single vertical line between columns, a double line, or none. There are 10 columns in our table, 5 on the left half and five on the right. Single vertical lines separate all of them except for the center pair of columns and the right and left hand edges of the table.

That’s the definition of our table. Now all we have to do is fill it in.

The first command after the definition is “`\hline`”, giving us the topmost horizontal line. Next we want a header to cross all the columns. We obtain this using the “`\multicolumn`” command, the first argument of which is the number of columns we want to span (here all 10), the second the column-formatting code for the multiple column (here, centered text with double vertical lines at each border of the table), and the third the text to be set. After the header and before we go to a new line, we insert a vertical rule to cause some extra white space to be set above and below the header text. The first argument of the “`\rule`” command tells how far below the base of the current line our rule extends. The second gives the rule’s width (this rule will be invisible because we specify 0 width), and the third tells how far above the baseline it extends. We adjust these dimensions by trial and error, depending on what we find pleasing.

We follow this by the usual double-backslash command for breaking lines inside special environments, and another horizontal line. And that takes care of our major heading.

Next we have to establish our individual column headers. Again we use `\multicolumn` to allow special formatting within the column, but this time we need to span only a single column. We format it with a double vertical column on the left (because this is the left edge of the table), and centered text. The heading itself is “`Code`”, standing for “`Control Number`”. Once again, the “`\rule`” command is used to provide a minimum height for the row of column headers. Then comes an `&` column-skip command and a call to the “`\storehead`” macro of Figure 17.

Each of the remaining four column headers is represented the same way in both sides of the table, so they can be

```

\begin{document}
\storetable{store}{\ }

\clearpage\newpage

\storetable{sessnotes}{Session Notes}
\end{document}

```

Figure 19: Using the Store Table

collected in this macro. Each spans a single column with centered text and a vertical line at its right edge, except for “`Price`”, which has a double vertical. The command “`\$`” typesets a \$ on a line below “`Price`”. The “`Size, Color...`” header is set in paragraph mode to allow \LaTeX to break gracefully into more than one line. Each column is followed by an `&` except for the last, since there’s no next column to skip to if we’re setting the right hand side of the table.

The first time we call “`\storehead`” we put an `&` after it to get us from the last column in the left half of the table to the first of the right half. Then we describe the header for the right hand “`Code`” column, with slightly different format codes than its left hand counterpart, followed by a column-advance, followed by the second call to “`\storehead`”. This time we don’t want a column-advance after “`\storehead`”, but we do want a linebreak command and another horizontal line. Now we’ve established all the headers. It only remains to get the actual store items into the table.

We do that with the “`\input`” command. This tells \LaTeX to read a file whose name is given by its argument, here “`#1.txt`”, and input the contents here. We’ll see how this command gets the input for both our tables.

After the “`tabular`” environment ends, the phrase “`Table` by \LaTeX ” is set in the smallest, “`\tiny`”, of \LaTeX ’s type sizes. No “`\LaTeX`” macro has been defined at that typesize, so we have to set the word \LaTeX properly, with all its case shifts and position changes, ourselves.

Finally, we’re ready to begin. Everything’s defined and ready to go, it just has to be called. We tell \LaTeX to get ready to begin outputting and then (Figure 19), we issue a call to “`\storetable`”. This call causes \LaTeX to begin processing “`\storetable`”, defining the table’s columns, setting its headers and reading and processing an input file. Recall that the first argument of

```

\documentstyle{article}
\textwidth=5in \textheight=8in
\topmargin=-1in \oddsidemargin=-.7in
\parskip=3in \parindent=0in
\fbboxrule=1pt \count20=0
\pagestyle{empty}

```

Figure 22: Labels Macros—Housekeeping

the “`\storetable`” call determines the input file that will be read by “`\storetable`”. The argument “store” means that the “`\input`” command will read the file “`store.txt`”.

Figure 20 shows the Store input file, filled with calls to “`\storeitem`”, which will load the table with data on individual store items, gradually building the table downward until the end of the input file is reached, followed by the end of the “`tabular`” environment.

After the “`\storetable`” macro has finished processing the “`store.txt`” input file, the “`\clearpage`” and “`\newpage`” commands are encountered, finishing and ejecting the page, and then “`\storetable`” is called again, this time to process the “`sessnotes.txt`” input file for the second table.

The “`sessnotes.txt`” input file (Figure 21) is formatted identically to the “`store.txt`” input file.

My last example is short. We’ve just seen one \LaTeX macro file produce output from two separate input files. In this example, those same two input files are processed by another \LaTeX macro file to produce entirely different output. In Figure 22 we see some of the same set-up commands we’ve used before. New features here are a setting of “`\parskip`”, the vertical distance between paragraphs, to 3 inches. We’re going to set things that are far apart vertically on the page. “`\fbboxrule`” does a similar thing here to the job “`\arrayrulewidth`” did in the last example; it determines the heaviness of rules.

And Figure 23 presents a redefinition of the “`\storeitem`” macro. It will still take three arguments, and they’ll be the same three arguments as before, because this version of “`\storeitem`” will still serve the same two input files.

This time, though, #1, #2, and #3 will go inside a 3.5-

```

\newcommand{\storeitem}[3]{\LARGE
\framebox{\parbox{3.5in}%
{\raggedright Code: #1\
\centering#2\
\raggedleft\$\#3}}}%
\ifnum\count20=0%
\hspace{.375in}\count20=1%
\else \par\count20=0\fi}

```

Figure 23: A New Definition of “STOREITEM”

```

\begin{document}
\input{store.txt}
\input{sessnotes.txt}
\end{document}

```

Figure 24: Setting the Labels

inch wide “`\parbox`” that, in turn, is inside a “`\framebox`” (a box with a visible frame around it). The first thing into the box, “`\raggedright`” (left-justified), is the word “Code”, followed by #1—which, as you’ll recall from Figure 14, is the control number for an item. There’s a line break, a “`\centering`” command, and #2, the item name. Next, after another linebreak, is a \$ and #3, the price, set “`\raggedleft`” (right-justified).

Finally, the macro ends with a conditional that inserts a horizontal space of .375 inches if the switch says we’re on the left hand side of the page or sets a paragraph break (which we previously set equal to three vertical inches) otherwise.

Figure 24 shows the complete set of commands which use the new definition of “`\storeitem`”. As before, we input “`store.txt`”, followed by “`sessnotes.txt`”, with their many calls to “`\storeitem`”.

And the result, Figure 25, is just what you expected. One small sign for each item, one large plus for the DECUS store. This is a good example of the power of \TeX and \LaTeX , invoked in a reasonably straightforward and not-too-difficult way, using and reusing data through simple redefinition of macros.

In my next session, I’ll describe the \LaTeX macros that typeset the Symposium Sessions-at-a-glance.

Figure 20: The Store Input file

```

\storeitem{39}{3-Ring Binder}{9.00}
\storeitem{01}{Backpack}{8.00}
\storeitem{02}{Bag, Bike}{4.00}
\storeitem{05}{Bag, Canvas Tote}{8.00}
\storeitem{21}{Bag, Sport}{10.00}
\storeitem{03}{Bookmark}{1.00}
\storeitem{04}{Business Card Case}{5.00}
\storeitem{06}{Coffee Warmer Plate}{6.00}
\storeitem{07}{Dart Board}{6.00}
\storeitem{33}{Flashlight (RSX SIG)}{5.00}
\storeitem{08}{Flippy Flyer (frisbee)}{3.00}
\storeitem{48}{License Plate Frame (BA SIC)}{3.00}
\storeitem{09}{Luggage Tag}{2.00}
\storeitem{35}{Magnet, DTR SIG}{1.00}
\storeitem{10}{Markers}{1.00}
\storeitem{32}{Microfiche (RSX)}{3.00}
\storeitem{13}{Money Clip}{6.00}
\storeitem{28}{Mug, VAX}{9.00}
\storeitem{20}{Notebook, Spiral}{1.00}
\storeitem{65}{PC Floppy, Basic 8}{25.00}
\storeitem{66}{PC Floppy, PC0001}{5.00}
\storeitem{67}{PC Floppy, PC0002}{5.00}
\storeitem{68}{PC Floppy, PC0003}{5.00}
\storeitem{69}{PC Floppy, PC0004}{5.00}
\storeitem{70}{PC Floppy, PC0005}{5.00}
\storeitem{71}{PC Floppy, PC0006}{5.00}
\storeitem{72}{PC Floppy, PC0007}{5.00}
\storeitem{73}{PC Floppy, PC0008}{5.00}
\storeitem{74}{PC Disk}{5.00}
\storeitem{14}{Pen Set}{10.00}
\storeitem{15}{Pen/Hi-Lighter Combination}{2.00}
\storeitem{11}{Pencil, Mechanical}{1.00}
\storeitem{16}{Portfolio}{5.00}
\storeitem{17}{Refill Pads (8 1/2 x 11)}{2.00}
\storeitem{18}{Ruler}{5.00}
\storeitem{19}{Scissors}{8.00}
\storeitem{12}{Secretary, Mini (pocket address book)}{12.00}
\storeitem{36}{Stick Pin, DTR SIG}{4.00}
\storeitem{22}{Sweater}{10.00}
\storeitem{50}{T-Shirt, CL SIG, Adult}{8.00}
\storeitem{52}{T-Shirt, CL SIG, Toddler}{8.00}
\storeitem{51}{T-Shirt, CL SIG, Youth}{8.00}
\storeitem{40}{T-Shirt, L\T SIG, (new)}{9.00}
\storeitem{41}{T-Shirt, L\&T SIG, (old)}{8.00}
\storeitem{46}{T-Shirt, Large Systems SIG}{9.00}
\storeitem{44}{T-Shirt, PC SIG, Adult}{9.00}
\storeitem{45}{T-Shirt, PC SIG, Youth}{8.00}
\storeitem{30}{T-Shirt, RSX, Adult}{8.00}
\storeitem{31}{T-Shirt, RSX, Youth}{8.00}
\storeitem{26}{T-Shirt, VAX SIG, Adult}{9.00}
\storeitem{27}{T-Shirt, VAX SIG, Youth}{8.00}
\storeitem{62}{Tape, OA SIG, V-SP-44}{76.00}
\storeitem{63}{Tape, RSX-11 SIG, 11-SP-84}{154.00}
\storeitem{64}{Tape, RT-11 SIG, 11-SP-83}{154.00}
\storeitem{61}{Tape, VAX SIG, V-SP-46}{154.00}
\storeitem{42}{Template, L\&T SIG}{4.00}
\storeitem{23}{Transparencies (set of 3)}{1.00}
\storeitem{24}{Travel Alarm w/pen}{19.00}
\storeitem{60}{Visor, LUG}{2.00}
\storeitem{ }{ }{ }

```

Figure 11 — Store Input File

Figure 21: Session Notes Input

```
\storeitem{39}{3-Ring Binder}{8.00}
\storeitem{57}{Session Notes, AI SIG}{13.00}
\storeitem{47}{Session Notes, BA SIG}{15.00}
\storeitem{49}{Session Notes, CL SIG}{13.00}
\storeitem{56}{Session Notes, DAARC SIG}{9.00}
\storeitem{59}{Session Notes, DMS SIG}{16.00}
\storeitem{34}{Session Notes, DTR SIG}{10.00}
\storeitem{53}{Session Notes, Graphics SIG}{9.00}
\storeitem{38}{Session Notes, L&T SIG}{13.00}
\storeitem{37}{Session Notes, Networks SIG}{9.00}
\storeitem{55}{Session Notes, OA SIG}{9.00}
\storeitem{43}{Session Notes, PC SIG}{8.00}
\storeitem{29}{Session Notes, RSX SIG}{11.00}
\storeitem{54}{Session Notes, Site Mgmt}{9.00}
\storeitem{58}{Session Notes, UNISIG}{16.00}
\storeitem{25}{Session Notes, VAX SIG}{15.00}
```

Figure 12 — Session Notes Input File

Code: 39
3-Ring Binder
\$8.00

Code: 18
Ruler
\$4.00

Code: 01
Backpack
\$7.00

Code: 19
Scissors
\$6.00

Figure 25: Store Item Labels

L^AT_EX Examples

Samuel B. Whidden
J. R. Westmoreland

Abstract

This is the second of two sessions in which examples of L^AT_EX usage are given. The authors hope to demonstrate that L^AT_EX, a document formatter based on the T_EX typesetting system, is both powerful and easy to use. This session presents examples somewhat more advanced than those of the previous session.

In my previous session, I described what T_EX and L^AT_EX are and gave some examples of how they work. In this session, I will continue with two more examples. All T_EX and L^AT_EX commands are described in detail in *The T_EXbook* and *The L^AT_EX Manual* both published by Addison-Wesley and available from the T_EX Users Group (see the last slide in this article). Those books should be consulted by anyone who seriously wants to use T_EX or L^AT_EX.

Manual typesetting of the Symposium Sessions-At-A-Glance (SAG) traditionally required several weeks and was a factor affecting the timeliness with which DE-CUS members received their Preliminary Programs in advance of the Symposium. Last Fall, in Anaheim, Jeff Jalbert, Chairman of the Symposium Committee, and Ned Rhodes, Chairman of its New Technology subcommittee, accepted my offer to see whether the process could be speeded up using L^AT_EX to typeset the SAG. The project was a success, and the SAG in your Preliminary Program for this Symposium was typeset that way. This session will illustrate the use of L^AT_EX by describing the macros that accomplished it.

J. R. Westmoreland, the Languages and Tools SIG's T_EX Coordinator, is participating in this session with me to help answer any difficult questions you may ask. One of the dangers, and rewards, of discussing your work in public is that there's always someone in the audience more experienced and skillful who can, and usually does,

point out all your silly mistakes. In this case, that will be all to the good, since that person will immediately get the job of producing the SAG from now on.

Before we examine the SAG macros, I'll show you a few examples of L^AT_EX usage taken from the SAG project's documentation. These are interesting if you are just starting to use L^AT_EX and you want to explore some of its possibilities. The documentation is a dozen pages long, of which I've included the first four as examples.

The documentation file starts with the usual housekeeping chores (Figure 5), setting up the size and position of the print area on the page, the document style (here, "article"; this tells L^AT_EX which of its various sets of macro definitions apply to this job), and paragraph characteristics.

Setting the "secnumdepth" counter to zero is a way of telling L^AT_EX not to number sections, even though we plan to use the "\section" and "\subsection commands. The command "\verb+" tells L^AT_EX to use its default heading style for the "article" documentstyle, which is to print the name of the current section, as given by the most recent "\section" command, in the running head on each page.

In the "article" documentstyle, L^AT_EX recognizes the "\title", "\author", and "\date" commands (Figure 6), storing the associated data away for later use. When

SAG Input Documentation

Sam Whidden
Languages and Tools SIG

May 23, 1986

General

SAG is typeset by \LaTeX , a dialect of the \TeX typesetting language. Each day of the Symposium is normally set on two pages, a left- and a right-hand page (but sometimes each of these has to be set on two pages, when there are too many rooms to fit on one page). Input files are created in a format described below and placed in Sam Whidden's DCS directory. Sam runs them through \LaTeX in his office at the American Mathematical Society in Providence, RI and returns them by mail to the DECUS office. It is expected that eventually the DECUS office will be able to install \TeX on one of its own machines so that the job can be accomplished entirely in-house.

Input files are generated in the DECUS Office as a result of the logging process at the end of symposium scheduling activity. They are created by the program which also generates a line-printer version of the SAG. \LaTeX lacks sufficient memory capacity to process an entire left- or right-hand page in a single pass, so the files are divided into segments as described below.

File Conventions

File Names

In cases where ten rooms or less are being composed into a SAG, the lefthand page is named 'DAY'.lxx and righthand page .rxx. (This might happen, for instance, if the sessions sponsored by one SIG are typeset into a "mini-SAG" for the convenience of its members.)

If there are more than ten rooms on a page, the page must be split into several files, each containing up to 10 rooms. The file containing the topmost rooms on the lefthand page is named 'DAY'.ltx. The file containing the bottom rooms on that page is named .lxx. If there are more than 20 rooms, a third file, named .lxx, is created containing the rooms between the first two files. In the very rare case of a

Figure 1: SAG Documentation—Page 1

symposium with more than 30 rooms, there will be more than one center section; instead of one .lcx file, there will be an .lc1 and an .lc2 (as well as an .rc1 and an rc2).

For example, MONDAY.LBX is the last file containing data for the lefthand Monday page. WEDNESDAY.RCX contains the middle group of rooms on the righthand Wednesday page. There should always be a "top" and a "bottom" (or else a "full") page file; In addition, there may be zero or more "center" files, as needed.

These file names are for convenience, and are used only for human reference. L^AT_EX relies on arguments to the `\saagpage` macro, discussed below, to control treatment of the various page segments. Nevertheless, observing the naming conventions helps avoid confusion and error. Typeset SAG pages are stripped together in the DECUS office from the partial pages produced from these input files.

Room Order

The order of the rooms in the input files should be checked with the Symposium Coordinator to ensure correctness.

File Beginning and End Identifiers

Each input file starts with the following line:

```
\saagpage{name of day}{page identifier, from the list below}{fall or
spring}{year}
```

page identifiers are:

- o leftfullpage
- o rightfullpage

or

- o lefttoppage
- o leftcenterpage
- o leftbotpage
- o righttoppage
- o rightcenterpage
- o rightbotpage

Here's a sample:

```
\saagpage{monday}{lefttoppage}{fall}{1985}
```

Each input file should end with the line:

Figure 2: SAG Documentation—Page 2


```
\endsaagpage<CR>
```

Note: The left- or right top- or bot- or full-page arguments cause the corresponding page headers or footers (the "clocks") to be typeset above or below the list of rooms and sessions, and must be used where appropriate. To insure proper spacing, the 'center' arguments, which are used only for groups of rooms appearing at neither the top nor the bottom of the page, also invoke a short form of the headers which is stripped out during the final paste-up process. Appendix I shows output corresponding to each of these arguments.

File Contents

Between `\saagpage` and `\endsaagpage`, the file contains data on rooms and the sessions held in them. Input for each room begins with the line

```
\room{name of room}
```

This is followed by lines describing that room's sessions, in the form

```
\session{no of periods spanned}{session number}{session name}
```

Each SAG page contains 28 quarter-hour periods, and a session may span any number of these time slots. (The typeset headers show time in half-hour increments in the form

10:00
10:30

, with extra tick marks indicating quarter-hour intervals).

The first argument of `\session` is this number of periods (this is the sole argument to `\opentime`, which is used instead of `\session` to cover times when the room is unused). The value of this argument must always be an integer. For each room, the total number of periods given in `\session` and `\opentime` lines must equal exactly 28.

The other two arguments of `\session` are the session number, beginning with capital letters denoting the sponsoring SIG, and the session name, which must be short enough to fit in the table cell.

Example:

```
\room{Embassy West}
\session{2}{RX056}{RSX DOCUMENTATION OVERVIEW}
\session{2}{RX030}{DECNET TASK-TO-TASK LINK}
\session{4}{RX010}{DECSECS}
\session{2}{RX059}{SUPER LIGHTS}
\session{2}{RX006}{VMS TO RSX MIGRATION}
\opentime{6}
\session{4}{RX011}{JUST A MODEST RSX PROPOSAL}
\session{6}{RX004}{RSX/POS Q \& A SESSION}
```

Figure 3: SAG Documentation—Page 3

For the appearance of this input as output, see Appendix I. The table captions appearing in Appendix I appear on the stripping galleys themselves to aid in identifying the various page segments.

Appendix II shows samples of actual input files, and Appendix III shows the full-page stripped output obtained from these input files.

Line Breaking—Making Titles Fit

T_EX generally hyphenates words automatically to make lines fit within the space allotted to the session title. T_EX doesn't normally try to hyphenate proper names (defined, as far as T_EX is concerned, as any word containing an uppercase letter.) But, since session titles appear in all uppercase, that feature is suppressed for SAG processing, and T_EX will try to hyphenate every word.

Even so, a session title may contain a phrase or a word unfamiliar to T_EX, which it can't hyphenate. And T_EX won't try to break a word that already has a hyphen or a slash in it, except at the hyphen or the slash. One way to tell T_EX where it may hyphenate a word is to insert a \- (a "discretionary" hyphen) wherever hyphenation would be acceptable: for example, "hy\-phen\-ate". The \-'s will disappear in the output, but T_EX will know it can hyphenate the word at those places. (But remember, if you do that, to insert discretionary hyphens at *all* the acceptable points in the word.)

Sometimes it is necessary to rephrase a title to allow it to fit into its box, especially when the box is small. A run-together phrase can be broken into separate words so T_EX will break the lines at the spaces. In extreme cases, a word or phrase may be entered in lower case, which is considerably more compact than the uppercase font used normally. While these circumstances may occur occasionally with half-hour sessions, they are rarely encountered in longer ones.

To keep a title (or a room name) from being broken at a particular space, replace the space with a ~ (tilde). For example,

Rooms 6 & 8

 may look odd. Typing "Rooms 6~\&~8" instead gives

Rooms 6 & 8

But don't put tildes where you don't have to; let T_EX have as much freedom to format the lines as you can.

Quotes

To get a left (or right) double quote, type two left (or right) single quotes.

Proofreading and Corrections

As with any computerized operation requiring care and quality control, the SAG must be examined closely for typesetting flaws or other errors. When the SAG output has been received and proofread in the DECUS office, corrections to the input files may be made as required and those files containing corrections retransmitted to the AMS for retypesetting.

Figure 4: SAG Documentation—Page 4

```

\documentstyle[12pt]{article}
\pagestyle{headings}          % use running headers

\topmargin=-0.25in           % move top margin up some
\oddsidemargin=-0in          % decrease the left margin

\textheight=9.0in            % make the page longer
\textwidth=6.5in              % make the text wider

\parindent=0pt                % use block-format paragraphs
\parskip=0.5\baselineskip     % with half a line between them

\setcounter{secnumdepth}{0}   % don't number sections

```

Figure 5: SAG Documentation—Houskeeping

```

\section{File Conventions}

\subsection{File Names}

```

Figure 7: SAG Documentation—Sections

```

\title{SAG Input Documentation}

\author{Sam Whidden\\
\vspace{.5in}
Languages and Tools SIG}

\date{\today}

\begin{document}
\sff

\maketitle

```

Figure 6: SAG Documentation—Title Group

it encounters the “`\maketitle`” command, after we’ve told it, by saying “`\begin{document}`”, to get ready to start outputting, it fetches the data associated with “`\title`”, “`\author`”, and “`\date`”, and uses them to make a title page. The “`\sf`” command tells \LaTeX to use a sans serif font as the text font in the body of the document. Figure 1 shows the title page it makes, using the data I entered with “`\title`”, “`\author`”, and “`\date`”.

I’ll show only selected \LaTeX commands from the documentation file. The two in Figure 7, just as you might imagine, enter a section name and a subsection name. See the result in Figure 1. \LaTeX automatically selects a type size and begins a new section and subsection. They aren’t numbered because we said not to number them. Figure 2 shows that \LaTeX picks up the section name to use as a running head.

Figure 2 also contains examples of itemized lists, a common construct in many kinds of documents. There are two lists, a 2-element list and a 6-element one, separated by text consisting of the word “or”.

```

\newcommand{\bmylist}{\begin{list}{$\circ$}{
  \topsep=Opt
  \partopsep=Opt
  \parsep=Opt
  \itemsep=Opt}}
\newcommand{\emylist}{\end{list}}
\vspace{0.5\baselineskip}}

page identifiers are:
\bmylist
\item leftfullpage
\item rightfullpage
\emylist

\hspace{1.5in}or

\bmylist
\item lefttoppage
\item leftcenterpage
\item leftbotpage
\item righttoppage
\item rightcenterpage
\item rightbotpage
\emylist

```

Figure 8: SAG Documentation—Lists

The input for this example, Figure 8, shows both the built-in power of \LaTeX and the ease with which its formats can be changed to suit individual requirements and tastes. \LaTeX has several “list” environments, all of which provide automatic spacing and indenting of your lists, nested up to four levels deep with correct formatting. If you say “ $\text{\begin{enumerate}}$ ”, items in your list are given roman or arabic numbers, depending on the nesting level. If you say “ $\text{\begin{itemize}}$ ”, items are presented with bullets or tick marks, again appropriate to the nesting level.

You identify the start of each item in the list with an “ \item ” command. If you give an argument to the “ \item ” command, that argument replaces the default label for that item. Thus, you can make lists in which each item has its own heading.

There’s also a general list environment which you can use or modify to suit your own taste. I can say “ $\text{\begin{list}}$ ”, followed by an argument that gives the default label, followed by a series of settings for other parameters, such as indentation width and the between labels and items or the vertical separation between items or at the top and bottom of the list, and a number of others. The \LaTeX manual concisely sets forth the complete list of parameters. You can see that this example establishes a default label consisting of the mathematical symbol “ \circ ”, plus a number of altered parameter values.

Now, instead of simply using this tailored environment, we can package it for use whenever we want! We make the whole thing be the defining string of a macro of our own. “ \newcommand ” is the command that establishes a new macro. Its first argument is the name of the new macro, and its second is the new macro’s definition. Here, we give our new macro the name “ \bmylist ” (for “begin my list”) and give as its definition our specialized list environment.

Environments in \LaTeX are structured, in the sense that they always begin and end with unique delimiters, such as “ $\text{\begin{list}}$ ” and “ $\text{\end{list}}$ ”. Our new macro definition establishes a new beginning delimiter, “ $\text{\begin{mylist}}$ ”. For consistency, we also name an ending delimiter for our special list environment, “ \emylist ”, and we toss in an extra vertical skip equal to half the current vertical distance between text lines.

Now when we type “ \bmylist ”, our list environment is invoked, and we create a 2-item list. After it, we shift right an inch and a half from the margin and type the word “or”. Then we use our list environment again for a

```

{\subsubsection{Example:}

\begin{verbatim}
\room{Embassy West}
\session{1}{RX056}{RSX DOCUMENTATION OVERVIEW}
\session{1}{RX030}{DECNET TASK-TO-TASK LINK}
\session{2}{RX010}{DECSECS}
\session{1}{RX059}{SUPER LIGHTS}
\session{1}{RX006}{VMS TO RSX MIGRATION}
\opentime{3}
\session{2}{RX011}{JUST A MODEST RSX PROPOSAL}
\session{3}{RX004}{RSX/POS Q \& A SESSION}

\end{verbatim}

```

Figure 9: SAG Documentation—The Verbatim Environment

Between `\verb+saagpage+ and endsaagpage`, the file contains data on rooms and the sessions

Figure 10: SAG Documentation—In-line Verbatim

slightly longer list. \LaTeX produces the result shown in Figure 2.

At the bottom of Figure 3 is a series of \LaTeX commands output as text, under the heading “Example”. Figure 9 shows all we need to do to produce it. First, we start a “`\subsubsection`” called “Example”. That automatically gives us our header.

Then we “`\begin{verbatim}`”, copy in our example from an input file, and “`\end{verbatim}`”. (Verbatim output is always set in the “`typewriter`” font.)

In the paragraphs just above this example in Figure 3 are some instances of \LaTeX commands output as part of the text. Such cases, where just a command or two is typeset, are handled as shown in Figure 10. For input, all that’s necessary is to type the “`\verb`” command in line with the rest of the text, followed by any character not used in the command, followed by the command(s), followed by the same delimiter character (here, the plus sign). The “verbatim” environment establishes a full set of style parameters. As with many \LaTeX environments,

there’s a switch command, in this case “`\verb`”, that (temporarily) turns on some of the characteristics of the environment, sufficient for a brief use.

The text in Figure 11 is reproduced from Figure 4. It is one last example of in-line verbatim text. It also contains some information about \LaTeX ’s hyphenating ability.

The input for that text is given in Figure 12. It contains a couple of instances of the “`\verb`” command. It also uses the command that writes the \TeX logo. Command words must end with a non-alphabetic character. Space is such a character, and \TeX recognizes it as a delimiter. But \TeX also swallows all spaces after command words, and won’t output them. So we insert a “command space”, “`\`”, which \TeX does pass along to the output, otherwise we’d get no space between the \TeX logo and the word following it.

My last example from the documentation are the in-line frameboxes shown in Figure 13. Input for them appears in Figure 14. We begin with a “`\parbox`”, which is a paragraph-mode environment, inside of which most style parameters can be adjusted. The “`\parbox`”’s first argument is its width, which we make three-quarters of an inch. Its second argument is the text to be set inside it, in this case “Rooms 6 & 8”. \LaTeX has a “centering” environment which is invoked by “`\begin{center}`” and “`\end{center}`”. Like many \LaTeX environments, this one has an equivalent switch command (the \LaTeX manual calls them “declarations”) for in-line use. The switch in this case is “`\centering`”.

Then we put an “`\fbox`” (a framebox) around the whole thing to make the box visible, and we’re done. This example also contains a description of \TeX ’s use of the “`~`” (tilde) to produce a space in the output at which \LaTeX won’t break the line.

Now we come to the SAG itself. The macros that create the SAGs contain a wealth of examples of things that work that you can copy or modify if and when you’re faced with similar tasks. DECUS Symposium Preliminary Programs show the full range of pages and circumstances that the SAG accommodates. Figure 15 illustrates the first SAG page—the left hand page of the two-page spread representing the first day’s sessions at a (mostly fictitious) Symposium. There’s a header spanning all the columns on the page, headers assigning half-hour time periods to each column, top and bottom, along with quarter-hour tick marks top and bottom, and a partial header displaying the period during which lunch is served. There are room headers at the left end of each row (and at the right end of each row on the right hand

Even so, a session title may contain a phrase or a word unfamiliar to $\text{T}_{\text{E}}\text{X}$, which it can't hyphenate. And $\text{T}_{\text{E}}\text{X}$ won't try to break a word that already has a hyphen or a slash in it, except at the hyphen or the slash. One way to tell $\text{T}_{\text{E}}\text{X}$ where it may hyphenate a word is to insert a `\-` (a "discretionary" hyphen) wherever hyphenation would be acceptable: for example, "hy\phen\ate". The `\-`'s will disappear in the output, but $\text{T}_{\text{E}}\text{X}$ will know it can hyphenate the word at those places. (But remember, if you do that, to insert discretionary hyphens at *all* the acceptable points in the word.)

Figure 11: SAG Documentation—Verbatim and Hyphenation

```
Even so, a session title may contain a phrase or a word unfamiliar to \TeX,
which it can't hyphenate. And \TeX\ won't try to break a word that already has
a hyphen or a slash in it, except at the hyphen or the slash. One way to tell
\TeX\ where it may hyphenate a word is to insert a \verb+--+ (a "discretionary"
hyphen) wherever hyphenation would be acceptable: for example,
'\verb+hy\phen\ate+'. The \verb+--+ 's will disappear in the output, but
\TeX\ will know it can hyphenate the word at those places. (But remember, if
you do that, to insert discretionary hyphens at {\sl all} the acceptable points
in the word.)
```

Figure 12: SAG Documentation—Input for hyphenation example

To keep a title (or a room name) from being broken at a particular space, replace the space with a `~` (tilde). For example,

Rooms 6 & 8

 may look odd. Typing "Rooms 6~\&~8" instead gives

Rooms 6 & 8

 But don't put tildes where you don't have to; let $\text{T}_{\text{E}}\text{X}$ have as much freedom to format the lines as you can.

Figure 13: SAG Documentation—In-line boxes

To keep a title (or a room name) from being broken at a particular space, replace the space with a `\verb+~+` (tilde). For example, `\fbox{\parbox{.75in}{\centering Rooms 6 \& 8}}` may look odd. Typing `'\verb+Rooms 6~\&~8+'` instead gives `\fbox{\parbox{.75in}{\centering Rooms\\6~\&~8}}`. But don't put tildes where you don't have to; let $\text{T}_{\text{E}}\text{X}$ have as much freedom to format the lines as you can.

Figure 14: SAG Documentation—Input for in-line boxes

MONDAY											SPRING 1986 DECUS U.S. SYMPOSIUM						
ROOM	9:00	9:30	10:00	10:30	11:00	LUNCH					1:30	2:00	2:30	3:00	3:30		
	9:30	10:00	10:30	11:00	11:30	12:00	12:30	1:00	1:30	2:00	2:30	3:00	3:30	4:00			
E. Ballrm A	NETSIG ROADMAP NB51		NETWORKS/COMM. UPDATE NB38					LAS OVERVIEW NB17		LAS UPDATE NB15		DIGITAL & PACKETSWITCHING NTWK NB15		NETWORKS OVERVIEW NB55			
E. Ballrm B	BASEWAY DAB14		BASEWAY IMPLEMENTATION ISSUES DAB17		DATA ACQU. & CTRL. DAB01		DEC'S RESPONSE TO REALTIME DAB31					MICROYAK WORKING GROUP MEETING Y144					
South 411	BUSINESS APPLICATIONS ROADMAP BAB31			APPL. IN PDB & FACILITIES MGMT BAB00		APPL. SOFTWARE FOR DEC SYSTEMS BAB17		WHAT ARE PEOPLE USING VTX FOR BAB16		WHAT PROBLEMS CAN VTX SOLVE? BAB53		BUYING 3RD PARTY ACCT'G SFTWRE BAB19		SELECTING APPLICATION DEV TOOL BAB34			
South 412	COMMERCIAL LANGUAGES ROADMAP CLB27		ANNOUNCING VAX COBOL GENERATOR CLB21					PRACTICAL NETWORKING NB44		THE NEW ANSI COBOL STANDARD CLB19		VAX COBOL DEMONSTRATION CLB20					
South 413				INTRODUCTION TO A-TO-Z DAB12		A-TO-Z IN BUSINESS ENVIRONMENT DAB14			A-TO-Z STATE OF THE PRODUCT DAB15		FORUM/FUTURE TECH DAB16		A-TO-Z APPL. GENERATOR INTRO DTB08				
Arena	YAK SYS. SIG. ROADMAP YB11		YAK SYS. UPDATE Y112		VMS UP-DATE YB07		L&T UPDATE AND ROADMAP L7B50		YAK NOVICE SOFTWARE Q & A Y156			RTM/RTM OVERVIEW Y117		RTM SYSTEMS OVERVIEW Y116			
Theatre					NET KEYNOTE NB43		MICROYAK UPDATE NB33		HUMAN ENGINEERING AT DEC YB27		GRAPHICS ON WORKSTATIONS GB14		YAKSTATION OVERVIEW GB30				
South 414				IBM AND WANG MIGRATION SB11					PRODUCTIVE CENTER MANAGEMENT SB07		MULTI-VENDOR NETWORK MAINT SB27		RT-11 KRON INTERNALS RT020				
E. Ballrm C	GAPSIG BUSINESS GB21		GAPSIG ROADMAP GB22		GRAPHICS STANDARDS GB05		GWS AND GWS-3D GB17		HIGH-END VAXCLUSTER I/O BENCHMA YB76		PERF. MGMT FOR VAXCLUSTERS Y332		CONSULTANTS FOR SYSTEM MGRS SB79				
North 233 & 234	SITE ROADMAP SB71					DOD WORKING GROUP SB33		SITE BUSINESS MEETING SB75		YAK SIG. WKG. GRP CHAIRMAN'S MTG Y144		FIELD SERVICE Q & A SB05					
W. Ballrm C	RST'S OPENING RSB12		RST'S ANNOUNCEMENTS RSB01		RST'S NEW HARDWARE SUPPORT RSB05		RST'S/E Y12 AND Y13 RSB04		RST'S TECH TIPS I (SPR) RSB15			RST'S ON A CLUSTER OF J11'S RSB23		BASIC-4.2 V2 HINTS AND KINKS CLB18			
W. Ballrm D	RSX SIG. OPENING SESSION RB07		RSX/POS. PRODUCT PANEL/Q & A RX010		WHAT THE PDP-11 LOOKS LIKE RX010			DDT CAN BE FUN RX028		RSX SIG. WORKING GROUP MEETING RX048		RSX 3RD PARTY HARDWARE RX030		AN ULTIMATE RSX SYSTEM RX010			
West 101				PC SIG. BUSINESS MEETING PB02		USING THE ATLANTA HOT-LINE PB06		NEW PC DIRECTIONS PB18		RAINBOW COMMUNICATIONS/NETWORK PB22		MS-DOS TECHNICAL W/Q&A PB28					
West 102	MUMPS ROAD MAP SESSION MB01		MUMPS TUTORIAL PART 1 MB00						DOD MUMPS APPLICATIONS MB12		MUMPS DOD REVERSE ENGINEERING MB11		MULTI-LINGUAL TEST TRANSLATION M013 MT013		DECREPORTER DT023		
W. Ballrm A & B						INTRODUCTION TO VAX SET LTB45			VAX LSE TUTORIAL LTB13		VAX DEBUG TUTORIAL LTB22						
West 105 & 106	DMS AND DTR/IGL SIBS OPENING DTB33		VAX INFO ARCH. OVERVIEW DMB38		VAX INFO ARCH. DATABASES DMB48		IBM VAX DB INTERCONNECT I DMB78		DATA MANAGEMENT SURVIVAL DMB18		OLTP CONCEPTS DMB31		MAKING DR USERS FRIENDLY DMB31		DMB10		
West 107				AI SIG. ROADMAP AIB23		AI INTRO AIB06					EXP. SYS. INTRO AIB42		AI LIT AIB09		5TH GENERATION AIB44		
West 109	UNISIG ROADMAP/ULTRIX PROD. PANEL UB02			ULTRIX & LASER PRINTERS UB27				NEW PROCESSORS FOR ULTRIX UB29		ULTRIX UTILITIES/TOOLS UB31		INTRO TO YACC AND LEX UB00		UNIX PATTERN MATCHING UB21			
North 214 & 216	EDUSIG BUSINESS MTG & ROADMAP EB31		K-12 SOFTWARE TOOLS EB35			COURSE MANAGEMENT & TESTING EB28					K-12 ADMIN SOFTWARE - PART 1 EB41		K-12 ADMIN SOFTWARE - PART 2 EB34				
North 230 & 231	IAS SIG. OPENING SESSION IB00				IAS PRODUCT PANEL IB04		WHAT ARE 4GLs DTB24		TEAMDATA AND RALLY OVERVIEW DTB08		WHAT IS ADEP? DTB34		VAX DTR REMOTE DATA ACCESS DTB01				
E. Ballrm D	HMS ROADMAP HB18		STORAGE SYSTEMS ROADMAP HB03		ELECTRONIC MEMORY OVERVIEW HB07		CAD SYSTEMS DEVELOPMENT Y321		MICROYAK W/S PRODUCT FEATURES GB18		MSD DIRECTIONS AND NEW PRODUCT HB01		PRINTERS FUTURE NEEDS HB21				
East 409 & 410	RT-11 SIG. BUSINESS MEETING RTB14		RT-11 SIG. ROADMAP RTB13		RT-11 PRODUCT PANEL RTB08						VAX EDCS DAB11		BASEVIEW DAB12				
North 218 & 219	DAARC OPENING SESS. DAB52		VAX LIMS/SM DATABASE DAB03		GETTING LIMS/SM TO WORK IN LAB DAB04						CUSTOMIZING YOUR LIMS SYSTEM DAB06		VAX LIMS SM IM PROTOCOL DAB05				
East 301 & 302	OASIS ROADMAP OB06		OA - THE STATE OF THE ART OB21		EXECUTIVE INFORMATION SYSTEMS OB11		TLC WITH A LEAN & MEAN SYSTEM OB57		MAIL STRATEGY OB28		STRATEGIC DIRECTIONS FOR OA OB46		OA SYSTEMS UPDATE OB22				
ROOM	9:00	9:30	10:00	10:30	11:00	LUNCH					1:30	2:00	2:30	3:00	3:30		
	9:30	10:00	10:30	11:00	11:30	12:00	12:30	1:00	1:30	2:00	2:30	3:00	3:30	4:00			

Figure 15: The Sessions at a Glance

```

\documentstyle{article}

\pagestyle{empty}

\textwidth=8.5in      \textheight=10.5in
\topmargin=-1in      \oddsidemargin=-.75in

\tabcolsep=1pt      \arrayrulewidth=.8pt

```

Figure 17: SAG Macros—Houskeeping

page, which isn't shown here).

Individual sessions may occupy any number of quarter-hour periods in a room. Each session's title appears at the top of its cell, left-justified, and its session number at the bottom, right-justified. In any room, any number of periods may be empty. In the case of shorter sessions, long titles may overlap session numbers unless the titles are manually shortened.

The SAG's input format is about as simple as could be devised (Figure 16). Input for a page begins with a “\saagpage” command, the arguments of which identify the day, the page segment, the symposium season (Spring or Fall), and the year. L^AT_EX's internal memory isn't big enough to hold an entire page, so input for one page is divided into top, bottom, or center segments.

Next, the input contains a call to the “\room” macro which conveys the name of the room to be typeset in the “Room” column, and signifying the start of session input for that room.

Next come “\session” commands, giving, first, the number of quarter-hour periods the session will occupy, the session number, and the session title. The “\opentime” macro is similar to the “\session” macro, except that the only argument gives the length of time the room is unused.

This input would be followed by the “\room” command for the next row in the table, and its session information.

L^AT_EX will read a file of SAG macro definitions (Figure 17) as well as SAG input files. It will also read a “style file” of macros from the L^AT_EX data base. In this case, the “\documentstyle” is “article”, indicating a particular set of general definitions which L^AT_EX will use for this

```

\font\saag=tphon6
\newcommand{\TINY}{\tiny\saag}

\font\eightbf=amssbx8   \font\tenssbf=amssbx10
\font\headfont=amssbx12 \font\cxfont=amssbx16

```

Figure 18: SAG Macros—Fonts

job.

“\pagestyle” is “empty” because we want no running heads or page numbers on these SAG pages. Commands follow which set the size and position of the printed area on the page. “\tabcolsep” establishes a value that increases the default separation between columns in the table. “\arrayrulewidth” sets the thickness of the rules in the table to a reasonably dark eight tenths of a point (a point is one seventy-second of an inch).

In Figure 18, we establish some T_EX fonts we will want to access which are not preloaded by L^AT_EX. The command “\font\saag=tphon6” tells T_EX that, when we give the command “\saag”, we want it to access the font file “tphon6.tfm” in its font data base and use the font metrics—that is, the character size and relative position information—which are given in that file. “Tphon6” is a small (6-point), closely spaced font, right for putting session titles in the tight spaces of the SAG. This command associates a logical font name with a physical font metrics file.

We also define a new command, “\TINY”. Its definition means that, when we specify “\TINY”, we are calling L^AT_EX's littlest font, “\tiny”. “\TINY” will use all of “\tiny”'s built-in characteristics, such as line spacing and interword spacing, but will use character metrics from Tphon6.tfm.

We also define 8-, 12-, 14-, and 16-point sans serif bold fonts for use in our various headers.

In Figure 19 we enter commands that place the slash character, “/”, in the class of characters which T_EX will accept as hyphenation points. This makes it easier for T_EX to break phrases such as “RSX/ POS PRODUCT PANEL/Q & A” in tight places. These are T_EX commands, passed along by L^AT_EX. The construct “\” tells T_EX to consider just the single following character. Putting a character into “\catcode” 13 means that that


```

\saagpage{Monday}{lefttoppage}{spring}{1986}
\room{E.~Ballrm\A}
\session{2}{NO51}{NETSIG ROADMAP}
\session{2}{NO30}{NETWORKS/COMM. UPDATE}
\opentime{2}
\session{2}{NO17}{LAS OVERVIEW}
\session{2}{NO15}{LAS UPDATE}
\session{2}{NO35}{DIGITAL \& PACKETSWITCHING NTWK}
\session{2}{NO50}{NETWORKS OVERVIEW}

```

Figure 16: SAG Input

```

\catcode'\/=13\def/{\char'\/\penalty0\hskipOpt
}}

\uchyph=1

```

Figure 19: SAG Macros—Defining extra hyphenation points

character is to be considered a command word, all by itself. So the command “`\catcode'\/=13`” tells T_EX that the character “/” is a command.

Then we define what the command “/” means, using T_EX’s method, the “`\def`” command, for defining a new command. After this, whenever T_EX comes to a “/” in the text, T_EX will think it’s a command, rather than just another text character, so it will act on it as a command rather than pass it to the output stream. So the first thing the new definition does is to call for outputting an ascii “/”. The T_EX command “`\char`” does this. Again, the “`\`” identifies the next character as our target.

After outputting an ascii “/”, the next thing our new “/” command does is establish a linebreak “`\penalty`” of zero. That give T_EX the right to break the line at the “/” with no penalty, which is the same as giving T_EX permission to break the line there if it wants to. Since T_EX is only allowed to break lines at spaces, our new “`\verb/+`” command ends by inserting a space with zero width, “`\hskipOpt`”, to complete the task of allowing T_EX to break lines at slashes when it wants to.

Normally, T_EX doesn’t hyphenate words containing uppercase letters, believing them to be names of things (“proper nouns”), which aren’t supposed to be hyphenated in English. “`\uchyph`” tells T_EX to ignore that rule and go ahead and hyphenate such words. We need it here, since session titles are all in upper case, and if we didn’t use it, T_EX could never break any lines to fit them in our tiny boxes.

Figure 20 presents several commands (“macros”) defined for the SAG. “`\newcommand`” is L^AT_EX’s method of creating a new command. “`\newcommand`” takes two required arguments enclosed in braces (“{ }”). The first is the name of the new command and the second is the new command’s definition. Between these two required arguments may appear an optional argument, enclosed in brackets (“[]”), which is the number of arguments to be passed when the new command is called.

In Figure 20, the principal new macro is “`\leftclock`”, which establishes the times printed in the headers and footers of left-hand pages (there’s a similar “`\rightclock`”, not shown). Another, “`\hour`”, is used in “`\leftclock`” to enter start and stop times in the column headers. As denoted by the “[1]” following the macro name in its definition, “`\hour`” takes a single argument in its call, namely a time range as can be seen from the calls to it in “`\leftclock`”.

“`\hour`” contains an example of L^AT_EX’s “`\multicolumn`” command. Since sessions may be scheduled in quarter-hour intervals, the SAG must recognize quarter-hour “ticks of the clock”. But printing headers for every fifteen-minute period would produce a too-crowded result, so headers are printed for each half-hour time period, with ticks marking the quarter hours. “`\hour`”, therefore, spans two table columns (the table itself will be defined in later macros) as shown by the value, ‘2’, of

```

\newdimen\minwidth      \minwidth=.23in
\newdimen\widewidth     \widewidth=\minwidth
\advance\widewidth by \arrayrulewidth

\newcommand{\hour}[1]{&\multicolumn{2}{p{2\minwidth}}{\clock{#1}}}
\newcommand{\widehour}[1]{&\multicolumn{2}{p{2\widewidth}}{\clock{#1}}}
\newcommand{\clock}[1]{\centering \eightbf #1}
\newcommand{\spacer}{\hspace*{\minwidth}&}

\newcommand{\leftclock}{\\ \hline&
\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer
\multicolumn{8}{|c|}{\rule{0pt}{.8em}\eightbf LUNCH}&\spacer\spacer
\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer\hspace*{\minwidth}\\
\cline{12-19}\parbox{.75in}{\tenssbf \centering ROOM}%
\hour{9:00 9:30}\hour{9:30 10:00}\hour{10:00 10:30}\hour{10:30 11:00}%
\hour{11:00 11:30}\widehour{11:30 12:00}\widehour{12:00 12:30}%
\widehour{12:30 1:00}\widehour{1:00 1:30}\hour{1:30 2:00}\hour{2:00 2:30}%
\hour{2:30 3:00}\hour{3:00 3:30}\hour{3:30 4:00}}

```

Figure 20: SAG Macros—Leftclock

its first argument.

Before the definition of “\hour”, a new dimension, called “\minwidth” is defined, having a value a little under a quarter inch. The SAG must display 7 hours—28 quarter-hour columns—plus a “room” column, on each page. On an 8.5-inch-wide page, the width of each quarter hour column can be no more than “\minwidth” (not counting the width of the vertical rules between columns). “\multicolumn’s” second argument specifies here that the contents of this double column are to be set in “p” (paragraph mode) to a width of 2 times “\minwidth”, with a vertical rule set at its righthand side. The third argument specifies the text to be set there by invoking another macro called “\clock”.

Remember that, when “\hour” is called, it will be called with one argument, a time range such as “10:00 10:30”. The “#1” in the defining string of “\hour” marks where this argument is to be inserted. Here, the result is that the argument is passed along to the “\clock” macro, which, like “\hour”, is defined as taking one argument in its call. “\clock” embeds the time-range argument in a “\centering” environment and invokes an 8-point bold sans-serif font with which to typeset it. It is convenient to define macros like “\hour” and “\clock” which will be used more than once and will reduce the number of keystrokes, and hence the number of errors, in the full set of definitions.

“\leftclock” itself will be called from inside a table-building macro, so it contains commands appropriate to tables. Its first move (caused by a double backslash command), is to start a new line in the table. “\hline” draws a horizontal line across the table, but leaves us positioned at the beginning of the table’s first (leftmost) column.

Next comes an &, which is L^AT_EX’s command to move to the next column in a table. This moves us to the second column, which is the first of the 28 quarter-hour columns. The command “\spacer” has been defined as a horizontal space of “\minwidth” followed by an &, the effect of which is to establish the horizontal distance to be occupied by the column, cause an intercolumn vertical rule (called for in the table definition) to be set, and then skip to the next column. Calls to “\spacer” move us across 10 columns to get to lunch.

Now we establish an 8-column “\multicolumn”, with text centered (“c”) and vertical rules on either side. The text to be centered consists of the word LUNCH plus a vertical rule .8 ‘em’s high (an ‘em’ is the width of an M in the current font) and 0 points wide (and therefore invisible). This rule forces a little vertical white space to make lunch look better.

Then an & moves us into the 19th quarter-hour column, and 9 more calls to “\spacer” bring us to the 28th column. This last column’s dimension is established by the command “\hspace*{\minwidth}”, which is the same

```

\newcommand{\saagpage}[4]{\gdef\saagday
{\uppercase{#1}}\gdef
\meeting{\uppercase{#3 #4}}\csname #2\endcsname
\gdef\tabname{\uppercase{#1}--#2}\ignorespaces}

```

Figure 21: The SAAGPAGE Macro

as a call to “\spacer” but without the & since there are no more columns to skip into. Another “\” command moves us down to the beginning of the next row.

We draw a horizontal line, using “\cline”, across table columns 12–19, which are the same columns spanned by the previous “\multicolumn” command, so we get a line for lunch.

We’re now back at the lefthand side of the table, so we write the “ROOM” header, calling for our 10-point sans serif boldface font. We put it inside a “\parbox” and use a “\centering” command (otherwise it would be left-justified in its box). Vertically, the word ROOM will come out at the top of the table established by the previous “\cline” command, putting some white space below “ROOM”, but since the “\cline” doesn’t come this far, “ROOM” will also have lots of white space above it (see Figure 15).

Then we invoke the “\hour” macro enough times to enter all the time periods in the left hand page of the SAG. We’re forced to define “\widewidth”, which we make equal to “\minwidth” plus the width of a vertical rule, and to define “\widehour” using “\widewidth” instead of “\minwidth” to take account of the fact that the multicolumn span containing LUNCH doesn’t include the quarter-hour tick marks the other columns do. Using “\hour” to write the time headers during the lunch period would result in a space deficit across this period equal to the thickness of four vertical rules, a quite noticeable four-hundredths of an inch, which would all be added to the rightmost column under the LUNCH header.

Recall that SAG input files begin with a call to the “\saagpage” macro, with four arguments: the day, the page segment, the season, and the year. Figure 21 gives the definition of “\saagpage”. Note that it expects four arguments with its call. As the first thing in its definition, “\saagpage” defines another macro, “\saagday”. “\saagday” is redefined each time “\saagpage” is called, and has as its definition the first argument (which is

the name of the day) in the call to “\saagpage”. Once “\saagpage” is called, and “\saagday” is defined, any call to “\saagday” will expand to that day name until the next call to “\saagpage” redefines “\saagday”. “\gdef” tells T_EX to make the definition of “\saagday” global so that it will remain defined after this call to “\saagpage” is completed.

Then another macro, “\meeting”, is “\gdef”ed, consisting of the season name (“#3”) and the year (“#4”). Now, when we need the name of the day or the Symposium in later header macros, we simply refer to “saagday” of “\meeting”.

Next we use the construct “\csname #2\endcsname” to call a macro whose name is whatever the second argument, “#2”, of the call is. Recall that the second argument is the page segment name, so this issues a call to a macro whose name is the same as the page segment name. If the page segment argument is, for example, “lefttoppage”, this results in a call to “\lefttoppage”. This call is expanded right away, before the rest of the call to “\saagpage” is completed, but when we do return to finish expanding “\saagpage”, all we do is define another macro named “\tabname”, consisting of the name of the day followed by the page segment, which we will use later on to give a label to the partial output for this segment.

Figure 22 shows the way that segment will look, once we’ve gotten through all our definitions and have run the input against them. Notice that the main header is going to invoke the “\saagday” macro and print “MONDAY” in the upper left hand corner. To the right of that, the “\meeting” macro will give us the season and the year, “Spring 1986”. The remaining text is just a literal in a later macro.

“LUNCH” spans columns 7–10 with our “\cline” underneath it. “ROOM” is the first column header, aligned with the upper time. Notice the label at the bottom of the table. This is the result of a call to the “\tabname” macro we just defined, and we’ll see how and where that call is made. The DECUS Office in Marlboro played a major role in creating the SAG. In particular, Brent Lapham, the DCS System Manager, prepared the input files, breaking them into the necessary segments for each page to avoid exceeding L^AT_EX’s memory limit. Judy Arsenault’s page makeup staff stripped the typeset output segments together. The label at the bottom of each segment helped avoid mixing them up.

In Figure 23, the second definition is “\lefttoppage”, which we just used as an example. The first definition,

MONDAY														SPRING 1986 DECUS U.S. SYMPOSIUM													
ROOM	9:00	9:30	10:00	10:30	11:00	LUNCH				1:30	2:00	2:30	3:00	3:30													
	9:30	10:00	10:30	11:00	11:30	12:00	12:30	1:00	1:30	2:00	2:30	3:00	3:30	4:00													
E. Ballrm A	NETSIG ROADMAP N051		NETWORKS/COMM. UPDATE N030						LAS OVERVIEW N017	LAS UPDATE N015	DIGITAL & PACKETSWITCHING NTWK N035		NETWORKS OVERVIEW N050														
E. Ballrm B	BASEWAY DA014		BASEWAY IMPLEMENTATION ISSUES DA017		DATA ACQU. + CNTRL DA008		DEC'S RESPONSE TO REALTIME DA031				MICROVAX WORKING GROUP MEETING Y148																
South 411	BUSINESS APPLICATIONS ROADMAP BA036		APPL IN PBX & FACILITIES MGMT BA060		APPL SOFTWARE FOR DEC SYSTEMS BA047		WHAT ARE PEOPLE USING VTX FOR BA046		WHAT PROBLEMS CAN VTX SOLVE? BA053		BUYING 3RD PARTY ACCTNG SFTWRE BA010		SELECTING APPLICATION DEV TOOL BA024														
South 412	COMMERCIAL LANGUAGES ROADMAP CL027		ANNOUNCING VAX COBOL GENERATOR CL023		PRACTICAL NETWORKING N048				THE NEW ANSI COBOL STANDARD CL018		VAX COBOL DEMONSTRATION CL020																
South 413	INTRODUCTION TO A-TO-Z BA012		A-TO-Z IN BUSINESS ENVIRONMENT BA014				A-TO-Z STATE OF THE PRODUCT BA015		FORUM/FUTURE TECH BA016		A-TO-Z APPL GENERATOR INTRO DT036																
Arena	VAX SYS. SIG ROADMAP V010	VAX SYS. UPDATE V112		YMS UP-DATE V097	LAT UPDATE AND ROADMAP LT050		VAX NOVICE SOFTWARE Q & A Y158				R200/R300 OVERVIEW Y117		R800 SYSTEMS OVERVIEW Y116														
Theatre	NET KEYNOTE N083				MICROVAX UPDATE H033		HUMAN ENGINEERING AT DEC V002		GRAPHICS ON WORKSTATIONS G024		VAXSTATION OVERVIEW G039																

Table 1: MONDAY-lefttoppage

Figure 22: A LeftTopPage Segment ready for page makeup

```

\newcommand{\leftfullpage}{\openleftpage
\global\let\endsaagpage=\closeleftpage}

\newcommand{\lefttoppage}{\openleftpage
\global\let\endsaagpage=\finishpage}

\newcommand{\leftcenterpage}{\leftpageinit
\leftclock\global\let\endsaagpage=\finishpage}

\newcommand{\leftbotpage}{\leftpageinit
\global\let\endsaagpage=\closeleftpage}

```

Figure 23: SAG Page Segment Macros

“\leftfullpage”, can be used when a page is small enough to be set in a single segment, but that’s usually only the case for test data.

The definition of “\lefttoppage” first calls “\openleftpage”, a macro which will invoke all the other macros necessary actually to set the left page headers, like “\saagday” and “\leftclock”, and others. Then we use a TeX convention, “\let”, to equate one command to another, to suite our needs of the moment. Just as each input file begins with a “\saagpage” command, each one ends with an “\endsaagpage” command. But the “\endsaagpage” command must do different things, depending on what sort of segment we’re setting—right page or left, top segment, center, or bottom. Rather than force the input to contain a different end command for each case, we use the fact that we know, at the beginning, what sort of segment we’re working with, and equate the standard end command to an appropriate macro. In this case, “\finishpage” will give us the actions we want at the end of a top page segment.

Notice that “\leftfullpage” calls “\openleftpage” at the start of the page and “\closeleftpage” at the end. Those macros set the complete set of headers, top and bottom. “\lefttoppage” begins with the same opening call, as you’d expect, but end with “\finishpage”, a macro which just finishes off the segment without setting any footers.

Again, as you’d expect, “\leftbotpage” ends with a call to a macro that will set the footer, but begins with “\leftpageinit”, which begins the segment with no headers. “\leftcenterpage” ought not to need either headers or footers, and it does open and close with the non-heading calls. But a call to “\leftclock” does have

```

\newcommand{\openleftpage}{\leftpageinit\hline
\multicolumn{15}{|c|}{\headfont\rule[-.4em]{
\Opt}{1.6em}\hspace*{2em}\saagday \hfill
\meeting\ DECUS U.S. SYMPOSIUM\hspace*{2em}\null}}%
\leftclock}

\newcommand{\leftpageinit}{%
\global\let\room=\leftroom
\global\let\session=\leftsession
\global\let\opentime=\leftopentime
\begin{table}
\begin{tabular}{|p{.75in}|*{28}{p{\minwidth}}|}}

```

Figure 24: Left-segment opening macros

to be inserted, artificially, and the headers stripped out during page-makeup, because otherwise IAT_EX takes its column widths only from the variable width of the session headers which it sets in the various columns, without realizing that the headers force a standard width on the table. Without this call to “\leftclock” the center segment just doesn’t match up with the top and the bottom.

The two left-segment opening macros appear in Figure 24. The first one, “\openleftpage”, begins by calling the second, so we’ll look there first. “\leftpageinit” is common to any left-page segment, since it’s called directly by those segments that don’t need the headers in “\openleftpage”. First, it sets the “\room” command, which we saw starts the input for each row in the table, to “\leftroom”. That is, until this assignment is changed, any call to “\room” is a call to “\leftroom”. On right hand pages, of course, a different assignment is used. Similarly, “\session” and “\opentime”, the two macros that convey information about room use per time period, are equated to their left-brain implementations.

Then the main table itself is begun. In IAT_EX, the “table” environment is used mainly to permit proper placement of a table on the page, allowing it to float to the top or appear on a separate page. The table environment also provides the label facility we saw previously.

The “tabular” environment provides the actual table definition, establishing 29 columns. The first is to be treated as though it contained a “\parbox” three quarters of an inch wide. This is the column that will contain our room names. It will have a vertical rule on each side. Following that comes a column described as a “\minwidth” wide

```
\newcommand{\closeleftpage}{%
{\leftclock\finishpage}
```

Figure 25: Left-segment closing macro

```
\newcommand{\finishpage}{\\hline\end{tabular}}%
\caption{\tabname}\end{table}\vfill\clearpage
\gdef\saagday{\null}\gdef\meeting{\null}}
```

Figure 26: The Finishpage Macro

“`\parbox`” with a vertical bar on its right side, and this column is repeated a total of 28 times.

Now we’re done with “`\leftpageinit`”, but we need to finish the expansion of “`\openleftpage`” if we came from there. After the call to “`\leftpageinit`”, we draw a horizontal line across the whole table we just defined and set the full-page header with a call to “`\multicolumn`” that spans all 29 columns. We use the 12-point sans serif bold font we defined earlier. We insert an invisible vertical rule that pushes the bottom of the current line down four tenths of one em and pushes the top up 1.6 ems, to provide a little extra white space above and below the header text. Then we force \LaTeX to indent (“`\hspace*`”) 2 ems, issue a call to “`\saagday`” to print the day, call for expandable horizontal space (“`\hfill`”) to push the following text over to the right side of the line, call “`\meeting`” to print the season and year of the Symposium, print the words “DECUS U.S. SYMPOSIUM”, and finish the line with a 2-em indentation, protected by a “`\null`” command to keep \LaTeX from gobbling up space at the end of a line. “`\openleftpage`” ends with a call to “`\leftclock`”, which establishes lunch time and all the column headers for the ROOM column and 14 half-hour time periods.

Of course, eventually we’ll have to end a page segment. “`\closeleftpage`” (Figure 25) is used to end a left bottom page segment, calling “`\leftclock`” to typeset the same column headers as footers at the bottom of the page, and then calls a common finishing macro used for any left page segment.

“`\finishpage`” (Figure 26) breaks the line, draws a horizontal rule, and terminates the “`tabular`” environment.

```
\newcommand{\leftroom}[1]{\\hline\multicolumn
{1}{p{.75in}}{\raisebox{-2ex}{\parbox
{.75in}{\tenssbf\centering #1}}}\ignorespaces}
```

Figure 27: The Leftroom Macro

```
\newcommand{\leftsession}[3]{&\multicolumn
{#1}{p{#1\minwidth}}{\entry{#1}{#2}{#3}}%
\ignorespaces}
```

Figure 28: The Leftsession Macro

We’re still inside the “`table`” environment, though, and it is that environment which provides the caption facility. We defined “`\tabname`” earlier, in the definition of the “`\saagpage`” macro, to be the name of the day plus the page segment name: Monday–Lefttoppage, for example. A call to “`\caption`”, with an argument of “`\tabname`” the caption. Then we leave the “`table`” environment. We insert vertical space (“`\vfill`”) to fill the page, eject the page with “`\clearpage`”, causing the table typesetting to be completed, and redefine “`\saagday`” and “`\meeting`” as empty macros, just to be sure that their definitions don’t get left over and picked up by subsequent macro calls without being properly redefined.

During left page segments, the “`\room`” command is equated to the “`\leftroom`” command, which is defined in Figure 27. It takes one argument, the room name. When “`\room`” is read in the input file, a line break (double backslash) command is issued and a horizontal rule is drawn. A “`\multicolumn`” command, spanning just the ROOM column, is issued to create a special environment for setting the room name. The column specifications make it a three-quarter inch box with vertical rules on either side. The “`\parbox`” (of the same size) provides the environment for the ten-point sans serif bold font and for centering. The “`\raisebox`” command, with a negative argument, lowers the text to approximately center it vertically in the cell.

Similarly, we equate the “`\session`” command to “`\leftsession`” for left-page segments (Figure 28). “`\session`” has 3 arguments: the number of quarter-hour periods, the session number, and the session title. First, the “`&`” moves us into the next column. Then we

```
\newcommand{\leftopentime}[1]{&\multicolumn{#1}%
{p{#1\minwidth}|}{\makebox[#1\minwidth]{}}%
\ignorespaces}
```

Figure 29: The Leftopentime Macro

issue a “`\multicolumn`” command spanning a number of columns equal to the number of quarter-hour periods (argument #1 of the call). The width of the box needed to hold “`\multicolumn`”’s text is equal to “`\minwidth`” multiplied by the number of columns spanned (#1), expressed to T_EX by this convention. There’s to be a vertical rule at the right side of the multicolumn span. The text to be set in the cell thus defined will consist of the session title (#3) and the session number (#2). That material is formatted by the entry macro, which we call here.

“`\ignorespaces`” eliminates spaces in the input stream which occur between commands (primarily because T_EX converts carriage returns occurring between commands into spaces). Such spaces would act like text occurring between “`&`” column-skip commands, and would force extra width into cells, destroying our table’s alignment.

“`\opentime`” (see Figure 29) has a definition similar to “`\session`”’s, except that, instead of calling the “`\entry`” macro to format the material going into the cell, we make an empty box. The “`\makebox`” command has an optional argument (optional arguments are enclosed in square brackets, as opposed to required arguments, which are enclosed in curly braces) which is the width of the box. Again, as with “`\session`”, the width of the cell is computed by multiplying “`\minwidth`” by the number of quarter-hour time slots. What goes in this box, in the case of “`\opentime`”, is nothing, which is enclosed in curly braces here.

Figure 30 shows a right hand page segment, looking much the same as a left hand page segment, except that the room column is on the right. To make the input simple, right hand input takes the same form as left hand input. That is, a segment starts with the “`\saagpage`” command, followed by a “`\room`” command, followed by “`\session`” and “`\opentime`” commands, and then more “`\room`”s. On left pages, the room name is set first, just as it’s input. But on right pages, “`verb+room+`” has to set the previous room name, then start a new line and save the new room name to be set when the next “`\room`”

```
\newcommand{\entry}[3]{\TINY
\parbox[t]{#1\minwidth}{\leavevmode
\, \ifxld\xntry{#1}\fi
\rlap{\vbox to \ht\parnbox{\hsize=#1\minwidth
\raggedright #3\ \phantom{#2}\vss}}%
\null\hfil\break
\null\hfil\break
\null\hfil\break
\null\hfill #2\,}}
```

Figure 31: The Entry Macro

is encountered. The macros that do this are similar to the left page macros, but have that added complication, which I won’t detail here (a complete listing of all the SAG macros appears at the end of this article).

Before we come to the “`\entry`” macro, we need a couple of definitions. It’s convenient for us to establish a standard line height, in the font we’re using for table matter, that we can refer to conveniently. We do that by using T_EX’s “`\newbox`” command to create a box. The box will represent the height of a pair of parentheses, which are tall characters, and “`\newbox\parnbox`” gives the box a reasonably mnemonic name.

We use T_EX’s “`\setbox`” command here to establish the contents of the box. We define it to be a horizontal box, “`\hbox`”—that is, an ordinary text-type box. We invoke the “`\TINY`” font that we defined for use in the table, and we “set” a pair of parentheses in that font. We use the “`\phantom`” command to set them, though, which means they’ll be invisible when we typeset this box, but they do nevertheless establish the height of the box, which is the only property of the box we’ll need to use.

The “`\entry`” macro, shown in Figure 31, is going to reuse the same three arguments that come with the “`\session`” command to typeset the contents of one table cell. Recall that these are the number of quarter-hour time slots, the session number, and the session title, in that order.

The cell contents are going to be set inside a “`\parbox`”, where we have local control of spacing & style. “`\parbox`” can have an optional argument (optional arguments are enclosed in square brackets) which indicates whether the top line, bottom line, or center (the default) of the “`\parbox`” is to be aligned with the current line of text;

SESSIONS-AT-A-GLANCE														MONDAY		ROOM	
4:00	4:30	5:00	5:30	6:00	6:30	7:00	7:30	8:00	8:30	9:00	9:30	10:00	10:30	11:00			
		TERMINAL SERVER OVERVIEW N045	DEC,IBM,DDP... COMMITMENT N028	OSI TUTORIAL N034				NETWORKS MAGIC SESSION N004						E. Ballrm A			
	WORD & DOC. PROC. UPDATE O050	COMPOUND DOCUMENTS IN OA O051			DIGITAL'S OFFICE ARCHITECTURE O040	PC'S IN OFFICE SYSTEMS O044			MESSAGE ROUTER APPLICATIONS O025					E. Ballrm B			
	PERF IN SMALL BUS ENVIRONMENT BA035	SIMP APPL W/ STATE TABLES BA033	ELECT. ORDERING / INVOICING BA031	MIXED ETHERNET N047		VAX REALTIME WORKING GROUP MTG V017		DECALC-PLUS O010		XWAY - SPREADSHEET TRANSLATOR O013				South 411			
	DEV DISTRIBUTED APPL. N057			VAX SYSTEM MGMT W/G MEETING V147										South 412			
	A-TO-Z AND ALL-IN-1 DIFFERENCE BA021	MULTINATIONAL SFTWR APPL BA024		PKG SOFT + INTERN'L MKT BA040										South 413			
	VAX BI ARCHITECTURE Y118	VAX/VMS PERF ON NEW PROCESSORS V130			VAX BI INTERFACING Y119			DEC ASKS ABOUT TOOLS V077		JUST A VAX MODEST PROPOSAL V010				Arena			
	VAXSTATION INTERNALS G036	AI WORKING GROUP MEETING V014		VAX MIGRATION WORKING GROUP V022		DECUS TRIVIA CONTEST BA027							Theatre				

Table 5: MONDAY-righttoppage

Figure 30: A RightTopPage Segment

in this case, we want to build from the top down. The first required argument of the “`\parbox`” is its width, computed by the “`\entry`” macro to be equal to the cell’s number of time periods times “`\minwidth`”.

The second and final argument of “`\parbox`” is its contents. We have two main tasks, plus a third I’ll describe afterward. The two main tasks are, first, to set the session title properly formatted within the cell, and, second, to set the session number in the lower right hand corner. For the moment, I’ll skip down to the “`\rlap`” instruction, which tells \TeX to set the following material rightward from its present position, then return to that position. The “`\rlap`” instruction applies to the material contained in its argument. That material commences with a “`\vbox`”, a box to be set in a vertical direction, rather than in a horizontal, “word by word” mode.

We start by giving this “`\vbox`” some dimensions. The convention “`\vbox to \ht`” causes \TeX to establish the “`\vbox`” with a height equal, in this case, to that of a parenthesis in the “`\TINY`” font, represented by “`\parnbox`” (the “`\ht`” command, which assigns a height to a box, must have another box from which to extract an existing height, which is why we defined “`\parnbox`”).

Our “`\vbox`” has a height; now it needs a width. We use \TeX ’s “`\hsize`” command to assign a width, once again computed from “`\minsize`” and argument #1 of the call to “`\saagpage`” and “`\entry`”. So now we have a box whose height is that of a tall character in our font and whose width is that of our current table cell. The next thing we tell \TeX is to typeset argument #3, the session title, ragged right. We’re doing this inside a “`\vbox`” which is the size of one line of text. The title may well require more than one line, but this poses no problem. Each subsequent line will keep the same dimensions as the first. We don’t know how many lines the title will take, but we set the session number at the end (invisibly), to be sure we’ve left enough space to do so.

Next, the “`\vss`” command does the same thing vertically that “`\rlap`” did horizontally. We’ve set at least one line inside the “`\vbox`”, but “`\vss`” tells \TeX that the vertical space we’ve used doesn’t count, just the way “`\rlap`” says that the horizontal space we’ve used doesn’t count. The result is that we’ve set the title in the box but have returned to our starting point horizontally and vertically, for our next move.

Which is to set three empty lines, followed by a fourth line on which we set the session number, pushed over to the right by “`\hfill`”. And we’ve accomplished what we set out to do, namely to typeset the session title in the

cell, and the session number in the lower right corner. As a matter of fact, typesetting the session number the way we did actually determines where the lower left corner will be. By setting three lines of “`\null`”, horizontal space, and linebreak, and then setting the session number on the fourth, we define the box as having four lines. The title may be shorter than that. If so, one or two lines between it and the session number will be blank. If it’s longer than that, or if there isn’t room enough left on the fourth line for the session number, then an error will occur. The session number may be overprinted and the title will have to be manually shortened to fit. (The job of ensuring that there were no too-long titles in the Dallas SAG was done by Brent Lapham in the DECUS Office so accurately that not one was present and the Dallas SAG was typeset without error on the first attempt.)

The code “`\,`” typesets a narrow space. We insert it here before “`\rlap`” is encountered and after the session number to add a little white space at the edges of the cell.

The command “`\ifxld`”, which stands for “if cancelled”, is a conditional statement which we define in Figure 32. It is a test to see if the word “CANCELLED” should be set in our cell. If “`\ifxld`” has been set to true, then the “`\xentry`” macro is called (with argument #1, from which the width of the cell can be computed). “`\fi`”, you’ll recall, is the ending delimiter for the conditional clause.

Text set inside \LaTeX ’s “`\parbox`” is set in paragraph—that is, vertical—mode. A paragraph is a vertical stack of boxes, each box containing one line; setting something in paragraph mode enables \TeX to break it into lines. If \TeX encounters a horizontal box at the beginning of a paragraph, it will put it on the next line. Here, “`\rlap`” creates such a box, and \TeX would leave an unwanted empty line at the top of the cell unless we told it to go into horizontal mode by saying “`\leavevmode`”. Once inside the “`\rlap`” box, \TeX is again in vertical mode and the session title is broken into lines. I must acknowledge the great help and advice given me in the definition of this macro by Barbara Beeton, the American Mathematical Society’s \TeX wizard.

“`\xentry`”, in Figure 32, provides a mechanism for cancelling a session. Recall that this macro was called just before we typeset the session title and number in the table cell. “`\xentry`” will, if necessary, typeset **CX** or **CXLD** or **CANCELLED** in the cell (depending on the width of the cell) *before the session title and number are typeset on top of it.*

```

\newcommand{\xentry}[1]{\rlap{\vbox to \ht\parnbox
{\vbox to 4\baselineskip
{\hsize=#1\minwidth\cfont\vfll\begin{center}%
\ifcase#1\or CX\or CX\or CXLD\or CXLD%
\else CANCELLED\fi
\end{center}\vfll}\vss}}}}

\newif\ifxld

\newcommand{\cancel}[3]{\global\xldtrue
\session{#1}{#2}{#3}\global
\xldfalse\ignorespaces}

```

Figure 32: Cancelled Sessions

“`\xentry`” has one argument, the number of time slots. We begin this time with “`\rlap`”, which immediately establishes to point to which `TEX` will return, horizontally, after its work. Next comes the start of a “`\vbox`” as before, with its height set to that of “`\parnbox`”. This time, before we set a width, we start a vertical box with a depth equal to 4 times the current (established by “`\parnbox`”) line height. That’s a vertical box the full height of our cell, which we create because we want to float something vertically to its center. Now we set the width to the cell width, with “`\minwidth`”, and invoke the 16-point font, which we named earlier as “`\cfont`”. “`\vfll`” inserts some vertical filler. Next we establish a horizontal centering environment.

Now we call `TEX`’s case statement with the cell’s number of time slots as its argument. So if we have a 1- or 2-period cell (a quarter-hour or half-hour session) case 1 occurs and we typeset **CX**. If we have a 3- or 4-period cell, case two holds and we get **CXLD**. If we have anything larger, we can fit the full word **CANCELLED** into it. We leave the horizontal centering mode, insert another vertical filler which balances the one just above it, centering our work vertically in the current 4-line deep “`\vbox`”. Then “`\vss`” comes along to spring us back up to the top of the outer, “`\rlap`”ed “`\vbox`”, ready to go back to the job of setting the title and session number in the same cell.

The switch inside the “`\entry`” macro that brought us to the “`\xentry`” macro was “`\ifxld`”. This, using “`\newif`” is how you define a conditional in `TEX`.

One last thing remains. How do we turn on the “`\ifxld`”

conditional? We define here a command called “`\cancel`”, and we must tell the input staff that, if a session is cancelled and a rerun of the SAG showing that fact is wanted, they must change that session’s “`\session`” command to “`\cancel`”. If they do that, then this macro is invoked (with the same 3 arguments—number of time periods, session number, and session title. The first thing “`\cancel`” does is to use `TEX`’s convention to set “`\ifxld`” on by saying (“`\global`”ly) “`\xldtrue`”. Then it issues the “`\session`” command (which resolves to either “`\leftsession`” or “`\rightsession`”), passing its three arguments on, but with the “`\ifxld`” switch turned on, too. Then all the other macros do their work, returning here to turn the “`\ifxld`” switch back off before closing and going on to the next line in the input file.

Use these examples for your own purposes, if you wish. If you need more information about `TEX` or `LATEX`, call the `TEX` Users Group (see Figure 33). Good documentation is available.

TEX USERS GROUP



- TUG publishes a newsletter containing valuable information on extensions to $\text{T}_{\text{E}}\text{X}$ - $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$, $\text{AMST}_{\text{E}}\text{X}$, etc., solutions to typesetting problems, bug fixes, and new developments in typesetting. In addition, it provides information on the latest commercial $\text{T}_{\text{E}}\text{X}$ products and services.
- TUG provides users a way to upgrade skills through seminars and classes for Beginners, Intermediate, and Advanced $\text{T}_{\text{E}}\text{X}$ ers.
- Through its regular meetings and BOFs (Birds of a Feather), TUG brings $\text{T}_{\text{E}}\text{X}$ users together to communicate on various topics of interest.
- TUG provides a centralized source for all $\text{T}_{\text{E}}\text{X}$ and $\text{T}_{\text{E}}\text{X}$ -related publications and videos. It keeps members informed of new publications and other material of interest, and provides an easy way to obtain them.
- The TUG Membership List includes the names of all members (individuals and institutions). Information is provided on users' computer and typesetting equipment to permit easy communication with colleagues.
- TUG maintains a list of Site Coordinators, specialists in implementations of $\text{T}_{\text{E}}\text{X}$ on various hardware, who have agreed to assist new users.

Call or write:

TEX USERS GROUP
Post Office Box 9506
Providence, Rhode Island 02940 U.S.A.
(401)272-9500

Figure 33: The $\text{T}_{\text{E}}\text{X}$ Users Group

The SAG Macros—A Complete listing

```

\documentstyle{article}
\pagestyle{empty}
\textwidth=8.5in      \textheight=10.5in
\topmargin=-1in      \oddsidemargin=-.75in

% This header file is called by symposium.tex, which also calls the saag
% input files participating in a production run. Production is
% initiated by the command "latex symposium".

% set separation between table columns and width of gridlines:

\tabcolsep=1pt      \arrayrulewidth=.8pt

% define some fonts; TINY will have the parameters of "\tiny" and the face
% of "\tphon6":

\font\saag=tphonb6      \newcommand{\TINY}{\tiny\saag}
\font\eightbf=amssbx8   \font\tenssbf=amssbx10
\font\headfont=amssbx12 \font\cxfont=amssbx16

%make "/" an acceptable hyphenation point:

\catcode'\/=13\def/{\char'\/\penalty0\hskip0pt }}

%permit hyphenation of words with uppercase letters

\uchyph=1

% These are macros to produce the "clocks" and other labels that are
% part of the head and foot of right and left pages:

\newdimen\minwidth      \minwidth=.23in
\newdimen\widewidth      \widewidth=\minwidth
\advance\widewidth by \arrayrulewidth

\newcommand{\hour}[1]{\multicolumn{2}{p{2\minwidth}}{\clock{#1}}}
\newcommand{\widehour}[1]{\multicolumn{2}{p{2\widewidth}}{\clock{#1}}}
\newcommand{\clock}[1]{\centering \eightbf #1}
\newcommand{\spacer}{\hspace*{\minwidth}\#}

\newcommand{\leftclock}{\\hline#
\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer
\multicolumn{8}{|c|}{\rule{0pt}{.8em}\eightbf LUNCH}\# \spacer\spacer
\spacer\spacer\spacer\spacer\spacer\spacer\spacer\hspace*{\minwidth}\\
\cline{12-19}\parbox{.75in}{\tenssbf \centering ROOM}%
\hour{9:00 9:30}\hour{9:30 10:00}\hour{10:00 10:30}\hour{10:30 11:00}%
\hour{11:00 11:30}\widehour{11:30 12:00}\widehour{12:00 12:30}%
\widehour{12:30 1:00}\widehour{1:00 1:30}\hour{1:30 2:00}\hour{2:00 2:30}%
\hour{2:30 3:00}\hour{3:00 3:30}\hour{3:30 4:00}}

\newcommand{\rightclock}{\\hline
\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer
\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer
\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer\spacer
\phantom{\eightbf LUNCH}\\
\multicolumn{2}{|p{2\minwidth}}{\clock{4:00 4:30}}%
\hour{4:30 5:00}\hour{5:00 5:30}\hour{5:30 6:00}\hour{6:00 6:30}%
\hour{6:30 7:00}\hour{7:00 7:30}\hour{7:30 8:00}\hour{8:00 8:30}%
\hour{8:30 9:00}\hour{9:00 9:30}\hour{9:30 10:00}\hour{10:00 10:30}%
\hour{10:30 11:00}\# \parbox{.75in}{\tenssbf \centering ROOM}}

% Control comes to this macro whenever the \room macro is encountered in
% left-page input. The \\ ends the previous line and the \hline draws a
% horizontal line across all columns. The \multicolumn spans just one
% column--the "room" column, first on the left-hand page--placing a

```

% vertical on either side of it and centering the room name inside it.
 % \ignorespaces prevents carriage-returns in the input file from be
 % treated as space characters by these macros:

```
\newcommand{\leftroom}[1]{\hline\multicolumn
{1}{p{.75in}}{\raisebox{-2ex}{\parbox
{.75in}{\tenssbf\centering #1}}}\ignorespaces}
```

% The \room macro occurs first, before the \sessions in the room, in the
 % input for both right- and left-hand pages. On left-hand pages the room
 % name is output right away, to be printed at the left of the row. But
 % on right-hand pages, the room name must be stored and saved to be
 % printed at the right end of the row, triggered by the NEXT occurrence
 % of the \room macro. Control comes here at the beginning of right-page
 % input, the first time the \room macro is encountered; it stores the
 % name of the first room for later use by \nextrightroom at the end of
 % the row. Once this is done, \room is redirected to \nextrightroom,
 % and \session and \opentime are set to handle the first session column
 % on the righthand page:

```
\newcommand{\firstrightroom}[1]{\gdef\roomname{#1}\global
\let\session=\firstrightsession
\global\let\opentime=\firstrightopentime
\global\let\room=\nextrightroom\ignorespaces}
```

% control comes here each time \room is encountered, except the first
 % and last; output (by calling \rightroom) the room name stored
 % earlier, at beginning of row, and save the next room's name. Reset
 % \session and \opentime to handle left edge of righthand page:

```
\newcommand{\nextrightroom}[1]{\rightroom\gdef\roomname{#1}\global
\let\session=\firstrightsession
\global\let\opentime=\firstrightopentime\ignorespaces}
```

% control comes here each time \room is encountered, except the first.
 % called from \nextrightroom and at the end of file in \rightpage Begins
 % with table-alignment (column counter) character (#):

```
\newcommand{\rightroom}{#\multicolumn{1}{p{.75in}}{\raisebox
{-2ex}{\parbox{.75in}{\tenssbf\centering \roomname}}}}
```

% used if the first "session" on the row is empty; The mbox is
 % necessary as the "title" (3rd) argument so as to force the correct
 % width:

```
\newcommand{\firstrightopentime}[1]{\firstrightsession{#1}%
{\ }\makebox[#1\widewidth]{}}\ignorespaces}
```

% Control comes here when the first \session or \opentime on each
 % right-page row is encountered. This macro ends the previous row and
 % draws a horizontal line, then draws the left hand vertical, then
 % resets \session and \opentime to use the generic versions of these
 % macros, \leftsession and \leftopentime, which are also used on the
 % lefthand page. Each quarter-hour time-slot is .23in (\minwidth)
 % wide. #1 of \session is the number of quarter-hour periods the session
 % requires, so #1\widewidth sets the correct \parbox width for this
 % session. \entry formats the cell contents. The \hline at the
 % beginning of \firstrightsession causes some extra vertical lines to be
 % drawn for partial pages, which are removed in the stripping process.

```
\newcommand{\firstrightsession}[3]{\hline
\multicolumn{#1}{p{#1\widewidth}}{\entry{#1}{#2}{#3}}\global
\let\session=\leftsession
\global\let\opentime=\leftopentime\ignorespaces}
```

% \leftsession and \leftopentime are the generic session and
 % empty-session macros used by both right- and left-hand pages. They
 % insert a vertical only at the right edge of each column.

```

\newcommand{\leftsession}[3]{#\multicolumn
{#1}{p{#1\linewidth}}{\entry{#1}{#2}{#3}}\ignorespaces}

\newcommand{\leftopentime}[1]{#\multicolumn{#1}{p{#1\linewidth}}{\%
{\makebox[#1\linewidth]{}}\ignorespaces}

% The entry macro formats the contents of each cell. \parenbox holds the height
% of a ( ) in the tphon font. \linewidth is the width of one quarter-hour
% time-slot plus the width of a vertical rule. The \vbox in \entry sets the
% session title (#3), leaving space for the session number (#2). Titles have to
% fit on three lines, plus no more of a fourth than will not overlap the session
% number. If a title is too long, it must be edited and reinput. \vss after
% the \parenbox height of ( ) allows subsequent lines to be set vertically below
% the first without error, beyond the bottom of the box. \rlap sets the \vbox
% rightward from the left edge of the \parbox. The \parbox contents after the
% \vbox right justify the session number on the fourth line.

\newbox\parenbox      \setbox\parenbox=\hbox{\TINY\phantom{()}}

\newcommand{\entry}[3]{\TINY \parbox[t]{#1\linewidth}{\leavevmode
\, \ifxld\xntry{#1}\fi
\rlap{\vbox to \ht\parenbox{\hsize=#1\linewidth
\raggedright #3\ \phantom{#2}\vss}}\null\hfil\break
\null\hfil\break
\null\hfil\break
\null\hfill #2\,}}

\newcommand{\xntry}[1]{\rlap{\vbox to \ht\parenbox{\vbox to 4\baselineskip%
{\hsize=#1\linewidth\cfont\vfill\begin{center}%
\ifcase#1\or CX\or CX\or CXLD\or CXLD \else CANCELLED\fi\end{center}%
\vfill}\vss}}}}

% The \cancel command acts identically to the \session command, except
% that it sets to true the "ifxld" condition, which in turn trips a call
% from the \entry macro to \xntry, overwriting the session with a
% "cancelled" indicator:

\newif\ifxld

\newcommand{\cancel}[3]{\global\xldtrue\session{#1}{#2}{#3}\global
\xldfalse\ignorespaces}

% The \saagpage macro identifies an input file, and must be the first line
% in that file. #1 is the name of the day; #2 identifies the page segment,
% #3 is the season (Spring, Fall), and #4 the year. The page segment
% identifier names a macro which is called to set the proper page segment.
% \tabname creates a label for the output identifying the page segment:

\newcommand{\saagpage}[4]{\gdef\saagday{\uppercase{#1}}\gdef
\meeting{\uppercase{#3 #4}}\csname #2\endcsname
\gdef\tabname{\uppercase{#1}--#2}\ignorespaces}

% The next 4 macros represent the 4 possible left-hand page segments:
% full, top, center, or bot. Each calls an appropriate start-up macro
% for its segment type, and sets \endsaagpage (which must be the last
% line of any input file) to an appropriate meaning:

\newcommand{\leftfullpage}{\openleftpage
\global\let\endsaagpage=\closeleftpage}

\newcommand{\lefttoppage}{\openleftpage
\global\let\endsaagpage=\finishpage}

\newcommand{\leftcenterpage}{\leftpageinit
\leftclock\global\let\endsaagpage=\finishpage}

\newcommand{\leftbotpage}{\leftpageinit

```

```

\global\let\endsaagpage=\closeleftpage}

% \openleftpage begins a left-hand page by creating all its headers,
% including the "clock".

\newcommand{\openleftpage}{\leftpageinit\hline
\multicolumn{29}{|c|}{\headfont\rule[-.4em]{Opt}{1.6em}\hspace*{2em}
\saagday \hfill\meeting\ DECUS U.S. SYMPOSIUM\hspace*{2em}\null}
\leftclock}

% \leftpageinit sets the meaning of the input file macros appropriately
% for a left-hand page, and initializes the table and tabular
% environments:

\newcommand{\leftpageinit}{\global\let\room=\leftroom
\global\let\session=\leftsession
\global\let\opentime=\leftopentime
\begin{table}\begin{tabular}{|p{.75in}|*{28}{p{\minwidth}|}}

% \closeleftpage sets the left "clock" at the bottom of the page and
% completes the page with \finishpage:

\newcommand{\closeleftpage}{\leftclock\finishpage}

% The following series of right-page macros parallel the above series of
% left-page macros:

\newcommand{\rightfullpage}{\openrightpage
\global\let\endsaagpage=\closerightpage}

\newcommand{\righttoppage}{\openrightpage
\global\let\endsaagpage=\finishrightpage}

\newcommand{\rightcenterpage}{\rightpageinit
\rightclock\global\let\endsaagpage=\finishrightpage}

\newcommand{\rightbotpage}{\rightpageinit
\global\let\endsaagpage=\closerightpage}

\newcommand{\openrightpage}{\rightpageinit\hline
\multicolumn{29}{|c|}{\headfont \rule[-.4em]{Opt}{1.6em}
\hspace*{2em} SESSIONS-AT-A-GLANCE \hfill \saagday \hspace*{2em}\null}
\rightclock}

\newcommand{\rightpageinit}{\global\let\room=\firstrightroom
\begin{table}\begin{tabular}{|*{28}{p{\minwidth}|}p{.75in}|}}

\newcommand{\closerightpage}{\rightroom\rightclock\finishpage}

\newcommand{\finishrightpage}{\rightroom\finishpage}

% Finish table and environments, end page, reset some values:

\newcommand{\finishpage}{\\hline\end{tabular}%
\caption{\tabname}\end{table}\clearpage
\gdef\saagday{\null}\gdef\meeting{\null}}

```

C PROGRAM PORTABILITY

Michael Tilson

Human Computing Resources Corp.
10 St. Mary Street
Toronto, Canada M4Y 1P9

416-922-1937
{decvax,uizoo,...}!hcr!hcradm!mike

ABSTRACT

This article discusses aspects of C program portability. Using examples, it is intended to illustrate aspects of the C language that are important for portability and which are easy to overlook if all of one's programming is confined to a VAX.

(Earlier versions of this paper have been published previously. For the DECUS Proceedings, some of the examples have been updated and other revisions have been made.)

Imagine that you have just received a big program from your friend at X University. The program is written in C, and runs on your friend's UNIX system. You also have C and UNIX, so you expect to have no problems, although you do recall that your friend uses a different brand of computer. You read the program off tape with no problem, and compile the program with no error messages. Then you run the program, and get this message:

```
Segmentation violation - core dumped
```

This isn't the desired result!

C LANGUAGE PORTABILITY

This article focuses on machine-independent coding in the C programming language, illustrated by examples. All of the examples have arisen in practice. However, it does not cover a number of related issues, such as portable library routines, operating system interface, or the command interface, although these are also very important. This article assumes a working knowledge of the C programming language, and focuses on some of the less obvious portability problems.

Why do we want portability? Computers are becoming a commodity. When you buy a computer, you want to ask a few questions: Does it run UNIX? (Or your favorite generic portable system.) How fast is it? How reliable is it? How big a program can you run? How much does it cost? You don't really want to ask very many other questions. In particular, you don't want to be forced to continue to buy from the same vendor if another computer is faster and cheaper. We want to reuse our software without any conversion cost. I might also add that portable software tends to be more reliable, since it tends to be less "dirty", and uses fewer "tricks".

At my company, we have a strong commercial interest in portability. Portability is important to many other software companies. As an example, in our case we have supported or developed UNIX versions for the PDP-11, the VAX, the VAX under VMS, the Prime 50 Series, the Control Data Cyber 180, the National 32000 and Motorola 68000 microprocessors, the PERQ workstation, the Computer Automation 4/95, and several other machines we can't talk about. We have also worked with UNIX on several other processors, such as the 8086 and Z8000. We sell UNIX software products on these and other machines. We estimate that non-portable software has cost us well over \$100,000 in the last year. (The first version of this article was written several years ago, but the number never seems to go down.)

LEVELS OF PORTABILITY

There are two possible kinds of portability. The first level of portability produces programs which are completely machine independent, completely type safe and size safe. The second kind of portability involves portability of resulting binary files from one "reasonable" machine to another, for example between machines which have 8 bit characters, 16 bit short integers, and 32 bit long integers. For example, you might have a

C cross compiler for the Intel 8086 which initially ran on a PDP-11, got moved to a VAX, and then got moved to the 8086 itself. This kind of program may not be fully portable (since it generates machine code), but can still be made easy to move over an important class of machines.

PORTABILITY PROBLEMS IN C

C is not a safe language. The original Kernighan and Ritchie reference book **The C Programming Language** describes a number of portability problems. To avoid portability problems, you must be familiar with all of the footnotes in that book. Once the ANSI X3J11 C standard is published, a more complete reference will be available, but the "footnotes" will be just as important.

Problems commonly arise from the careless use of pointers and the mismatching of types, from assumptions about the sizes (in bits) of various types, and from alignment assumptions (such as the order of chars within a short int.) Type casts and the UNIX "lint" program checker don't solve all problems. Type casting can make lint shut up without solving the portability problem. There is no substitute for getting it right.

C has been described as a "high level assembler". While this is an exaggeration, it is true that many C programmers visualize the resulting compiled code as they write C programs. This may result in greater efficiency on some machines, but it does not make for portable software.

EXAMPLES

The remainder of the article will focus on actual programming examples which have arisen in HCR's work. This is not an exhaustive list of what can go wrong, but rather an illustration of the care which you must take when writing portable programs. I should add that many of the examples are "lintable", at least with the default level of checking provided by some versions of lint. (Newer versions will catch more of the errors shown here.) In the following examples, you will see calls to "abort". On the machines most people are used to, "abort" will not be called. However, for each example there are machines for which the "abort" call will happen.

Example 1:

```
int *p;
char *q, *q2;
...
q2 = q;
p = (int *)q;
q = (char *)p;
if(q != q2)
  abort();
```


The above can fail because there is no guarantee that the conversion of a pointer from "char *" to "int *" will preserve all of the bits of precision if the "char *" had not already been aligned to an "int" boundary. The above works on the VAX, PDP-11, and most other machines, but will fail on on a number of other (typically word-addressed) machines. This kind of type mismatch is common in UNIX software.

Example 2:

```
int *p;
char *q;
int arr[50];

p = &arr[20];
q = (char *) &arr[0];
q++;
if(p < (int *)q)
    abort();
```

How can this fail? Variable `p` is clearly at a higher memory address than `q`. No it isn't. The `q++` could produce an internal bit representation which, if considered as an "int *" quantity, would look like a word pointer at a larger memory address. This example is adapted from the UNIX shell command interpreter, and we first saw it on the CA 4/95.

Example 3:

```
int *a, *b;

a = (int *)sbrk(0);
b = a;
b--;
if(a < b)
    abort();
```

What could be wrong with this? On a segmented architecture, the UNIX memory allocator `sbrk` might return an address at the start of a memory segment. Pointer arithmetic is only valid if the pointers remain within a known contiguous and properly aligned storage structure. The `b--` could wrap around to the high end of a memory segment. This example comes from a version of the UNIX text editor.

Example 4:

```
execl("/bin/echo", "echo", "hello", 0);
```

This error is from many versions of the UNIX Programmer's Manual. The `execl` system call takes a list of character strings, terminated by a null pointer. The constant "0" is guaranteed to be a valid null pointer if used in an expression. However, C does no type checking across function calls. On a machine with 32 bit pointers and 16 bit integers, the above could cause trouble. It would be better to use:

```
execl("/bin/echo", "echo", "hello", (char *)0);
```

Note that on the popular Intel 8086 architecture, it makes good sense to implement 16-bit integers and 32-bit pointers.

Example 5:

```
char *p = (char *) 0;

if( *p != 0)
    abort();
```

This is another common problem with "null" pointers. The pointer `p` is a null pointer. It is not valid to assume that a null pointer in turn points to a zero quantity, although this happens to work on many UNIX implementations. In fact, the VAX and PDP-11 implementations go out of their way to make this work. But this is highly non-portable, and will cause no end of trouble on other architectures.

Example 6:

```
int x;

x = 70000;
if(x != 70000)
    abort();
```

This is simple. The above works on a VAX, but not on the PDP-11, because the number 70000 fits in an "int" on the VAX, but not on the 11.

Moral: always explicitly specify "long" if a number is going to be larger than somewhat. (A reasonable "somewhat" is 32767.)

Example 7:

```
char *p, *q;
int n;

n = (int)p;
q = (char *)n;
if(p != q)
    abort();
```

The C language guarantees that there is a large enough int to hold the bit representation of a pointer. However, on a machine with 32 bit pointers and 16 bit ints, the above will fail because the conversion to int will overflow. Conversion to anything other than "long" is not portable. (By the way, conversion of pointers to integers is in general a bad idea, and is always non-portable if the value of the "int" is ever examined. The only safe thing to do is convert back to the same kind of pointer.)

Example 8:

[Assume this is running on a machine with a large (e.g. megabytes), contiguous, uniform address space.]

```
char *p, *q;
long int n;

n = 0;
q = p;
for(i=0; i<5; i++) {
    q += 32000;
    n += 32000;
}
if((q-p) != n)
    abort();
```

The subtraction of two pointers is defined by the manual to yield "int", and not necessarily "long int". Once again, int might not be large enough to hold the difference. This might be an unfortunate restriction in some C compiler, but it is dictated by the current C definition.

Example 9:

```
struct {
    short int a_magic;
    long int a_text;
    long int a_data;
} header;

(void) write(f, (char *)&header, sizeof header);
```

What could be wrong with that? Everything is nicely type cast, and even the return value from `write` is fastidiously voided. The problem is that a binary structure is being communicated to the outside world via the UNIX `write` system call. This always indicates a possible portability problem, since the resulting file can't be moved from machine to machine. Increasingly, we will see networks of dissimilar UNIX machines, but with network-wide file systems. One should be careful about producing binary files, and one should never assume that they will be portable.

Example 10:

```
long int n;
printf("%d\n", n);
```

This example is the bane of everyone who has ever had to move programs from the VAX to the PDP-11. The "%d" prints an "int", which on the VAX is the same as "long int", but not on the PDP-11. This is an example of the more general case of function argument mismatch, but one which is not checked by "lint" (although it really should be.)

Example 11:

```
double x = 1.234e30;
printf("%d\n", x);
```

This is an example of a common programming error. A similar example was used in a very glossy advertising brochure sent out by a company promoting its C training courses. It looks like the programmer wanted to

print the number as decimal digits truncated or rounded to an integer. The fix proposed by the glossy brochure was as follows:

```
printf("%d\n", (int)x);
```

On the VAX, the above "fix" prints "0", rather than the correct number. On other machines, an overflow exception might be generated. This is an example of throwing in a type cast to patch a problem, rather than getting it right. Here a type cast was used to try an correct a datatype mismatch error. In other cases casts are used to remove portability problems. In all cases there is no substitute for simply making things match up correctly in the first place. Here is a much better solution:

```
printf("%.0f\n", x);
```

which prints a floating point number correctly rounded to the nearest integer.

Example 12:

```
unsigned char c;

c = '0' - 1;          /* the character before '0' */
if( (c - '0') < 9 )
    abort(); /* called only if "c" is a digit */
```

This example is supposed to check whether `c` is in the range '0' to '9'. With many UNIX compilers, this works because `c` is unsigned. If `c` is greater than '9', everything works as you might expect. If `c` is less than '9', you get a negative number. However, because the variable is "unsigned", the entire expression is taken as unsigned, and the negative number is taken to be a very large positive number. This is an old assembly programmer's trick to check both ends of a 0-based range in one operation.

However, some compilers use "value-preserving" rules. Under such rules, `c` would be widened to the next larger "int" big enough to hold all possible values of `c`, and the conversion to unsigned would never occur. In general, it is bad to rely upon the esoteric details of automatic type conversion in an expression. A better solution would be:

```
if( (unsigned)(c - '0') < 9 )
    abort();
```

Of course, you could forget trickery altogether and use something like:

```
if( isdigit(c) )
    abort();
```

Example 13:

```
char *filename;
filename = mktemp("/tmp/aXXXXXX");
```

This is an example that is straight from the UNIX reference manuals, and seemingly confirmed by the AT&T System V Interface Definition. Surely it can't be a problem? Well it is. The `mktemp` routine modifies its argument string. If you want to be portable, you should not assume that string constants ("/tmp/aXXXXXX") can be modified. A better solution would be:

```
char filestr[80];
char *filename;

filename = mktemp(strncpy(filestr, "/tmp/aXXXXXX"));
```

This solution also has the advantage that the code can be executed more than once, since the string constant hasn't been destroyed.

Example 14:

```
to = bp->b_ptr;
asm("movc3 r8,(r11),(r7)");
bp->b_ptr += put;
```

This bit of program was supplied by a certain educational institution which is well known for its extensive UNIX modifications. This program fragment illustrates the extreme case of non-portability. "Asm" is a **key-word** in the UNIX "portable" C compiler. It causes the string argument to be emitted into the assembly source which results from compiling the program. In this example, there is no comment to say what is going on. The programmer **knows** what C variables are in particular registers. (The variable "to" is one of the registers used in this VAX assembly instruction. You guess which.)

If you feel you **must** write such "efficient" code, at least write it a bit more cleanly:

```
#ifdef vax_with_certain_compiler_version
    asm("movc3 r8,(r11),(r7)");
#else
    --- insert portable C equivalent here ---
#endif
```

Example 15:

```
int TheQuickBrownFox;
int TheQuickGreyFox;
```

This causes no end of trouble when moving code from VAX UNIX systems (which typically allow this) to other systems, which sometimes don't. While meaningful variable names are laudable, names which are not unique in the first seven characters are not portable to other UNIX systems. (On non-UNIX systems, you might even be restricted to six characters.) With a little thought, you can always choose nice names which are also portable:

```
int TheBrownQuickFox;
int TheGreyQuickFox;
```

CONCLUSION

These examples are only an indication of the kinds of portability problems are often found in real C programs. You can avoid problems by viewing the machine as an abstract entity, and not making assumptions that depend upon actual bit representations. With a little bit of care, your software will be usable on a wide range of machines. If your software is worth using more than once, then it should be portable, since it's worth using on more than one machine.



NETWORKS SIG

A HIGH SPEED LOCAL AREA COMPUTER NETWORK ACROSS
THE GODDARD SPACE FLIGHT CENTER

James P. Gary
Kenneth E. Lehtonen
William H. Mish

Space and Earth Sciences Directorate
NASA Goddard Space Flight Center
Greenbelt, Maryland 20771

Edward D. Rothe
Michael C. Spinolo
Marc Peters

Mission Operations and Data Systems Directorate
NASA Goddard Space Flight Center
Greenbelt, Maryland 20771

ABSTRACT

This paper describes a high speed, hybrid baseband/broadband local area communications network used primarily for heterogeneous computer-to-computer networking now operating at the NASA Goddard Space Flight Center. This network uses separate Ethernet/IEEE 802.3 baseband segments for interconnecting computers within each of approximately 20 major GSFC buildings. These buildings are typically separated from one another by distances of 100 to 1000 meters. High speed inter-building communications are provided via a CCTV/CATV industry standard digital broadband subsystem using standard 6 mhz frequency channels. Data link layer bridging between the baseband and broadband local area network technologies is provided via Applitek Corp. NI10/E Ethernet Bridges located in each building.

This Center-wide network concurrently and transparently supports the high level host-to-host network protocols DECnet, XNS, and TCP/IP on Ethernet. Packets are encapsulated within Applitek's UniLINK slotted time division multiple access message format for broadband transmissions. Nodes presently functioning on the network include various DEC superminicomputers, an IBM 3081, numerous IBM PC'S, and various vendor's Unix work stations. The IBM 3081 mainframe is interfaced to the high speed network via an Interlink IBMmv/DECnet Gateway that allows the IBM 3081 to participate as a peer node within a DECnet network. The local network includes interfaces with the ARPAnet and other wide area networks.

This paper includes a description of the local network's design rationale, and an indication of throughput performance results derived from various memory-to-memory and disk-to-disk data transfer tests conducted simultaneously among multiple pairs of the end user computer nodes.

1.0 INTRODUCTION

The Goddard Space Flight Center (GSFC) is a major center of the National Aeronautics and Space Administration (NASA) with responsibilities in advancing space and Earth science research and in developing and managing a complex set of technical resources to enable effective analysis of space-based observations. The main portion of the GSFC is located in Greenbelt, Maryland on a 550 acre campus with over 30 major buildings (see Figure 1)

housing approximately 6000 on-site civil servant and contractor scientists and engineers. The GSFC is organizationally structured primarily according to special scientific and engineering disciplines, but there are also formal organizations for projects and extensive formal and informal matrix management.

Because of the complexity of its numerous scientific and engineering research activities, the GSFC already has substantial existing computing resources installed at its Greenbelt site and is continuously

augmenting these capabilities to meet new requirements. A partial list of the present, locally installed, major computer-related resources which are applied only in the space and Earth science research program (not included in this network are the extensive systems primarily providing in-line operational support for flight missions) includes:

- o High speed computation: 1 CDC CYBER 205; 1 IBM 3081-K; 1 Amdahl 470 V/6-II; 1 Amdahl 470 V/7B; a number of commercially available minicomputer-attached array processors such as the FPS 180V; and a specially designed attached array unit with 128 x 128 processing element;

- o Interactive data analysis including advanced image processing and display: 2 DEC VAX 8600/8650s; about 50 DEC VAX-11/780s, 750s, and MicroVAXs; and about 20 other superminis and user workstations from HP, SUN, Masscomp, etc.;

- o On-line mass storage: IBM 3850 (220 gbytes); Masstor M860 (110 gbytes); Shugart Optimum 1000 optical disk (2 gbytes); plus active magnetic tape libraries with several tens of thousands of tape reels.

These systems are variously configured, are mostly located in different buildings from one another, and are frequently under the management cognizance of different organizations internal to the GSFC.

The need for an integrated communications network to support the local networking of these computers had been developing at GSFC for several years. Such needs at GSFC had in the past been partially satisfied, as at other sites, by point-to-point links of various kinds, usually using coaxial cable or twisted pair wiring. With the inherent limitations of such methods becoming an increasing liability with the number of potential nodes, alternative networking topologies and technologies were needed. The present GSFC local area computer network (LACN) developed out of an extensive series of studies (Berman, 1982; Rebibo, 1982; Lehtonen, 1983; Mish, 1983) and subsequent discussions focused on providing a packet oriented multiple access bus to transparently support higher layer peer-to-peer computer networking protocols among various computers throughout the GSFC, to do so at relatively low cost, and to provide throughput performance at least as good as point-to-point techniques. GSFC also undertook the implementation of the GSFC LACN to further enhance the productivity of the scientific and engineering users of GSFC's presently installed computers and to enable more flexible options in planning for the acquisition of additional computer systems capabilities (Gary and Rothe, 1985; Halem et al., 1985).

Concurrent with the ongoing implementation of the high speed GSFC LACN, GSFC has planned to install a digital PABX to replace the existing voice telephone exchange and to satisfy a significant amount of Center-wide terminal-to-host connectivity. A Rolm CBX-II has been procured for this purpose and "cut in" use of the CBX-II is planned to start in May 1986. While the CBX-II is expected to handle all voice communications in a non-blocking mode, data transfers through the CBX-II will be limited to a maximum of 19.2 kbps asynchronous and 56 kbps synchronous communications. The computer network

which is the focus of this paper is targeted at higher speed host-to-host communications requirements which cannot be met through the use of the CBX-II.

This paper provides a description of the overall design and present state of GSFC's LACN, and is organized as follows:

Section 2 summarizes the network design objectives while Section 3 overviews elements of the design itself. Sections 4 and 5 provide more descriptive information about GSFC's use of the relatively new NI10/E Ethernet Bridge and IBMmv/DECnet Gateway products from Applitek Corporation, Inc. and Interlink Computer Sciences, Inc., respectively. Section 6 overviews the present network configuration. Section 7 presents both the results compiled to date and a description of the benchmark tests planned for assessing communications subnet and end-to-end computer network throughput performance in various types of computer-to-computer data communications situations. Section 8 contains a brief indication of some follow-on GSFC LACN plans.

2.0 NETWORK DESIGN OBJECTIVES

The design of GSFC's LACN has taken numerous interrelated objectives into consideration. A brief description of the key objectives is provided in the following subsections.

2.1 End Usage

The general goal of GSFC's LACN is to provide integrated, low cost, high speed intercommunications among a heterogeneous set of scientific computers, peripherals, and terminal/workstations at GSFC and to provide suitable interfaces for links to remote hosts and other networks such as the ARPANet and NASA's Space Physics Analysis Network (SPAN) (Green and Peters, 1985). GSFC scientific users frequently use a local VAX or advanced user workstation to interactively display data and extract information parameters; however, they also need high speed access to GSFC's large mainframes for production data processing of large data bases and computing numerical models of the physical phenomena being studied. Powerful desktop image display capabilities are also beginning to be acquired for local image processing. These image display workstations need high speed access to the larger superminis and mainframes to share resources such as database files, disk space, high speed printer/plotters, tape drives, and array processors. The intent of GSFC's high speed LACN is to substantially increase the productivity of the users of the separate computer systems by improving the users' access to all computer resources through a high performance interconnection of the individual computers.

A partial list of the initial set of major computers to be interconnected via the high speed GSFC LACN is shown in Figure 2. This list also identifies the GSFC internal discipline-oriented organizations responsible for managing these computers and the separate GSFC locations where these computers are presently installed. In addition to this initial set of computers, about 50 others are planned to be added to the high speed GSFC LACN by the end of 1986. Additionally, many smaller personal computers

(PCs), advanced user workstations, and terminal servers have already interfaced to the GSFC LACN and many more are also planned.

2.2 Network Functionality and Performance Requirements

2.2.1 High Speed Communications Subnet Connectivity and Expansion

One requirement of GSFC's LACN has been a communications subnet which enables relatively simple high speed connectivity among all of the various types of computers whose managers' chose to participate in the network. Additionally, the communications subnet is required to support several higher layer communications protocols, such as Digital Equipment Corporation (DEC)'s DECnet Phase IV, the ARPAnet-related Transmission Control Protocol/Internet Protocol (TCP/IP) set, and Xerox's XNS protocols, to enable the full range of network application services needed by GSFC's end users. Furthermore, the communications subnet's design and implementation have had to anticipate and easily allow for substantial connectivity growth, i.e., the addition of many new computers, without changing the overall architectural design initially installed.

2.2.2 Network Applications Support

General applications to be enabled by GSFC's LACN have included inter-computer file transfer, remote batch job submission, electronic mail, remote logons or virtual terminal support, and in some cases task-to-task communications. Also, ideally, the user interface for use of the network-wide applications services has needed to be closely compatible with the operating system command syntax of the host systems with which users were already familiar or had to be very easy to learn. Unfortunately, at present, no single set of application interconnection protocols has existed to satisfactorily accomplish all of these ultimate goals among the full heterogeneous mix of computer systems needing to be networked at GSFC. Hence, for the present, GSFC has planned to concurrently support a limited number of vendor dependent and independent network protocols among overlapping key subsets of GSFC's local computer systems.

Since a large fraction of GSFC's local computers was built by DEC, support of DECnet has been an early implementation requirement for GSFC's LACN. Furthermore, to facilitate DECnet-based users' access to the mainframe computers of the NSESCC (see Figure 2), another major requirement has been the installation of a high speed gateway capability to enable a substantial subset of DECnet's network services to be exercised with NSESCC's IBM 3081 virtually functioning as a normal DECnet node/host (Mish *et al.*, 1985). Specific network/user services required include:

- o DECnet-based user initiation of a file transfer to/from the IBM 3081;
- o IBM 3081-based user initiation of a file transfer to/from a DEC host;
- o automatic data format conversion of I*2, I*4, R*4, R*8 and character data files between a DEC host and the IBM 3081;

- o DECnet-based user initiation of a batch job submission to the IBM 3081;
- o IBM 3081-based user initiation of a batch job submission to a DEC host; and
- o a rudimentary mail facility between a DEC host and the IBM 3081.

DECnet-based user access to NSESCC's CYBER 205 and Amdahls is also required, but initially will be enabled indirectly and transparently through NSESCC's IBM 3081 via inter-mainframe data communications links using IBM's RSCS and NJE protocols across multiple intra-NSESCC channel-to-channel adaptors and 19.2 kbps and higher point-to-point lines.

Before the rapid emergence of advanced user workstation capabilities, only a few isolated Unix-based workstations existed at GSFC. Recently, however, broad interest has developed both for ARPAnet access and for local computer network applications support performed under control of the TCP/IP set. The minimum set of TCP/IP and related network application services required to be supported by GSFC's LACN include:

- o Internet Protocol (IP) MIL-STD-1777;
- o Transmission Control Protocol (TCP) MIL-STD-1778;
- o File Transfer Protocol (FTP) MIL-STD-1780;
- o Simple Mail Transfer Protocol (SMTP) MIL-STD-1781;
- o Telnet Protocol and Options (TELNET) MIL-STD-1782.

2.2.3 Summary Throughput Performance Requirements

The bandwidth requirements of GSFC's LACN vary largely depending on the end user applications. In general, however, the network has had immediate needs to enable end-to-end, disk-to-disk data transfers between interconnected VAX-class hosts at rates in the range of 250 kbps to several megabits per second. Furthermore, ten or more concurrent transfers at these rates need to be supported by the communications subnet. Also, three or more concurrent transfers at these rates need to be supported individually by each host's high speed interface with the communications subnet.

2.3 Architectural Development Guidelines

Design, engineering, and implementation requirements of GSFC's LACN are described below. In general, these requirements have included compatibility with appropriate international, national, and industry standards and the planned use of easy-to-extend off-the-shelf technologies rather than the in-house development and installation of special purpose solutions.

2.3.1 ISO OSI Reference Model

General compliance with the International Standards Organization (ISO) reference model for Open Systems Interconnection (OSI) has been a design guideline for GSFC's LACN.

2.3.2 IEEE 802 Local Area Network (LAN) Standards

Institute of Electrical and Electronics Engineers (IEEE) standards have been used whenever possible (The Ethernet, 1982; IEEE 802, 1984). However, considerations of the need for a low cost solution, relatively large local area distances to be covered, and on-shelf availability of suitable computer interfaces and communications software to support high speed end user applications across a network of non-homogeneous computers have prevented the simple identification and selection of a single standard technology or single vendor product line in meeting all of GSFC's requirements.

Ethernet/IEEE 802.3 technologies have been preferred in implementing intra-building networks. This technology is a widely accepted communications medium and data link protocol for providing high speed data exchange among computers and other digital devices located within a moderately-sized geographic area. This technology uses commonly available coaxial or fiber optic cables and a data link protocol referred to as carrier sensed multiple access with collision detection (CSMA/CD). This technology has the advantage that it is relatively low cost to acquire and install, it is very reliable, and a large number of computer system suppliers have provided a wide range of devices that support it. Unfortunately, its design limitations include a maximum total length among the longest path of 2800 meters achievable when using fiber optic cables between repeaters and counting transceiver cable lengths.

A broadband LAN, similar to IEEE 802.4 token bus or 802.5 token ring standards and based on Community Antenna Television (CATV) component technologies, could easily meet GSFC's LACN distance requirements and potentially could take advantage of an extensive already-underground coaxial cable plant at GSFC which had been used previously only for closed circuit television (CCTV). This cable plant, representing an investment of approximately \$1M, includes cables extending from building 8 into nearly every other building at GSFC. Major limitations, however, which have prevented GSFC's LACN implementation solely with 802.4/802.5-based technology have included:

- o limited speed in available computer interfaces to 802.4/802.5-based LAN technologies;
- o relatively high cost to engineer and install 802.4/802.5-based LANs to a growing, non-predetermined number of locations requiring connectivity; and
- o relatively high cost to implement a 802.4/802.5-based LAN which would have minimum impact on on-going operations as new users were installed.

2.3.3 Higher Layer Interconnection Functionality

While a high speed communications subnet is an essential component of any high throughput computer network, the subnet alone is by no means sufficient to providing functional user services across the network. Depending on the computers among which communications are required, the network application services required, and the throughput performance required, the existence of generally computer resident higher layer service protocols and their

operability with the communications subnet are also necessary facets to providing the network application services required by end users. Draft specifications of some vendor-independent higher layer protocols are in preparation and/or review by the ISO and a number of other organizations responsible for establishing standards. However, until the implementation of these protocols matures to the point of meeting all other GSFC LACN requirements, an operational requirement of GSFC's LACN will continue to include concurrent support for various vendors' products.

3.0 INITIAL GSFC LACN SYSTEM DESIGNS

Numerous options were studied in planning GSFC's LACN. Ultimately, a hybrid design incorporating a unified mix of baseband and broadband LAN technologies was selected for implementing an initial operational capability. Key elements of that initial design are summarized in the subsections that follow.

3.1 Architectural Overview

The architectural reference model used in planning GSFC's initial LACN implementation is provided in Figure 3. This figure illustrates many important architectural design decisions related to GSFC's LACN. These design decisions include:

- o the planned hybrid mix of IEEE 802.3 (baseband) LANs enabling individual (and intra-building) host connectivity and CCTV/CATV industry standard digital broadband LAN technology interconnecting the baseband segments;
- o the functional separation of the communications subnet from the higher layer protocols;
- o communications subnet support for the separate yet concurrent operations of multiple higher layer protocols among logically separate sets of computers; and
- o the possible participation of a single host in multiple logically independent networks enabled by different higher layer protocols co-residing on the host. (Such co-residence does not imply gatewaying even though this would be desirable in a number of instances.)

3.2 Use of Ethernet

Essentially all intra-building connectivity among local computer systems has been accomplished through the use of Ethernet/IEEE 802.3 compatible LANs. Selection of this technology for intra-building installation has provided low cost, highly reliable operation even in the face of irregularly scheduled additions of new hosts to the LACN. Full connectivity with all other hosts is enabled simply through a host's single Ethernet connection to the LACN.

Multi-segment Ethernets have been installed to date in 12 separate GSFC buildings. Several more are planned before the end of 1986. Additional information on GSFC's use of Ethernet is provided in Appendix A.

3.3 Use of CATV

Inter-building communications among GSFC's intra-building Ethernets has been implemented primarily through the use of a digital broadband LAN generally compatible with CCTV/CATV industry standards and the use of commercially new broadband/baseband bridging units (see Section 3.4). The broadband LAN has been implemented at GSFC on already underground coaxial cables formerly intended only for CCTV. The extent of the underground coaxial cable plant at GSFC is illustrated in Figure 4. Additional information on GSFC's use of the CATV cables for the LACN is provided in Appendix A.

3.4 Bridging the Ethernet and CATV LANs

High speed internetwork connectivity between GSFC's inter-building broadband LAN and GSFC's numerous intra-building baseband Ethernets has been accomplished through the use of NI10/E Ethernet Bridges acquired from Applitek Corporation, Inc. (Applitek, 1985 a through g; Dahod, 1983 a and b). Such high speed bridging units are a relatively new LAN technology, but their use is growing rapidly nationwide. Because of pressing needs to fulfill its long standing requirements, GSFC was very active in designing elements of and promoting the original development of this new type of LAN technology. Furthermore, GSFC eventually served as a beta test site for the Applitek NI10/E units which subsequently were the first such units to be offered commercially. In brief, the bridging units used in GSFC's LACN:

- o use standard transceivers to connect to the intra-building Ethernets;
- o acquire all Ethernet packets transmitted by other devices on the local Ethernet;
- o automatically generate address lists identifying the devices on the local Ethernet;
- o encapsulate Ethernet packets destined for other Ethernet segments and broadcast them on the broadband LAN to other bridging units; and
- o acquire all encapsulated messages from the broadband LAN but retransmit onto the local Ethernet only those packets which are destined for devices on that local Ethernet.

Salient designs of the Applitek NI10/E's which have made their use particularly attractive in GSFC's LACN include:

- o the use of 10 mbps modems operating in the 6 mhz channels of the broadband LAN;
- o the use on the broadband LAN of a message slotted data link layer protocol called UniLINK which allows either user preselected or load dependent message slot allocations, thereby enabling the broadband LAN to perform similar to either a token bus LAN operating with deterministic performance or a CSMA LAN operating with on-demand performance; and
- o transparency to higher level protocols.

Further descriptive information on these bridging

units and their use in GSFC's LACN is provided in Section 4.

3.5 Higher Layer Interconnection Protocols

To satisfy the immediate and broad needs of a large number of end users, GSFC's LACN at present uses several different integrated sets of higher layer application protocols. Each protocol set performs many internal functions which ultimately enable the integrated set to conduct both generic and specific forms of task-to-task communications among even dissimilar computers so long as those computers are also hosting a peer-compatible set of the same higher layer protocols. Each protocol set essentially enables the independent operation of a computer network which is logically separate from a computer network enabled by a different protocol set. Sometimes, however, two or more different protocol sets co-reside on a single host computer to enable that host to concurrently participate in the respective logically separate networks enabled by the different protocol sets. Such co-residence, however, does not in and of itself enable internetwork gatewaying. Sets of higher layer computer-to-computer communications protocols already functioning at high speeds in GSFC's LACN include:

- o DEC's DECnet operating among GSFC local DEC VAX 8600's, VAX-11's, microVAX's, and PDP-11's;
- o the TCP/IP-related set of ARPAnet protocols (see Section 2.2.2) operating primarily among GSFC local Unix-based user workstations but also operating among a few DEC VMS-based systems; and
- o XNS-related intercomputer protocols operating primarily among GSFC local PC's with direct Ethernet connections.

These computer-to-computer communications protocols generally enable a full range of network applications including inter-computer file transfer, remote batch job submission, electronic mail, remote logons or virtual terminal support, and in most cases task-to-task communications. Additionally, GSFC's LACN supports several sets of terminal/file/print server protocols. These include:

- o DEC's LAT-11 terminal server protocol operating among GSFC local DECserver 100's and various DEC minicomputers;
- o SNA-related terminal server protocols operating among Ethernet server devices, such as Bridge Communications' CS/1-SNAs, which enable IBM 3278 display station emulation and IBM host connectivity for approximately 200 asynchronous terminals; and
- o XNS-related terminals/file/print server protocols operating among GSFC local Ethernet server devices which enable either generic or vendor specific terminal/file/print server-to-host connectivity for approximately 200 asynchronous terminals, 10 file servers, and 10 print servers.

Also, GSFC's LACN includes a number of Ethernet communications server devices which are dedicated solely to performing remote packet communications routing. These include:

- o DEC DECnet Router Servers used to interconnect GSFC's LACN with remote DECnet nodes via the wide area NASA SPAN; and
- o TCP/IP bridges, particularly Vitalink Communications Corporation's TransLANs, used to interconnect GSFC's LACN with the ARPAnet via a bridging link with the University of Maryland to an ARPAnet IMP.

Approximate counts, by networking protocol set type, of the number of host computers and terminal/file/print server devices which either are currently on-line to GSFC's LACN or are expected to be added by the end of 1986 are:

<u>NETWORKING PROTOCOL SET</u>	<u>GSFC LOCAL HOSTS/DEVICES CURRENTLY ON-LINE TO LACN</u>	<u>ADDITIONAL UNITS EXPECTED BY END OF 1986</u>
o Computer-to-Computer		
- DECnet	27	50
- TCP/IP	13	10
- XNS	100	100
o Terminal/File/Print Server-to-Host		
- LAT-11	5	5
- SNA	2	2
- XNS	15	15
o Packet Communications Routing		
- DECnet	0	2
- TCP/IP	1	2

3.6 DECnet Gateway to IBM Mainframes

Because of GSFC's need to enable high speed DECnet-based users' access to the mainframe computers of the NSESCC (see section 2.2.2), GSFC has directly interconnected NSESCC's IBM 3081K with GSFC's LACN via an Interlink Computer Science's IBMmvs/DECnet Gateway (Interlink, 1986 a through f; Lehtonen and Gary, 1986; Mish *et al.*, 1985). High speed performance and functional requirements which have been met via this Gateway Interlink, include:

- o Transfer of files between the IBM mainframe and a DEC superminicomputer at rates exceeding 200 kbps while performing CHARACTER data format conversions between the native mode representations of such data in the respective hosts;
- o Concurrent support for up to 16 separate network application sessions;
- o Record level access to variously organized (e.g., ISAM, QSAM, VSAM) cataloged data sets on the IBM host; and
- o Use of native host command languages, respectively for both DEC-based and IBM-based users, to perform all supported network applications.

Further descriptive information on this Gateway and its use in GSFC's LACN is provided in Section 5.

4.0 THE APPLITEK NI10/E ETHERNET BRIDGE

4.1 Hardware

The NI10/x series of networking devices from Applitek use a modular design utilizing the IEEE 796 Multibus (Applitek, 1985a). A NI10 unit consists of four modules:

- The Media Access Unit
- The Network Controller
- The Subscriber Processor
- The Device Interface

Discussed here are features of the NI10/E Ethernet Bridge, configured as it is used in the GSFC LACN (see Figure 5).

4.1.1 The Media Access Unit

The Media Access Unit (MAU) provides physical access to the chosen transmission medium for UniLINK. The media supported are broadband or baseband coaxial cable, and optical fiber cable. At GSFC, broadband CATV cable is accessed via frequency agile (20 to 375 mhz) Quadrature Phase Shift Keyed (QPSK) modems occupying a 6 mhz CATV channel. This keying technique yields 1.67 bits/hz with a BER 10 e-8. Currently one channel is utilized by the hybrid LACN - 65.75 mhz reverse/257 mhz forward.

4.1.2 The Network Controller

The Network Controller is independent of the type Media Access Unit and the Device Interface used in the NI10. The Network Controller, together with its Rx/Tx State Machine, implements the chosen UniLINK access method and controls access to the cable. The UniLINK access method is controlled by software commands, which are discussed in 4.2. The Rx/Tx State Machine controls access at the bit level and converts parallel data streams to serial for the Media Access Unit and vice versa.

4.1.3 The Subscriber Processor

The Subscriber Processor is the primary control board of the NI10. Its design is based on the MC68000 microprocessor (MPU). The Subscriber Processor controls the flow of data between the Device Interface and the Network Controller, provides the user interface to the NI10 network commands, and controls a 3.5" floppy disk drive. The power up configuration, a field support history file, and operating software for the NI10 are stored on the floppy disk. Also contained on the Subscriber Processor are a watch dog timer, to prevent the system from hanging up due to a software failure, and a real-time clock, to provide overall system timing.

4.1.4 The Device Interface

A variety of Device Interfaces is being developed for the NI10 including RS 232c (async, bisync), RS 449, T-1, DR11 (Unibus interface) and Ethernet. At GSFC, the interface is the Ethernet via an Ethernet controller board with its resident software. At present, the Communication Machinery Corporation's

(CMC) Ethernet Node Processor (ENP) is implemented with the IEEE 796 Multibus (Communication Machinery Corp., 1984). The architecture of the ENP consists of a MC68000 MPU performing supervisory functions over a LANCE VLSI Ethernet Family Controller, a closely coupled RAM, ROM for on board monitor and control software, and a IEEE 796 Multibus interface for connection to the other modules of the NI10/E. Applitek has announced development of an Ethernet controller which will be more specialized for the services of the NI10/E.

4.2 Software

Control of the NI10 is distributed between the hardware modules described in the previous section and is a combination of firmware, on-board software, and downloaded software.

4.2.1 Media Access Unit and Network Controller

Due to the redundancy of and the high performance required by the functions of the Media Access Unit and the Network Controller, these functions are performed in firmware implemented on each of these boards. These functions include UniLINK network timing, data link services and actual signaling on the transmission media.

4.2.2 Subscriber Processor

An independent copy of the pSOS-68k Operating System resides on each Subscriber Processor board. Control software is downloaded from the 3.5" floppy drive when the NI10 is powered up or reset by the network manager. Downloaded software includes UniLINK Tx and Rx RAM server processes, startup configuration, and NI10 control mode software.

4.2.3 Device Interface: Ethernet Controller

Communication between the ENP and the Ethernet physical medium is handled by the ENP Kernel software, in ROM, and the LANCE device. Communication between the ENP and the Subscriber Processor is handled by software downloaded to the Ethernet controller from the floppy disk. The NI10/E bridges filter Ethernet packets to insure that only the packets that need to be bridged are broadcast by or accepted from the UniLINK subnet. This is accomplished through a Device Map stored in each NI10/E. The Device Maps are built automatically and can be manipulated by the network manager. Addresses can be disabled, enabled, appended, and deleted. These functions are provided for by software downloaded to the ENP from the floppy during bootstrap.

4.3 Theory of Operation

A diagram illustrating the encapsulation of Ethernet packets into the message frames of Applitek's UniLINK data link layer protocol is provided in Figure 6. A diagram illustrating the encapsulation and end-to-end transmission of user data between hosts using the GSFC LACN is provided in Figure 7. Additional information on the operation of NI10/E's is provided in Appendix B.

4.3.1 The UniLINK Access Method

A brief discussion of the UniLINK access method as implemented in the NI10/E's at Goddard follows (Dahod, 1983 a and b).

Network time is divided into numbered messages. All nodes recognize the message numbers and can only transmit in the message slots assigned to them. Message assignment on the NI10/E network can be done either manually, by the network manager, or assigned automatically and dynamically by the UniLINK process. Under manual allocation the network behaves in a deterministic manner similar to token ring. Under the automatic allocation, message slots are acquired by the NI10/E's as more bandwidth is needed, and the performance is similar to a contention network. At Goddard, the LACN is allocated in manual mode to provide a deterministic environment for implementation and testing. Only the lowest two layers of the ISO OSI model are implemented by UniLINK in the NI10/Es. Since error recover, flow control, security measures, etc. are not required or implemented by Ethernet, these function are relegated to the higher level protocols running in the hosts.

5.0 THE INTERLINK IBMmvs/DECnet GATEWAY

For several years, GSFC's scientific community has had an outstanding requirement for a high-speed (>100 kbps) network interconnecting DEC hosts with NSESCC's IBM mainframes, specifically for bulk file transfers. High speed access to the NSESCC is particularly important for several reasons: (1) the IBM host (3081-K) is capable of processing complex mathematical models and related scientific and engineering data; (2) a significant amount of mass storage both in terms of online (disks and mass store) and offline (tapes) is available to end users; (3) a high-speed laser printer (IBM 3800-3) is available for the production of high-quality documents containing text and graphics; and, (4) the NSESCC's IBM and compatible mainframes frontend a CDC CYBER 205.

Prior to the installation of the IBMmvs/DECnet Gateway at the NSESCC in March 1986, inter-host communications between GSFC DEC and IBM hosts was achieved on an ad hoc basis via the installation of point-to-point links with maximum data transfer rates of 56 kbps (Lehtonen, 1983). These rather inexpensive solutions worked rather well for the immediate hosts involved but offered no easy solution for the other DEC hosts that needed to gain access to the NSESCC IBM mainframes. In addition, the DEC user was burdened with having to know the details of IBM's Job Control Language (JCL) to enable initiation of file transfers to the IBM host. With the successful installation of this Gateway, the NSESCC IBM 3081 now appears on the network as a peer DECnet node allowing for transparent access from the remaining DECnet nodes (Interlink, 1985 a). To accomplish this network transparency, a combination of hardware (3711 Network Controller) and software residing in the IBM host (MVS/DECnet) and the 3711 has been developed by Interlink Computer Sciences. See Figure 8 for a depiction of this connectivity.

5.1 The 3711 Network Controller

The 3711 Network Controller serves as a front end processor by providing all the appropriate hardware interfaces to implement DECnet functionality as well as to attach to an IBM I/O block multiplexor channel. The 3711 is comprised of a DEC micro PDP 11/73 CPU using LSI-11 technology, a DEQNA interface to enable the connection to an Ethernet segment transmitting data at 10 mbps, and an Auscom 8900 Channel Interface Board which provides all the

functions necessary to enable the attachment of high-speed interfaces to the IBM channel at data rates up to 2 mbytes/s (Farmer, 1982). (The use of an Auscom interface eliminates the requirement for attaching to an IBM 3725 front end processor which limits data transfers to 230 kbps.)

The Auscom 8900 enables access to an IBM standard channel interface. To use the 8900 the IBM host must support an IBM channel (or look-alike) with one or more subchannel addresses reserved for this application. The channel can be configured as a block mux, selector, or byte-mux channel (Farmer, 1983).

5.2 The MVS/DECnet Software

The heart of the Gateway is the Interlink proprietary MVS/DECnet software residing in the IBM host and executing under the IBM MVS/SP Operating System. The MVS/DECnet software provides the necessary host architectural transparency to enable DEC remote users to transfer files back and forth to the IBM mainframe and, conversely, allows IBM users to access files located on remote DECnet nodes. Intimate knowledge of the file structures on each host is no longer a necessity for gaining access to various files. In fact, the software automatically translates data formats to the user's native format, as required. The user can also control the data translation by creating an appropriate data dictionary file which defines the format of each record in the file to be translated.

This Gateway software executes as a started task under the IBM MVS operating system (Note: a version that operates under the IBM VM operating system is also available from Interlink Computer Sciences). In addition to the functions previously mentioned, this software also controls the connection between DECnet and itself by establishing and maintaining a DECnet environment within the IBM mainframe--up to 32 concurrent sessions can be active: 16 outbound from the IBM host and 16 inbound to the IBM host. The following information summarizes the standard functions supported by the MVS/DECnet software:

5.2.1 DEC Node Initiated Communications

Functions in this category include: access to entire files (data sets) or to specific records within the file; IBM data set creation, deletion, or transfer using standard DEC utilities or commands; translation of data in uniformly formatted files (ASCII/EBCDIC, integer, floating point, single and double precision) or based upon a user-specified data dictionary that contains the format of each individual record; brief or full directory listings of data sets currently residing at the IBM host site; submission of jobs to the IBM host for execution and subsequent retrieval of relevant job output; and, submission of files to the JES print queue for printing at the IBM central printing facility (Interlink, 1985 a).

5.2.2 IBM Initiated Communications

Interlink has developed a Network File Transfer (NFT) Utility designed to provide the TSO IBM mainframe user with access to remote files located on DEC hosts. Access to these files is achieved through either an interactive ISPF menu session, a direct TSO command environment, or through the

submission of a batch job containing NFT subcommands. As with DEC node initiated communications, this NFT Utility enables an IBM user to send and receive files ("push and pull"), delete files, list the files within a DEC directory, and submit jobs to DEC batch and print queues.

The NFT Utility SEND and RECEIVE subcommands enable the IBM user to access files in sequential, relative, or indexed format. DEC files are manipulated using standard DECnet file specification within the ISPF menus or directly within NFT commands. Unlike the DEC user, however, IBM users cannot use the NFT Utility to gain access to individual records within DEC files.

To delete a DEC file, the IBM user uses the DELETE subcommand; to list the contents of a remote DEC directory, the DIRECTORY subcommand is issued. Information about the file size, organization, protection, etc. is displayed (Interlink, 1986 c).

5.2.3 Task-to-Task Interface

Interlink has also implemented a low level task-to-task interface capability. A user task running on a DEC host can communicate with a IBM application program executing on the IBM mainframe. A set of callable routines (from FORTRAN, for example) can be used to initiate and maintain a DECnet connection, to send and receive data, and to terminate a session (Interlink, 1985 e).

5.3 DEC Software

In order to perform most of the functions supported by the Interlink approach, no extra software is required to be installed on DEC hosts. (Obviously, the standard DECnet networking software, Phase III or Phase IV, will be required to enable the attachment of a DEC host to a DECnet network.) The only exception is if the remote DEC user wishes to submit a job to the IBM mainframe or to print a file on the IBM facility's printer subsystem--in this case, each DECnet node wishing such access must install the Interlink-provided Job Entry Subsystem (JES) Utility software, IBMJUT. See Appendix C for more details on this utility.

The software that actually provides the functionality for connecting the 3711 to an existing DECnet network resides in the DEC micro PDP 11/73 host. This software is incorporated within the RSX-11M-PLUS, Version 2.1 Operating System which provides all the functions for implementing DECnet Phase IV.

Additionally, Interlink has developed a Pass Through Task (PTT) that provides the connection between DECnet and the Auscom-related IBM Channel Interface Driver. The purpose of the PTT is to service requests to and from the DECnet network in conjunction with the MVS/DECnet software executing in the IBM mainframe.

5.4 Performance of Gateway

At GSFC's request, Interlink conducted several file transfer benchmarks between a VAX 11/780 and an IBM 4381-M2 using a micro PDP 11/73 processor in the 3711 Network Controller (Interlink Technical Proposal, 1985). The resulting performance was

dependent upon the type of data translation performed and the record size in bytes. For image data (i.e. no translation), transfer rates of over 400 kbps were achieved; for INTEGER*2 data, transfer rates of over 300 kbps were achieved. GSFC's IBM 3018-K should be able to achieve comparable or greater data transfer rates but as of this date comparable benchmarking of the Gateway has not yet been undertaken (see Section 7).

6.0 CURRENT GSFC LACN CONFIGURATION

6.1 LACN Communications Subnet

6.1.1 Configuration Overview

As of May 1986, in-use operation of GSFC's LACN broadband circuits extended to 10 GSFC buildings. In each of these buildings, at least one Applitek NI10/E bridged with an intra-building Ethernet segment serving as that building's primary baseband trunk. Fiber optic repeaters extended LACN Ethernets to 2 additional buildings. An overview of this configuration is provided in Figure 9.

6.1.2 Broadband Channel Assignments

Currently installed in building 8 are head end frequency translators capable of supporting 5 separate pairs of 6 mhz channels. Prior to May 1986, two pairs of channels, i.e. channels 3'/P operating at 60-66/252-258 mhz and channels 5'/S operating at 78-84/270-276 mhz, were in use. Also, prior to May 1986, functional use of the separately assigned channels had been determined by separate GSFC internal organizations considered to be the primary user of the assigned channels. Since there are considerable expansion restrictions to this type of channel assignment, GSFC has been considering making future channel assignments based on the planned use of different higher layer protocol sets which enable logically different computer networks. Such "by protocol" channel assignments, however, would require multiple NI10/E's in buildings with Ethernets that support the connection of computers which concurrently co-host multiple high layer protocol sets. As an alternative, GSFC is also investigating feasible schemes for inter-channel bridging.

6.1.3 Intra-Building Ethernets

Overview diagrams illustrating the extent of the intra-building Ethernets currently integrated into GSFC's LACN are provided in Figures 10A and 10B.

6.1.4 Operations

A Network Monitoring Station (NMS) has been configured in building 8 at the head-end of the CATV system. The NMS includes an IBM compatible PC which is interfaced to a Applitek NI10/T serial interface on the broadband subnet. The control functions of the NI10/T can communicate with the control levels of the NI10/E Bridges (Applitek, 1985b). From the centralized NMS, GSFC network managers are able to allocate bandwidth to each of the NI10's and monitor their usage and performance. Software for the PCs consist of common vendor supplied communication programs and spreadsheets, as well as GSFC-developed BASIC routines for extracting raw data from the responses of the NI10 units. Procedures for plotting

traffic loads and patterns and for packet level error performance have also been developed in-house at GSFC.

6.1.4.1 Setup and Maintenance

After a physical broadband channel has been installed, setup of the NI10/E subnet involves these steps.

- 1) Configuration disks are set to bring each NI10 on line in contention mode with all units contending for the same slots and in manual allocation mode. Frequency allocation is also assigned at this time.

- 2) Each NI10 is booted with its configuration disk.

- 3) The NMS operator, observing the NI10/E's as they come on line, can allocate each NI10/E its reserved and dedicated message slots.

- 4) The NMS operator checks packet performance and makes the necessary adjustments to the RF system to deliver the lowest error rate (typically 10e-08 at the packet level).

Maintenance involves monitoring performance and usage to provide the best performance at the broadband level. New NI10/E's are added in a contention spacing provided for this purpose and then allocated their bandwidth by the NMS operator.

6.1.4.2 Usage Monitoring

An example of an NMS display summarizing broadband channel usage, where 6 min averages are plotted against time, is provided in Figure 11.

6.2 DECnet-Enabled Computer Networking

An indication of the current operational extent of DECnet-enabled computer networking in GSFC's LACN is provided in the following subsections.

6.2.1 Configuration Overview

As of May 1986, approximately 25 GSFC local computers located in 8 different GSFC buildings were on-line to GSFC's LACN as DECnet nodes. In most cases, each of the DECnet-enabled computers has interfaced with the GSFC LACN via a single Ethernet connection using a DEC DEUNA (for UNIBUS systems) or DEC DEQNA (for Q-bus systems) Ethernet communications controller board. (Several DEC DELUAs have also been ordered but none have yet been delivered.) Use of a single interface per computer to achieve full interconnectivity with the rest of the DECnet-enabled GSFC LACN has minimized the communications processing overhead of the computers so interconnected. A few computers are physically interconnected via pairs of DEC DMR-11s which intercommunicate synchronously using DEC's DDCMP protocol. Also, one VAX 11/780 computer front-ending the MPP in building 28 was still needed to bridge two Ethernets (see Appendix A) until the delivery of more NI10/E's will enable GSFC to extend broadband LAN services to more GSFC buildings. An indication of these in-place interfaces and of the physical topology of the GSFC LACN just supporting

DECnet-enabled GSFC local computer networking is provided in Figure 12. An indication of the logical topology of GSFC local computers engaged in DECnet-enabled high speed computer networking is provided in Figure 13. This latter figure also provides a further perspective on GSFC's use of the Interlink Gateway to enable DECnet access to the mainframe capabilities of the NSESCC. A summary list of GSFC local nodes currently in-place or planned to be installed by the end of 1986 is provided in Appendix D.

Not shown in Figures 12 and 13 are approximately 5 DECserver 100's in various buildings providing terminal/print server support for approximately 40 DEC VT1XX/2XX's and 5 DEC LP11's. Also not shown in Figures 12 and 13, except for GSFC's SPAN link, are a number of GSFC remote DECnet links which are also in use at this time. Some are enabled by leased point-to-point lines operating at 4800 and 9600 bps and others by dial-up lines operating at 1200 bps. Both static and dynamic asynchronous DDCMP connections are used through a number of DEC DZ11 and DMF32 asynchronous communications ports.

6.2.2 Operational Management

6.2.2.1 Setup and Maintenance

Beyond the normal efforts associated with network software licensing, acquisition, and installation per host, setup and maintenance of the DECnet-enabled network within GSFC's LACN has primarily involved:

- o the establishment of network unique DECnet node numbers for all nodes;
- o netgening uniform DECnet data base configuration parameters;
- o updating DECnet internal routing/link costs to best match with GSFC's dynamically growing network, both local and remote; and
- o testing designs for a 9.6-56 kbps backup communications subnet in the event that parts or all of the interbuilding broadband LAN failed to function.

The DECnet-enabled computers in GSFC's LACN are a major participant in the NASA SPAN which is growing very rapidly. To facilitate growth, the NASA SPAN is now using several DECnet area numbers in its assignment of DECnet unique network addresses. Area 6 has been allocated to the GSFC LACN and a number of other regionally nearby computers, such as those at Harvard and Penn State, with which GSFC LACN nodes have remote DECnet links. Other major areas within SPAN, as of December 1985, included:

- 2 nodes in area 1 (Los Alamos National Labs area)
- 27 nodes in area 5 (JPL and USGS, Flagstaff, area)
- 50 nodes in area 7 (SPAN general use area)
- 7 nodes in area 8 (MSFC local area net area)
- 10 nodes in area 9 (UCSD, LOCKHD, UCLA area)

Many other areas and nodes have been included in the NASA SPAN since the beginning of 1986.

Within GSFC's area 6, DECnet node numbers have been assigned by a central coordinator as needed starting from 1. On the other hand, selection of DECnet node names is made by the individual user nodes. Periodically, an updated master list of GSFC's area 6 DECnet node numbers and names is distributed, with similar data from all other SPAN areas, throughout the entire SPAN. Also distributed with this list are recommended parameter settings for netgening a host's DECnet data base configuration. While the DECnet node names (not node numbers) and many of the netgen settings may be changed at the individual user nodes without negatively affecting the rest of the network, the common use of a standard set of names/settings has minimized problems frequently found with rapid network growth and has simplified overall network use.

In addition to the efforts applied to set up and maintain DECnet-enabled computer networking using the high speed communications subset of GSFC's LACN, preparations have also been initiated among GSFC local nodes to effect a backup, albeit slower speed, DECnet-enabled computer network using GSFC's new Rolm CBX-II once it is installed. These preparations have included test checkout of locally developed procedures to initiate dynamic asynchronous DDCMP connections from at least one router host on each building's intra-building Ethernet to prevent any building from being cut off from the GSFC LACN.

6.2.2.2 Usage Monitoring

Currently within GSFC's LACN, DECnet specific usage is monitored only at individual nodes if at all. GSFC is planning to acquire and install DEC's Network Management Control Center (NMCC)/DECnet Monitor product in the near future.

6.3 TCP/IP-Enabled Computer Networking

An indication of the current operational extent of TCP/IP-enabled computer networking in GSFC's LACN is provided in the following subsections.

6.3.1 Configuration Overview

As of May 1986, approximately 10 GSFC local computers located in 3 different GSFC buildings were on-line to GSFC's LACN as TCP/IP-enabled nodes. Two of the TCP/IP-enabled computers in GSFC's LACN are DEC VMS-based systems with Excelan EXQS 204 front end controller boards and EXOS 8043 TCP/IP Protocol Package; one VMS-based system is TCP/IP-enabled via software from the Wollongong Group. The other computers are Unix-based user workstations with Ethernet controller boards from 3Com, Interlan, and Xerox. An indication of these in-place interfaces and of the physical topology of the GSFC LACN just supporting TCP/IP-enabled GSFC local computer networking is provided in Figure 14. Also shown in Figure 14 are approximately 10 other GSFC local computers which GSFC expects to bring on-line to the GSFC LACN as TCP/IP-enabled nodes by the end of 1986 (again see Appendix D). Additionally shown in Figure 14 is GSFC's LACN in-place interconnection with the ARPAnet. This interconnection is enabled through the use of a pair of Vitalink Communications Corporation's TransLAN Ethernet bridges and a leased 56 kbps line between GSFC's LACN and an Ethernet at the University of Maryland

from which GSFC accesses an ARPAnet IMP. This link was first enabled in November 1985.

6.3.2 Operational Management

6.3.2.1 Setup and Maintenance

Beyond normal system maintenance, setup and maintenance of the TCP/IP-enabled network within GSFC's LACN have involved only the assignment of network unique TCP/IP node addresses for all nodes. At the present time, GSFC uses a set of addresses, established by the ARPAnet-related Internet Network Information Center, sublet to GSFC by the University of Maryland.

6.3.2.2 Usage Monitoring

Current TCP/IP-based networking activity levels within GSFC are very low but are growing steadily. Most activity is with remote ARPAnet nodes and usage monitoring is currently performed via the Vitalink TransLAN units.

6.4 XNS-Enabled Computer Networking

As of May 1986 approximately 100 XNS-enabled computers were currently Ethernet connected in GSFC's LACN and approximately 100 more were expected before the end of 1986. For the most part, they were all PCs. Since a description of the networking applications enabled by this networking protocol deserves more space than can be allotted here, further description of this networking activity will be deferred, to be separately documented in a related follow-on report.

7.0 NETWORK THROUGHPUT PERFORMANCE

This section includes an indication of the end-to-end throughput performance which has been achieved to date using GSFC's LACN.

GSFC's throughput performance studies of the LACN have been undertaken to determine if key design objectives have been met and to gain insight into the overall performance characteristics of the network. Since many of the subsystems integrated into GSFC's LACN have involved relatively new technology, to date GSFC's primary interest in assessing network performance has been to determine merely first order end-to-end throughput rates. Such end-to-end throughput rate measurements, as opposed to subsystem specific tests, are considered to be most important from an end user's perspective and are the ones specified in GSFC's LACN requirements (see Section 2).

The results of the studies presented in this section need to be considered preliminary indicators of GSFC's LACN throughput performance. System managers of both the communications subnet and the host computers in GSFC's LACN are still tuning a number of parameters which greatly affect LACN end-to-end throughput performance. Examples of these tunable parameters include the assignment of dedication or contention to the various message slots in the UniLINK protocol, the allocation of pipeline buffers in the various DECnet hosts, and the tradeoff of increasing the number of concurrent sessions versus few sessions with additional buffers to enhance access through NSESCC's IBMmvs/DECnet Gateway. Furthermore, major upgrades affecting several

subsystems comprising the LACN have already been made since these indicators were generated or are planned in the immediate future. Examples of these upgrades include shielding from RFI the active components in the broadband LAN, the upgrade of the NI10/E modems to ones with better signal-to-noise enveloping, and the installation of revised NI10/E software specifically redesigned to increase data flow performance. A more detailed analysis of GSFC's LACN performance with these upgrades installed is in progress and will be separately documented in a follow-on report.

The throughput indicators presented in this section have been derived from various host memory-to-memory and disk-to-disk data transfer tests conducted in evening hours between lightly loaded single pairs, and concurrently among multiple pairs, of end user computer nodes in GSFC's LACN. No standalone use of hosts or the communications subnet has been attempted in the end-to-end throughput tests conducted to date. Routine daily throughput within GSFC's LACN is similar to that indicated here, but is sometimes more subject to delays originating within hosts due to heavy internal user workloads.

7.1 Measurement Approach

Because GSFC's LACN includes new technology, off-the-shelf tools and methodologies for measuring performance have not been readily available. GSFC, therefore, has undertaken significant ongoing development to enable its desired end-to-end performance measurements. This has included the implementation of host-to-host oriented, software controlled tests using various networking protocol sets. In implementation to date, DECnet-based tests have comprised the overwhelming majority, with only a few TCP/IP-based tests having yet been conducted. Separate tests involving only user task controlled host memory-to-host memory transfers, as opposed to disk file-to-disk file transfers, have been performed to disclose the effects of operating system dependent I/O.

Generation of the majority of the test results compiled to date has been accomplished through a GSFC developed TEST package. The TEST package interactively collects user specified test scenarios using menus and prompts, builds appropriate data transfer commands and/or task software, downloads the commands/software to from 1 to approximately 10 pairs of host computers (possibly all independent or overlapping) as defined by the user's test scenario, and enables either memory-to-memory or disk-to-disk data transfers between the pairs using prescribed record sizes and record counts or file sizes, again, as defined by the user's test scenario. Originally written in DEC's Digital Command Language (DCL), the TEST package has recently been rewritten in C to facilitate its being ported to run on Unix based systems.

7.2 Control Tests

Various control tests have been conducted to date to establish a comparison baseline of throughput performance in host-to-host data transfers when both hosts are connected to the same physical Ethernet. For example, separate tests have parameterized record sizes and number of records sent. Sample results of such tests, which effectively establish

upper limits on memory-to-memory transfers between VAX's connected to Ethernets using DEC DEUNAs, are illustrated in Figures 15 and 16, respectively.

7.3 Inter-Building End-to-End Throughput Measurements

A complete set of end-to-end throughput measurements involving variously selected hosts (number of pairs, building location, host type, operating system type, operating system release level, etc.) has yet to be accomplished. The following results, however, do provide a preliminary though coarse indication of the GSFC LACN's current end-to-end throughput performance capabilities.

7.3.1 Inter-Building Disk-to-Disk Through the Interlink Gateway

A series of 5 independent measurements, essentially DECnet-based, were conducted between a VAX 8600 and NSESCC's IBM 3081K. The computers were in different buildings on Ethernet segments interconnected by the broadband LAN. Using 512 byte records and 512 records/file in each test, and performing entire file ASCII/EBCDIC translations, disk-to-disk throughput transfer rates were 104 kbps on average, with a standard deviation of 57 kbps and a maximum of 205 kbps. Results from similar tests using different record and file sizes are illustrated in Figure 17.

7.3.2 Inter-Building Memory-to-Memory

A series of 91 independent measurements completely DECnet-based were conducted between variously selected pairs of VAX-11/780's, with only one pair of VAX's transferring at a time. Each VAX of the pair were in different buildings on Ethernet segments interconnected by the broadband LAN. Using 1024 byte records and 1024 records transferred in each test, memory-to-memory throughput transfer rates were 346 kbps on average, with a standard deviation of 130 kbps and a maximum of 748 kbps.

8.0 NETWORK DEVELOPMENT PLANS

As the GSFC LACN matures, the need to provide enhancements is being anticipated. Consequently, a number of development activities and new applications are being considered and are discussed in the following sections.

8.1 RF Channel Allocation

A strategy for allocation of the several RF channels on the broadband medium is needed and a number of approaches are being considered. These include allocation by organization, by protocol or by traffic loading. A final determination from among these options will be made only after more information about actual growth patterns has been acquired.

8.2 Enhancements to the Broadband LACN

8.2.1 Direct Broadband LACN Interfaces

A development path which would allow for higher end-to-end throughput performance within the presently evolving LACN is, in appropriate instances, to eliminate the Ethernet feeder segments and provide dedicated interface units directly between computer I/O busses and a broadband channel. This option together with the use of a data transfer protocol specifically "tuned" for high-throughput, could serve as a basis for a higher-speed data transfer service. A separate RF channel on the broadband LACN could serve as a testbed vehicle for prototyping ideas such as this without impacting the operational networks. Since this prototype would not participate in the same communication sub-nets, there would exist no need for complete compatibility with the medium access method or data link protocol used on those channels.

8.2.2 Higher Bandwidth RF Channels

Another option being considered is to provide very high (50-100 mbps) aggregate data rate channels on the broadband medium rather than the standard 6 mhz (10 mbps) channels. If projected traffic load levels warrant, implementing such channels would serve to reduce the operational management problems associated with maintaining several of the standard channels and the consequent necessity to provide inter-channel bridging. This option is contingent on the development of appropriate high-speed RF modems, which is not certain at this time. Should these modems materialize, the use of such channels to support higher end-to-end throughput would also be considered. This would have to be coupled with the availability of similarly high performance computer bus interfaces and network protocol software to be justified as a feasible alternative.

8.3 High Performance Computer Networking based on Optical Fibers

Optic fiber cables are presently being routed to most Center buildings. While the present broadband LACN can be augmented to permit very high aggregate data capacities, it will probably "top out" around 5 mbps in end-to-end throughput for a 6 mhz channel. While this is presently adequate, computer I/O busses operating in the multiple tens of mbytes per second range can be expected soon. Even though additional development in RF modems can also be expected, the exploitation of fiber optic technology has considerably more potential for supporting high throughput networking than the broadband medium. Again, the issue of high level protocol software is an important factor.

8.4 IBM 3270 Graphics Support For Remote Users

GSFC contains a significant number of DEC minicomputer sites as well as a lesser number of IBM mainframes (3081, 4341, etc.). Although a majority of the IBM users communicate with their respective IBM hosts via a dial-up or point-to-point, twisted-pair network using inexpensive ASCII start-stop terminals, a growing number of users particularly those with graphics requirements are procuring IBM or IBM-compatible 3270 terminals and connecting these via either the traditional twisted-pair

technology or IBM-specified coaxial-cable. A number of these 3270 terminals are functionally capable of displaying graphical images composed at the IBM mainframe site. The IBM 3270 terminal attached to an IBM 3274 cluster controller represents a de facto standard for IBM terminal communications. As mentioned above, this growing proliferation of cluster controllers and associated graphics and text-only terminals requires the implementation and maintenance of a separate coaxial-based (or twisted-pair), point-to-point network with its associated problems of pulling coaxial cable to multiple offices, implementing long-distance cable links through GSFC's nearly full underground ducts, and tracking the various communications hardware and software that comprise this network.

As an alternative to the above scenario, GSFC is currently examining various vendor offerings (Applitek, Sytek, Ungermann-Bass) in the area of 3270 terminal connectivity using broadband LAN technology. The theory behind using broadband technology is that existing IBM clusters now located remotely could be positioned back at the host IBM mainframe site (for ease of hardware and software maintenance) and replaced with broadband network adapters to provide the existing terminal connectivity (see Figure 18). Such a scheme would offer the following advantages: (1) achieving local response times at the remote terminal (2.34 mbps)--this is especially critical for transmitting graphics which often are composed of 50,000 bytes of information; (2) isolating IBM data traffic from the other network traffic by using a separate broadband channel; (3) eliminating multiple cable pulls each time a new cluster site is identified; and, (4) easing maintenance of the 3270-based system as all hardware would be now located at the central IBM facilities.

Acknowledgments

The authors wish to express their thanks: to NASA managers Dr. Caldwell McCoy, Dr. Erwin Schmerling and Joseph Bredekamp for their continued support for this effort; to David Howell for his active participation with the authors to system engineer large parts of GSFC's LACN, particularly the XNS-enabled network; to John Arslanian for his support in the RF system design and the use of GSFC's CCTV cable plant; to the Allied Bendix Field Engineering technical crew, particularly Dave Yoest and Jeff Powell, for their many efforts beyond the call of duty in installing, certifying, troubleshooting and operating the network, particularly the broadband RF system; to Charles Cosner for his expert system analysis and programming assistance, particularly in the development of the network throughput performance TEST package; and to the Applitek Corporation for working with GSFC in the development of the Ethernet Bridge, without which the implementation of the hybrid LACN design would not have been feasible.

APPENDIX A - GSFC LACN USE OF ETHERNET AND CATV COMPONENTS

Ethernet

Most of GSFC'S Ethernet segments consist of standard 50-ohm coaxial cable, such as Beldon's 9880 or DEC's BHE2 yellow polyvinyl chloride covered cables. In a number of instances, primarily involving terminal and PC connections, 50-ohm thin cable (RG58) has been used for intra-organizational segments which are then connected via local repeaters to a common intra-building thick Ethernet trunk. In two instances (actually the first inter-building segments of the LACN installed in February 1985), it was determined that inter-building LACN connectivity could best be achieved at that time via the installation of two separate extended Ethernets using fiber optic cables and remote repeaters. Subsequently, these two separate Ethernets were linked into a single logical network at the network protocol level (OSI layer 3) by using a host computer as a store-and-forward router. In these cases, 650 and 960 meters of 6 strand fiber optic cable were installed between the building pairs 22 and 28 and the building pairs 16 and 28, respectively. The two extra optical fiber pairs in each optical fiber cable run were provided for contingencies, such as damaged fibers, as well as possible future extensions to the network. The optical fibers used are 100/140 micron graded-index type fibers. These have a bandwidth distance product of 100 Mhz.Km, and so can theoretically support data transmission up to 100 mbps in future point-to-point connections between computers located in either of the building pairs. Fiber optic connections with standard Ethernet coax cables were made via Amphenol 906 connectors to pairs of DEC's DEREK remote half-repeaters. A schematic identifying these connections is provided in Figure A1. (The location of several computers identified in Figure A1 has changed since this initial GSFC LACN network configuration.) Ethernet transceivers concurrently in use on GSFC's LACN include: DEC's H4000; 3Com's 3C100 revision 1 (without heartbeat) and 3C102 revision 2; Interlan's NT100; and TCL's 2010EB revision 1, 2010IS revision 2, and 2100 modular multiport.

CCTV/CATV

Because of corrosive water damage at a large number of cable splice locations, extensive portions of GSFC's cable plant have had to be replaced to certify the plant for digital broadband LAN use. GSFC has two coaxial cables extending to each building from building 8. Nevertheless, for implementation of its initial LACN, GSFC has elected to use a single cable mid-split CATV/CCTV industry standard digital broadband LAN. Under this approach, a single coaxial trunk is allocated 17 full duplex, 6 mhz channels in the 5 mhz to 300 mhz frequency band as illustrated in Figure A2. With two cables available and the use of 10 mbps modems operating in each of the 6 mhz channels, GSFC's broadband LAN has the potential for supporting a total aggregate data transfer capacity of 340 mbps throughout GSFC. An overview of the RF designs supporting GSFC's LACN is provided in Figure A3. The primary LAN components used in the broadband aspects of GSFC's LACN include:

- o multi-channel 192.5 mhz frequency translators from Bridge Communications, Inc.;

o splitter/combiners to accommodate 24 separate building circuit paths; and

o Augat Broadband VFA 450 and VRA 200 20 db amplifiers and Jerrold TF-108D-HE filters per building circuit path to isolate each building's circuit path from failures occurring elsewhere in the LACN.

APPENDIX B - NI10/E OPERATION

Inbound Data Flow from the Ethernet

A simplified view of data flow from the Ethernet to the UniLINK network follows (Communication Machinery Corp., 1984).

1) The ENP, driven by the Kernel software, listens to the Ethernet for packets. Packets are received into on-board dual-ported RAM. The end of packet sequence causes the LANCE to signal the Ethernet receive process (ERxP). At power up, the ERxP subroutine is downloaded to on-board RAM and the vector for the routine is loaded into the LANCE ERxP vector location. In operation the ERxP performs the following checks:

a) If the destination address appears in the Device Map then discard the packet, since the destination is on the local Ethernet.

b) If the destination does not appear in the Device Map then the destination is on a remote Ethernet. The source address is then checked in the Device Map for authority to access the bridge. If the address is not in the Device Map, it is added as a new source on the Bridge's Ethernet. New sources have default access to the Bridge.

c) The Relay Transport Layer Process is signaled, which encapsulates the entire Ethernet packet in a UniLINK format, and signals the Subscriber Processor's Tx RAM server.

d) All multi-cast packets are bridged through.

2) The Subscriber Processor, via DMA, moves the UniLINK formatted packet to the Tx RAM (4 kbytes FIFO) from the dual ported RAM on the ENP.

3) From Tx RAM the Network Controller hardware and firmware take over to send the data over the broadband network.

Outbound Data Flow to the Ethernet

A simplified view of the outbound data flow from the broadband UniLINK network to Ethernet follows.

1) Every UniLINK packet is received and loaded into Rx RAM (8 kbytes FIFO) and the Subscriber Processor Rx RAM server is signaled.

2) The Subscriber Processor, via DMA, moves the UniLINK packet from Rx RAM on the Network Controller board to the dual port RAM on the ENP and signals the ENP UniLINK Receive Process.

3) The destination address is checked:

a) If the destination is not in the Device Map, then the packet is discarded.

b) If the destination is in the Device Map, then the destination's authority for access to the bridge is verified.

c) The Relay Transport Layer Process (RTLTP) is signaled. The RTLTP strips the UniLINK format off the Ethernet packet and signals the LANCE to transmit the packet in dual port RAM.

d) The Kernel firmware and LANCE transmit the packet onto Ethernet.

e) All multi-cast packets are bridged through.

APPENDIX C. JES UTILITY SOFTWARE (IBMJUT)

The JES Utility enables DECnet users to submit JCL streams for execution on an IBM mainframe and to monitor the status of these jobs. The VAX user issues a SUBMIT/REMOTE command, while the PDP user issues the NFT command. Just prior to issuing these commands, the user will have had to initiate a JES "session" via the invocation of the JES command (see below) at his or her terminal. This JES command establishes a connection with software executing on the IBM side that enables the subsequent issuance of VMS DCL commands or RSX NFT commands that initiate IBM JES functions.

For the VAX user, standard DCL commands are used which submit command procedure files to either submit a job for execution on the IBM mainframe or to print a file on the IBM printer subsystem. It should be noted that in either case the file of interest must reside on the IBM side of the network prior to issuance of the appropriate DCL command (a SEND command could have been previously issued to transfer the file to the IBM facility).

For the PDP user, the standard NFT Utility is used to submit jobs or print files on the remote IBM node. The PDP user has two choices regarding the submission of a job to the IBM remote: (1) the file can already exist at the IBM node in which case the execute option (/EX) of the NFT command is used or (2) a file can be transferred then automatically submitted from the remote DEC node using the submit option (/SB) of the NFT command. In the case where a file is to be printed on the remote IBM printer, the spool option (/SP) of the NFT command is used. Here, however, the file to be printed must not reside on the IBM mainframe in order for the spool option to function properly.

In order for any of the remote DEC users to submit work for the IBM JES, the JES User Interface Task must first be invoked. This is accomplished by typing in the 'JES' command at the user terminal with appropriate qualifiers that tailor the JES Utility session. Examples of these qualifiers include: IBM node name, username, password, account number, etc. Once this JES Utility session has been established, the DEC user can issue various JES commands: FETCH a job's output datasets to the local DECnet node; PURGE unwanted job output; SHOW the status of jobs submitted; and, EXIT the JES Utility session.

APPENDIX D - GSFC LOCAL COMPUTERS IN THE LACN

This appendix provides separate lists of major GSFC local computers which 1) were already functioning in GSFC's LACN as of April 15, 1986 and 2) were planning to interface with the GSFC LACN by the end of 1986. The lists are only partially complete in that no XNS-enabled computers (mostly PCs) are listed. Also, some major computers had not yet finalized their 1986 network interface plans.

In the lists, multiple entries in the "protocol" column do not necessarily imply gatewaying, but rather concurrent and logically separate participation in the logically separate networks enabled by the protocols.

PARTIAL LIST OF GSFC LOCAL DATA SYSTEMS ON-LINE TO SESNET/LACN (PAGE 1 OF 3)
AS OF 4/15/86

<u>DIVISION LEVEL ORGANIZATION</u>	<u>COMPUTER SYSTEM</u>	<u>NODE NAME</u>	<u>BUILDING /Room</u>	<u>PROTOCOL*</u>
MISSION OPERATIONS DIVISION/510	VALID WORKSTATION	VLS18	14/TBD	T
DATA SYSTEMS TECHNOLOGY LABORATORY/520	DEC VAX 8600	DSTL86	23/N405	D , T
	DEC VAX-11/785	DSTL85	23/N405	D
	DEC VAX-11/750	KIRK	23/N405	D
	VALID WORKSTATION	VLS11	23/N405	T
	VALID WORKSTATION	VLS12	23/N405	T
	VALID WORKSTATION	VLS16	23/E407	T
	VALID WORKSTATION	VLS17	23/W426	T
	IRIS WORKSTATION	CAD522	23/N405	T
INFORMATION PROCESSING DIVISION/560	VALID WORKSTATION	VLS14	23/W320	T
	VALID WORKSTATION	VLS15	23/TBD	T
LABORATORY FOR ATMOSPHERES/610	DEC VAX-11/780	PACF	21/C222	D
	DEC VAX-11/780	VAP	22/111	D
	DEC MICROVAX II	MV1	22/384	D
	DEC MICROVAX I	MV2	22/384	D
	DEC MICROVAX I	MV3	22/384	D
	DEC MICROVAX I	MV4	22/384	D
	DEC VAX-11/780	LTP	16W/N70	D
NASA SPACE AND EARTH SCIENCES COMPUTING CENTER (NSESCC)/630.1	IBM 3081K (CPU0)	SCFVM	1/6	B
	(CPU1)	SCFMVS	1/6	B , D
	AMDAHL 470 V/6-11	VPFVM	22/687	B
	AMDAHL 470 V/7B	VPFMVS	22/687	B , R
	CDC CYBER 205	CY2	22/687	R
NATIONAL SPACE SCIENCE DATA CENTER (NSSDC)/630.2	DEC VAX-11/780	NSSDC	26/121	D
	DEC PDP-11/23	OPTDSK	26/121	D
GODDARD IMAGE AND INFORMATION ANALYSIS CENTER (GIIAC)/630.3	DEC VAX-11/780	MPP	28/W189	D , T
	DEC VAX-11/780	IAF	28/W294	D
	DEC MICROVAX I	MV5	28/W294	D
	SUN MICROSYSTEMS WORKSTATION	SUN2	28/W120R	T
LABORATORY FOR HIGH ENERGY ASTROPHYSICS/660	DEC PDP-11/44	BBXRT	2/264	D
	VALID WORKSTATION	VLS13	2/TBD	T
LABORATORY FOR ASTRONOMY & SOLAR PHYSICS/680	DEC PDP-11/44	STARS	21/GC21	D
	DEC VAX-11/750	CHAMP	21/GC21	D
	DEC VAX-11/750	UIT	21/GC21	D
LABORATORY FOR EXTRATERRESTRIAL PHYSICS/690	DEC VAX-11/730	SIRIS	2/234	D
	DEC VAX-11/785	LEPVAX	2/S114	D
APPLIED ENGINEERING DIVISION/730	DEC VAX-11/780	CSDR	7/E178	D
SPECIAL PAYLOADS DIVISION/740	DEC VAX-11/780	VX740	5/C235	D
ENGINEERING SERVICES DIVISION/750	DEC MICROVAX II	ELDYN	21/L108	D

ADDITIONAL GSFC LOCAL DATA SYSTEMS WITH SESNET/LACN INTERFACE PLANS BY END OF FY86 (PAGE 1 OF 3)
AS OF 4/15/86

<u>DIVISION LEVEL ORGANIZATION</u>	<u>COMPUTER SYSTEM</u>	<u>NODE NAME</u>	<u>BUILDING /ROOM</u>	<u>PROTOCOL*</u>
MISSION OPERATIONS DIVISION/510	DEC VAX 8600	GROSF	14/S287A	D
	DEC VAX-11/785	ESTIFA	14/W22	D
	DEC PDP-11/84	ESTIFV	14/W22	D
DATA SYSTEMS TECHNOLOGY LABORATORY/520	DEC PDP-11/44	DSTL44	23/N405	D T
	DEC VAX STATION II	TBD	23/N405	D
	DEC VAX STATION II	TBD	23/N405	D
NASA COMMUNICATIONS DIVISION/540	DEC PRO350	PRO350	12/E126	D
FLIGHT DYNAMICS DIVISION/550	DEC VAX 8600	TBD	23/TBD	D
	DEC VAX-11/780	TBD	23/TBD	D
INFORMATION PROCESSING DIVISION/560	DEC VAX-11/785	IPDGW	23/N304	D H
ORBITING SATELLITES PROJECT/602	DEC VAX-11/750	TBD	7/TBD	D
	DEC VAX-11/750	TBD	7/TBD	D
LABORATORY FOR ATMOSPHERES/610	DEC VAX-11/730	TBD	7/TBD	D
	DEC MicroVAX II	DE10	11/E138	D
	DEC MicroVAX II	DE614	11/E138	D
	DEC MicroVAX II	DE696	TBD	D
	DEC PDP-11/70	PACF70	21/C222	D
	MASSCOMP MC-500DP	TBD	TBD	T
NATIONAL SPACE SCIENCE DATA CENTER (NSSDC)/630.2	DEC VAX 8600	TBD	26/121	D T
	DEC MicroVAX II	TBD	2/TBD	D
LABORATORY FOR HIGH ENERGY ASTROPHYSICS/660	DEC PDP-11/44	EGRET	2/264	D
LABORATORY FOR OCEANS/670	DEC VAX-11/750	OCEAN1	22/291	D
	DEC MicroVAX II	OCEAN2	22/291	D
	DEC MicroVAX II	OCEAN3	22/289	D
LABORATORY FOR ASTRONOMY & SOLAR PHYSICS/680	DEC VAX-11/750	HRS	21/GC21	D
LABORATORY FOR EXTRATERRESTRIAL PHYSICS/690	DEC MicroVAX II	SIRIS2	21/TBD	D
SPACE TECHNOLOGY DIVISION/710	DEC VAX-11/780	TBD	11/TBD	D
	DEC VAX-11/750	TBD	5/TBD	D
	DEC MicroVAX II	TBD	5/TBD	D
APPLIED ENGINEERING DIVISION/730	DEC VAX-11/750	FIRAS	7/E172	D
	DEC VAX-11/750	DIRBE	7/E172	D
	DEC VAX-11/750	DMR	19/E13	D
	DEC VAX-11/780	VX730	11/E138	D
	DEC VAX-11/750	VAX731	5/113A	D
	DEC VAX-11/785	VX750	5/W214A	D
	DEC VAX-11/750	VX750B	5/W214A	D

* B - BITNET (IBM), D - DECNET (DEC), H - HYPERCHANNEL (NSC),
R - RHF (CDC LCN), T - TCP/IP (DARPA), X - XNS (XEROX)

UNOFFICIAL PARTIAL LIST OF MO&DSD LOCAL DATA SYSTEMS
TO INTERFACE WITH UNIFIED HYPERCHANNEL-BASED LAN

<u>DIVISION LEVEL ORGANIZATION</u>	<u>COMPUTER SYSTEM</u>	<u>NODE NAME</u>	<u>BUILDING /ROOM</u>	<u>PROTOCOL*</u>
MISSION OPERATIONS DIVISION/510	DEC VAX-11/785	DOC1	14/TBD	H
	DEC VAX-11/785	DOC2	14/TBD	H
	IBM 4341	CMS1	14/TBD	H
	IBM 4341	CMS2	14/TBD	H
	DEC PDP-11/70	AP01/4	14/TBD	H
	DEC PDP-11/44	SPF1	14/TBD	H
	DEC PDP-11/44	SPF2	14/TBD	H
DATA SYSTEMS TECHNOLOGY LABORATORY/520	DEC PDP-11/44	DSTL44	23/N405	H, D, T
	DEC VAX 8600	DSTL86	23/N405	H, D, T
	IBM 4341	SSP	3/S191	H
FLIGHT DYNAMICS DIVISION/550	NAS 8040	FDF1	3/S191	H
	NAS 8040	FDF2	3/S191	H
INFORMATION PROCESSING DIVISION/560	DEC VAX-11/785	IPDGW	23/W304	H, D
	IBM 4341	MACS	23/C426	H
	SPERRY UNIVAC 1100/82	TPF	23/N218	H
	TBD	CDHF	23/E418	H
	IBM 370/145	TELOPS	23/N120	H

* B - BITNET (IBM), D - DECNET (DEC), H - HYPERCHANNEL (NSC),
R - RHF (CDC LCN), T - TCP/IP (DARPA), X - XNS (XEROX)

APPENDIX E - MANUFACTURERS OF MAJOR SUBSYSTEMS IN
GSFC'S LACN

Applitek Corporation
107 Audubon Rd
Wakefield, MA 01880

Auscom, Inc.
2007 Kramer Lane, Suite 102
Austin, Texas 78758

Bridge Communications, Inc.
10401 Bubb Rd.
Cupertino, California 95104

Digital Equipment Corporation
Maynard, MA 01754

Excelan, Inc.
2180 Forture Dr.
San Jose, California 95131

Interlink Computer Sciences, Inc.
39055 Hastings Street Suite 203
Fremont, California 94538

Rolm Corporation
4900 Old Ironsides Dr.
Mail Stop 626
Santa Clara, CA 95050

Vitalink Communications Corporation
1350 Charleston Road
Mountain View, CA 94043

Wollongong Group, Inc.
1129 San Antonio Road
Palo Alto, CA 94303

3Com
1390 Shorebird Way
Mountain View, California 94043

GLOSSARY

ASCII	American standards character for information interchange
BER	Bit error rate
bps	bits per second
CATV	Community antenna television
CCTV	Closed circuit television
CDC	Control Data Corporation
CMC	Communication Machinery Corporation
CPU	Central processing unit
CSMA/CD	Carrier sense multiple access with collision detection
DCL	Digital command language

DDCMP	Digital data communications management protocol	MVS/SP	Multiple virtual storage
DEC	Digital Equipment Corporation	NASA	National Aeronautics and Space Administration
DMA	Direct memory access	NFT	network file transfer utility of DEC RSX
DMR11	Communications interface for the Unibus	NJE	Network job entry of IBM
DNA	Digital Network Architecture	NMS	Network monitoring station
DR11	parallel interface for the Unibus	NSESCC	NASA Space and Earth Sciences Computing Center
DZ11	serial interface for the Unibus	OSI	Open Systems Interconnection
EBCDIC	Extended binary coded decimal interchange code	PABX	Private automatic branch exchange
ENP	Ethernet Node Processor	PC	Personal computer
FIFO	First in first out	PTT	Pass through task
FPS	Floating Point Systems	QPSK	Quadrature phase shift keying
FTP	File transfer protocol	QSAM	Queued sequential access method
gbytes	gigabytes	RAM	Random access memory
GSFC	Goddard Space Flight Center	RF	Radio frequency
hz	hertz	RFI	Radio frequency interference
I/O	Input/output	ROM	Read only memory
IBM	International Business Machines	RSCS	Remote spooling communications subsystem
IEEE	Institute of Electrical and Electronics Engineers	RSX	DEC operating system for the PDP family of computers (specifically RSX-11M-PLUS for the PDP 11/73)
IMP	Interface message processor	RTLTP	Relay transport layer process
IP	Internet protocol	SMTP	Simple mail transfer protocol
ISAM	Index sequential access method	SNA	System network architecture of IBM
ISO	International Standards Organization	SPAN	Space Physics Analysis Network
ISPF	Interactive system productivity facility	T1	1.544 mhz bandwidth transmission channel
JCL	Job control language of IBM	TCP	Transmission control protocol
JES	Job entry subsystem of IBM	TELNET	Telnet virtual terminal protocol and options
kbps	kilobits per second	TSO	Time sharing option of IBM MVS/SP operating system
kbytes	kilobytes	UniLINK	Applitek's proprietary protocol that runs on the Applitek NI10/E Ethernet Bridge
LACN	Local area computer network including the local area communications subnet	VLSI	Very large scale integration
LAN	Local area network	VM	Virtual memory operating system of IBM
LANCE	Local area network controller for Ethernet	VMS	DEC operating system for the VAX family of computers
mbps	megabits per second	VSAM	Virtual sequential access method of IBM
mbytes	megabytes	XNA	Xerox network architecture
mhz	megahertz	XNS	A particular implementation of XNA
MPU	Micro processing unit		

References

- Applitek Corp., Applitek Product Description: Ethernet Bridge, October 1985.
- Applitek Corp., Applitek Product Description: Network Management System, October 1985.
- Applitek Corp., Applitek Product Description: Broadband RF Modem, October 1985.
- Applitek Corp., Applitek Product Description: Baseband Modem, October 1985.
- Applitek Corp., Applitek Product Description: Optical Fiber Tap, October 1985.
- Applitek Corp., Applitek Product Description: NI10/DR11, October 1985.
- Applitek Corp., Applitek UniLAN Concepts: NI10/T Software Version 1.1, March 1985.
- Berman, R. L., Local Area Network Vendor Survey and Recommendation, Computer Sciences Corporation, CSC/TM 82/6134, August 1982.
- Communication Machinery Corp., Ethernet Node Processor (ENP) Kernel Software Manual, October 1984.
- Dahod, A. M., 10M-bps LAN Combines Benefits of CSMA/CD and Token Passing, Mini-Micro Systems, November 1983.
- Dahod, A. M., Multiple Access Method Embraces Popular Local Net Schemes, Data Communications, November 1983.
- The Ethernet, A Local Area Network, Data Link Layer and Physical Layer Specification, DEC, Intel, Xerox, Version 2 AA-K759B-TK, November 1982.
- Farmer, M., Auscom's Protocol Specification for IBM to Ethernet, Auscom, Inc., December 1982.
- Farmer, M., Private Communication, April 21, 1983.
- Gary, J. P. and E. D. Rothe, ADP Feasibility Study for Space and Earth Sciences Network (SESnet) Stage I (1985/1986), April 24, 1985.
- Green, J. L., and D. J. Peters (editors), Data Systems Users Working Group (DSUWG), Introduction to the Space Physics Analysis Network (SPAN) - First Edition, NASA Technical Memorandum NASA TM-86499, April 1985.
- Halem, M. et al., Space Data and Computing Division Program Plan: Fiscal Years 1985-1990, April 25, 1985.
- IEEE-802 LAN Family of Standards, IEEE Press, December 1984.
- Interlink, Remote Node Users Guide, Publication No. UG01RN.3.2U, November 1, 1985.
- Interlink, IBMmvs/DECnet Gateway Installation Guide, Publication No. IG01BAS.2C, September 1985.
- Interlink, Network File Transfer User's Guide, Publication No. UG02NFT2.3U, January 15, 1986.
- Interlink, Translation Record Definition Utility User's Guide, Publication No. UG01TRDU.2.2U, November 11, 1985.
- Interlink, Task to Task Handler User's Guide, Publication No. UG01TTH.1.0U, March 15, 1986.
- Interlink, System 3711 Installation Guide, Version #2.1, Publication #8404, revised November 1984.
- Interlink Technical Proposal, NASA/Goddard Space Flight Center Solicitation No. RFP5-36923/039, September 1985.
- Lehtonen, K. E., Alternative Methods for Networking DEC Host Computers with an IBM 3081 Host Computer, White Paper, January 1983.
- Lehtonen, K. E. and J. P. Gary, Guidelines for Usage of NSESCC IBMmvs/DECnet Gateway, SESC-UG19-00, March 1986.
- Mish, W. H., Networking DEC and IBM Computers, Proceedings of the Digital Equipment Computer Users Society, May 1983.
- Mish, W. H., K. E. Lehtonen, and J. P. Gary, Specifications for a System to Enable High Speed DECnet Functionality on an IBM Mainframe, NASA/GSFC RFP5-36923/039, issued August 29, 1985.
- Rebibo, K. K. and H. G. Miller, Sciences Directorate Local Area Network Study, MITRE Corporation Metrek Division, WP 82W00428, August 1982.

GODDARD SPACE FLIGHT CENTER LOCATION MAP

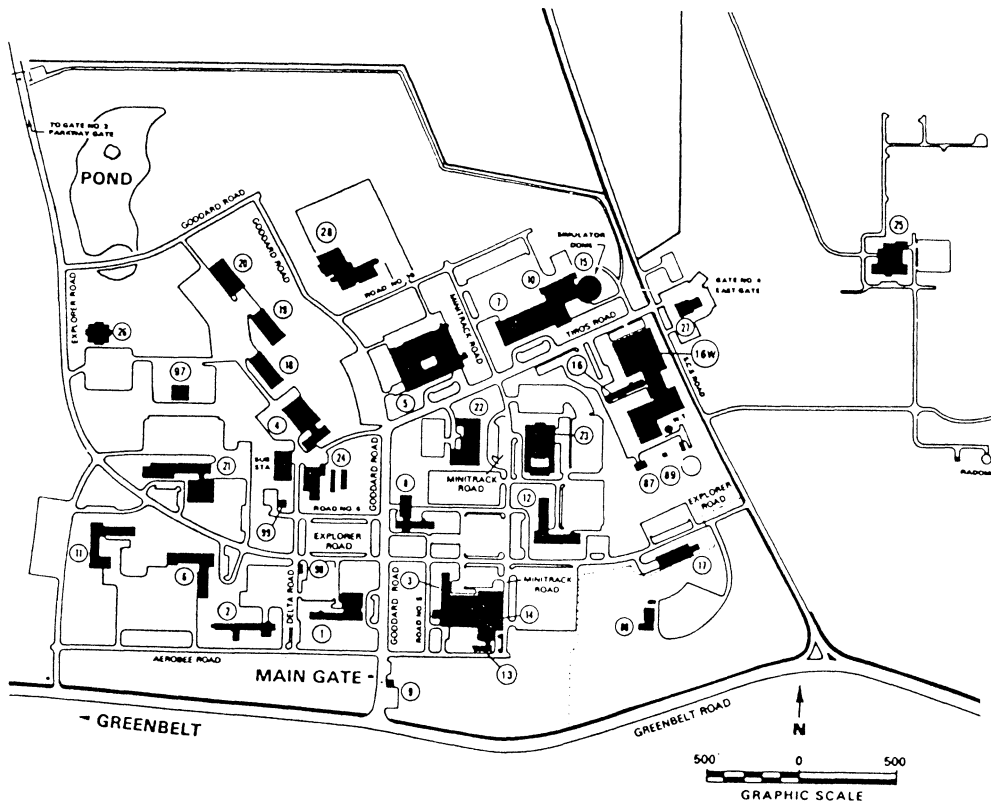
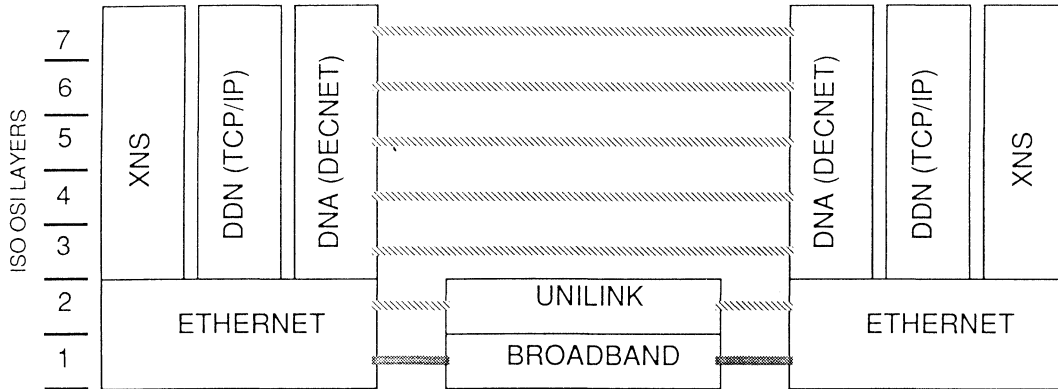


Figure 1. GSFC Building Location Map

<u>Division Level Organizations</u>	<u>Computer System</u>	<u>Node Name</u>	<u>Building/Room Location</u>
Data Systems Technology Laboratory	DEC VAX-11/780	DSTL86	23/N405
	DEC VAX-11/785	DSTL85	23/N405
	DEC VAX-11/750	KIRK	23/N405
Laboratory for Atmospheres/610	DEC VAX-11/780	PACF	21/C222
	DEC VAX-11/780	VAP	22/111
	DEC MicroVAX II	MV1	22/384
	DEC MicroVAX I	MV2	22/384
	DEC MicroVAX I	MV3	22/384
	DEC MicroVAX I	MV4	22/384
Laboratory for Terrestrial Physics/620	DEC VAX-11/780	LTP	16W/N70
NASA Space and Earth Sciences Computing Center (NSESCC)/630.1	IBM 3081K (CPU0)	SCFVM	1/6
	(CPU1)	SCFMVS	1/6
	Amdahl 470 V/6-II	VPFVM	22/G87
	Amdahl 470 V/7B	VPFMVS	22/G87
	CDC CYBER 205	CY2	22/G87
National Space Science Data Center (NSSDC)/630.2	DEC VAX-11/780	NSSDC	26/121
	DEC PDP-11/23	OPTDSK	26/121
	MODCOMP Classic II	MODCOMP	26/121
Goddard Image and Information Analysis Center (GIAC)/630.3	DEC VAX-11/780	MPP	28/W189
	DEC VAX-11/780	IAF	28/W294
	DEC MicroVAX I	MV5	28/294
	Sun Microsystems Workstation	SUN2	28/W120R
Laboratory for High Energy Astrophysics/660	DEC PDP-11/70	LHEA	2/S214
	DEC PDP-11/44	BBXRT	2/264
	DEC PDP-11/44	EGRET	2/56
	DEC PDP-11/34	GRIS	2/210
Laboratory for Astronomy & Solar Physics/680	DEC PDP-11/44	STARS	21/GC21
	DEC VAX-11/750	CHAMP	21/GC21
Laboratory for Extraterrestrial Physics/690	DEC VAX-11/730	SIRIS	2/234
	DEC VAX-11/785	LEPVAX	2/S114
Applied Engineering Division/730	DEC VAX-11/780	CSDR	11/E240

Figure 2. Partial List of GSFC Computers to be Initially Interconnected via the LACN



GSFC LACN ARCHITECTURE

Figure 3. GSFC LACN Architectural Reference Model

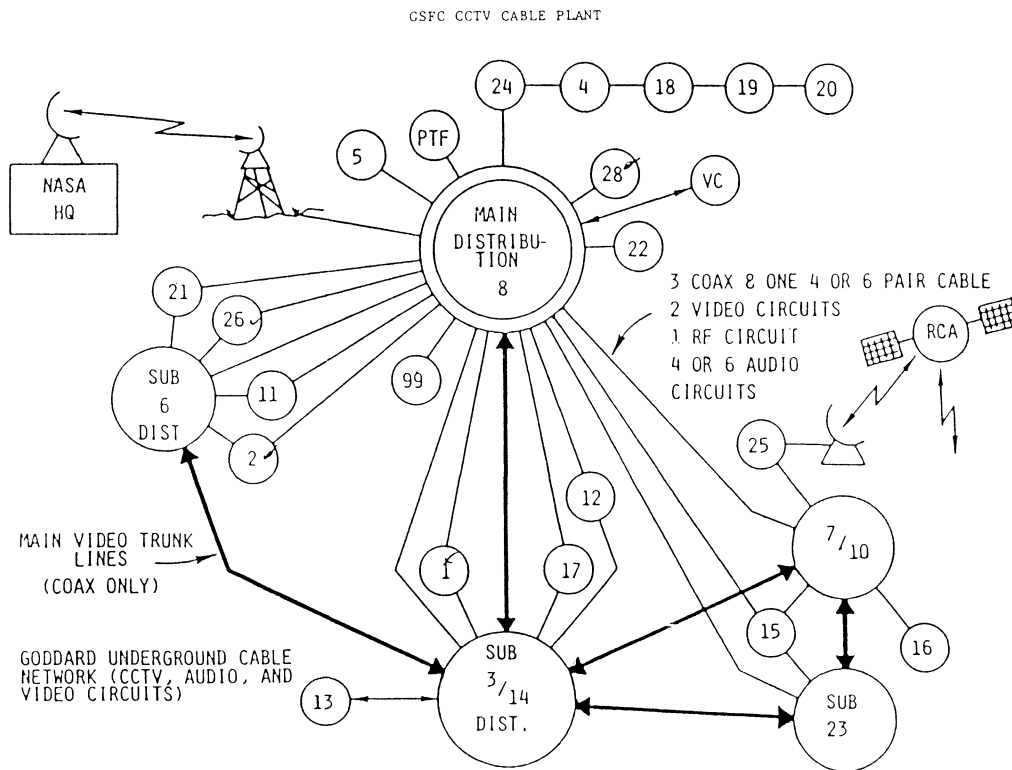


Figure 4. GSFC CCTV Underground Coaxial Cable Plant

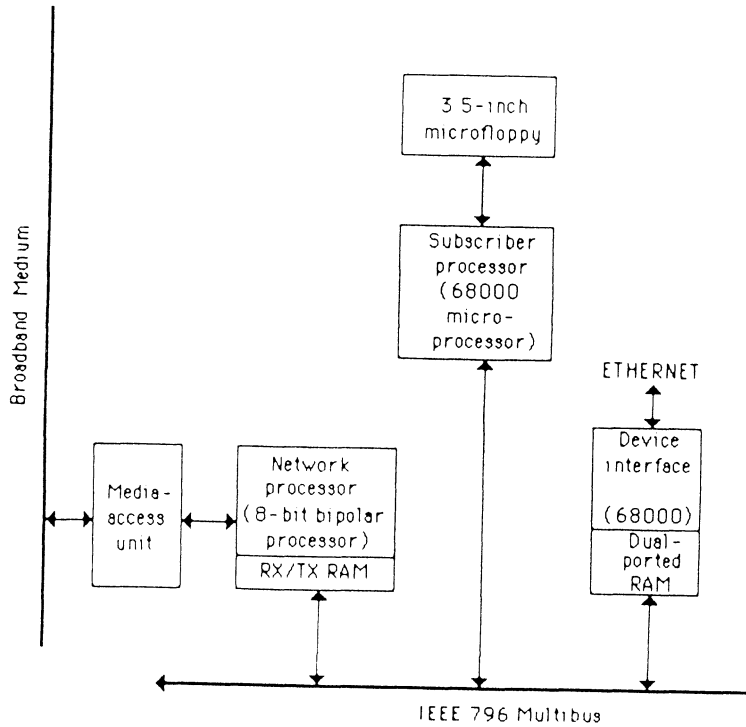
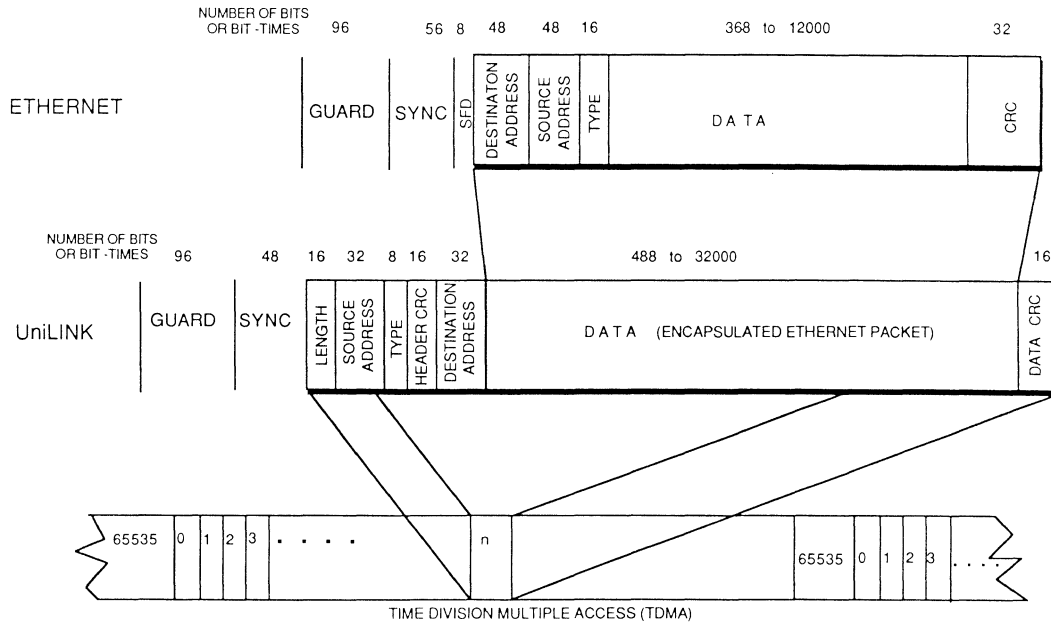
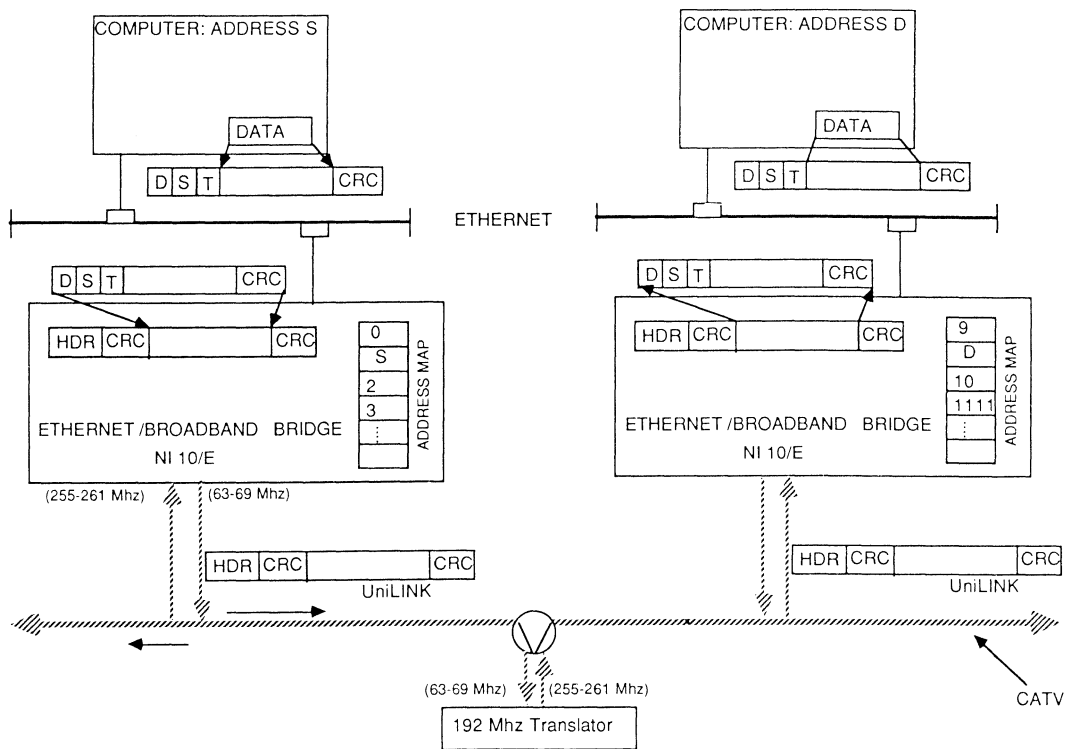


Figure 5. NI10/E Ethernet Bridge Configuration



GSFC LACN DATALINK PROTOCOLS

Figure 6. UniLINK Encapsulation of Ethernet Packets



DATA TRANSMISSION ON THE GSFC LACN

Figure 7. End-to-End Transmissions through GSFC's LACN

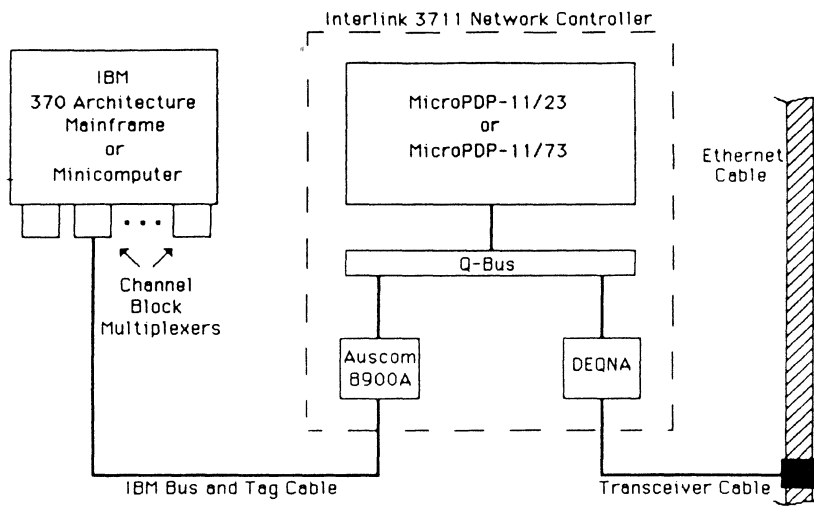


Figure 8. Interlink IBMvs/DECnet Gateway Interface to GSFC LACN

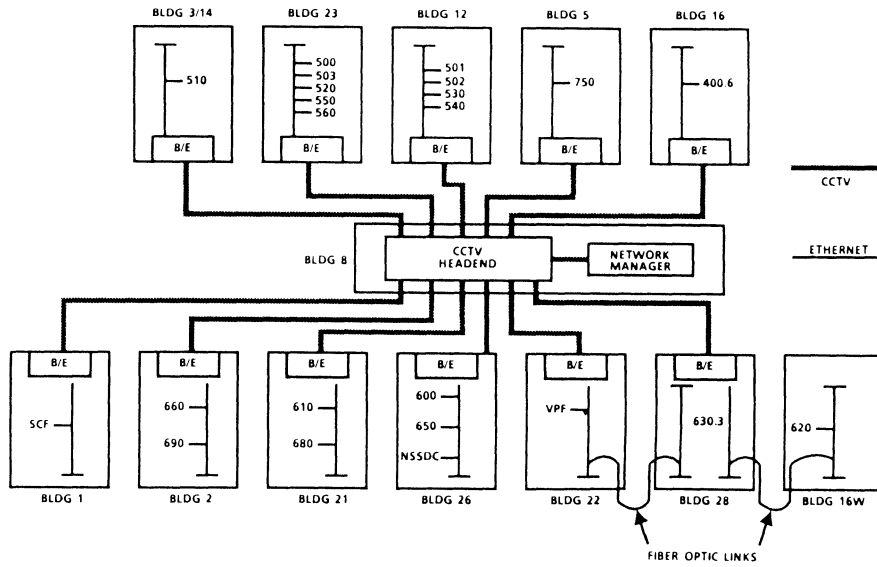


Figure 9. Current GSFC LACN Configuration

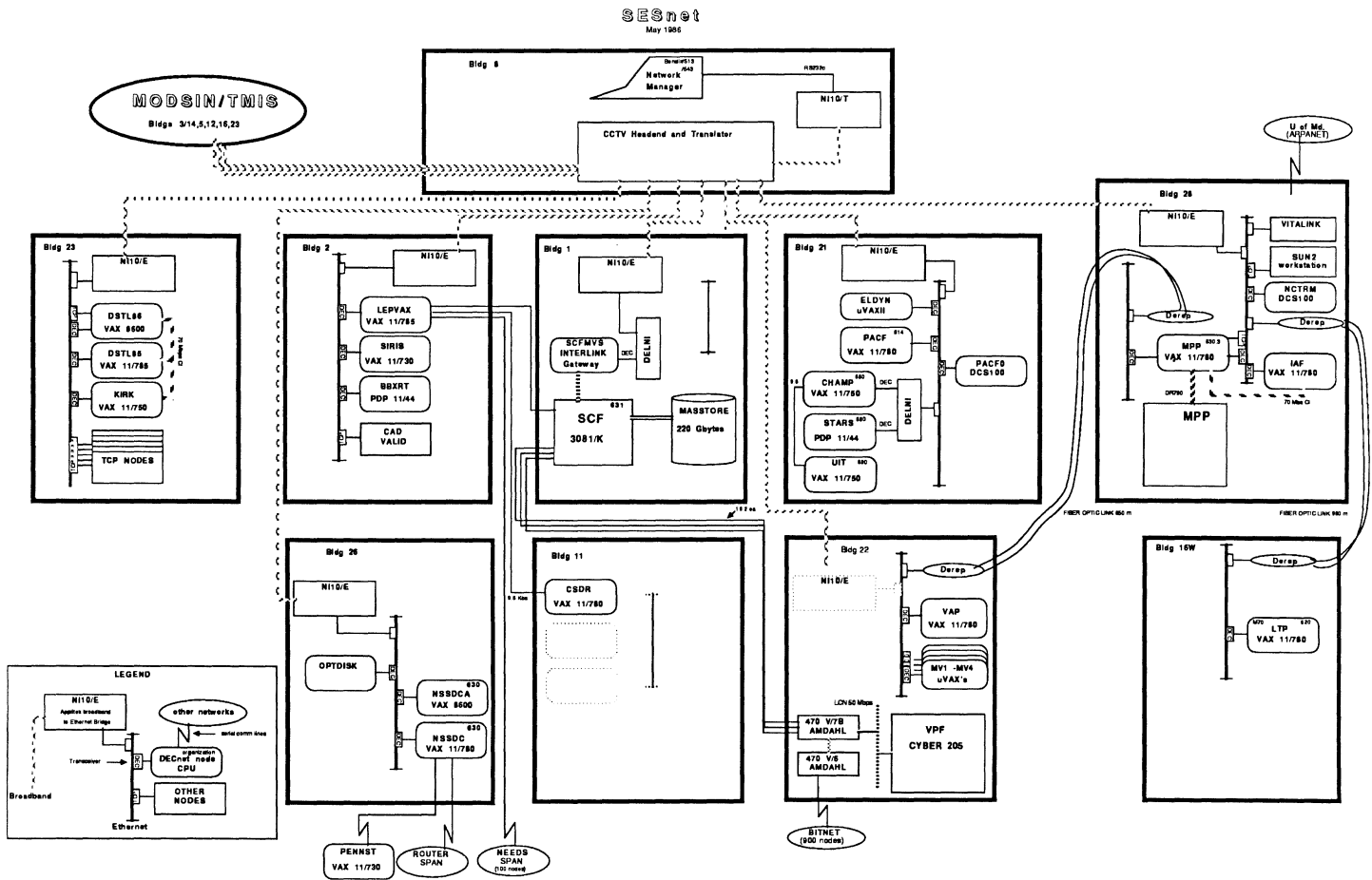


Figure 10A. GSFC LACN Intra-Building Ethernet Networks Using Channel 3/P

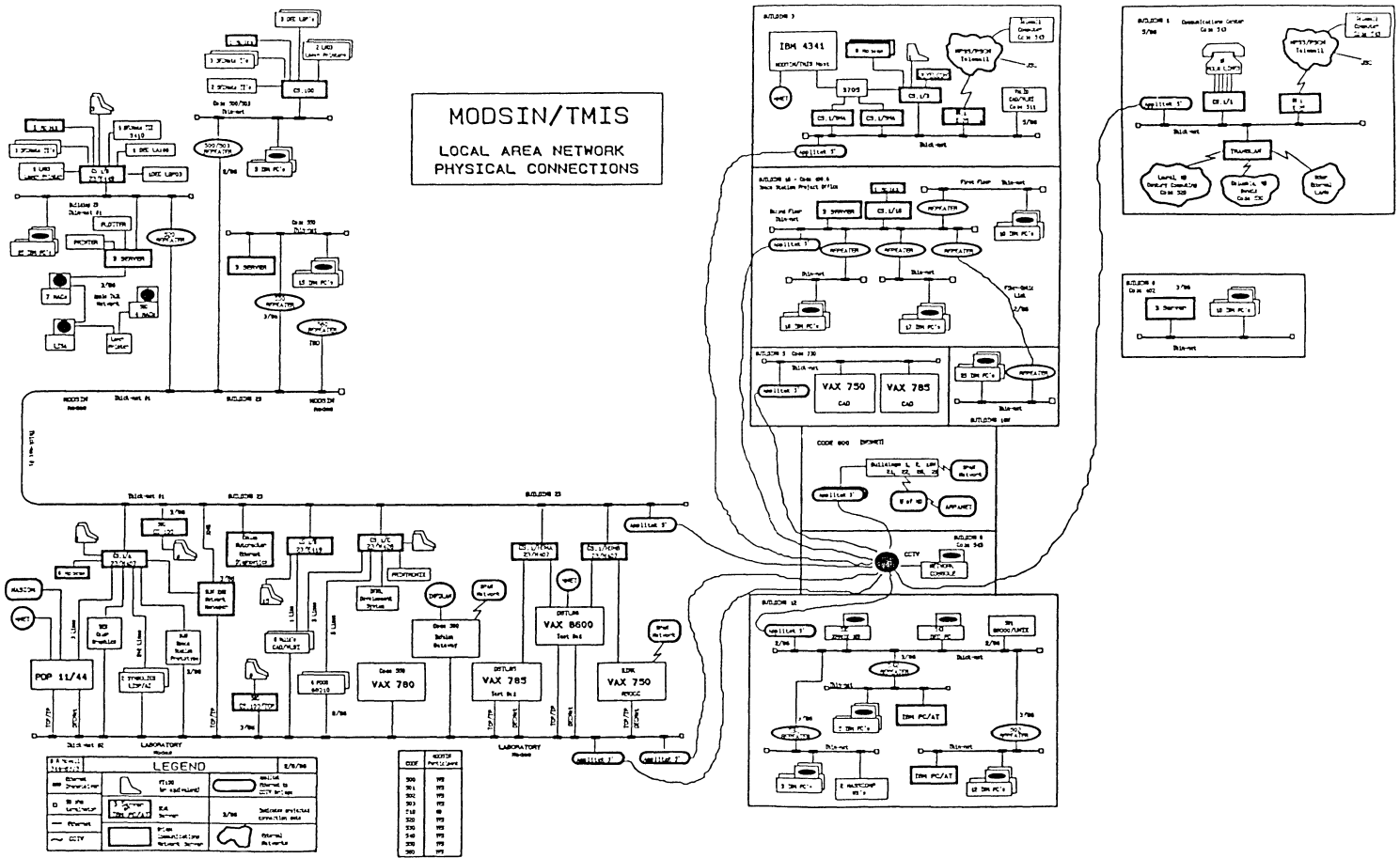


Figure 10B GSFC LACN Intra-Building Ethernets Using Channel 5'/S

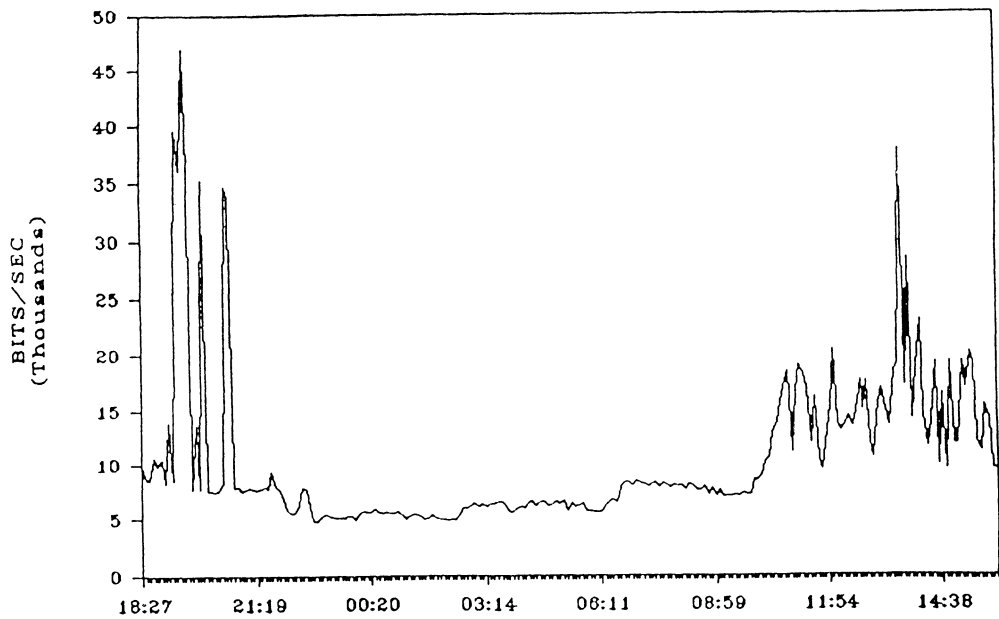


Figure 11. Example Display of Data Traffic Activity on One Inter-Building Channel of GSFC's LACN; 6 min averages are plotted as a function of time.

Space and Earth Sciences Local Area Network (SESnet)
 10 Mbps Digital Communications Subnet
 with Nodes Running DECnet
 as of March 27, 1986

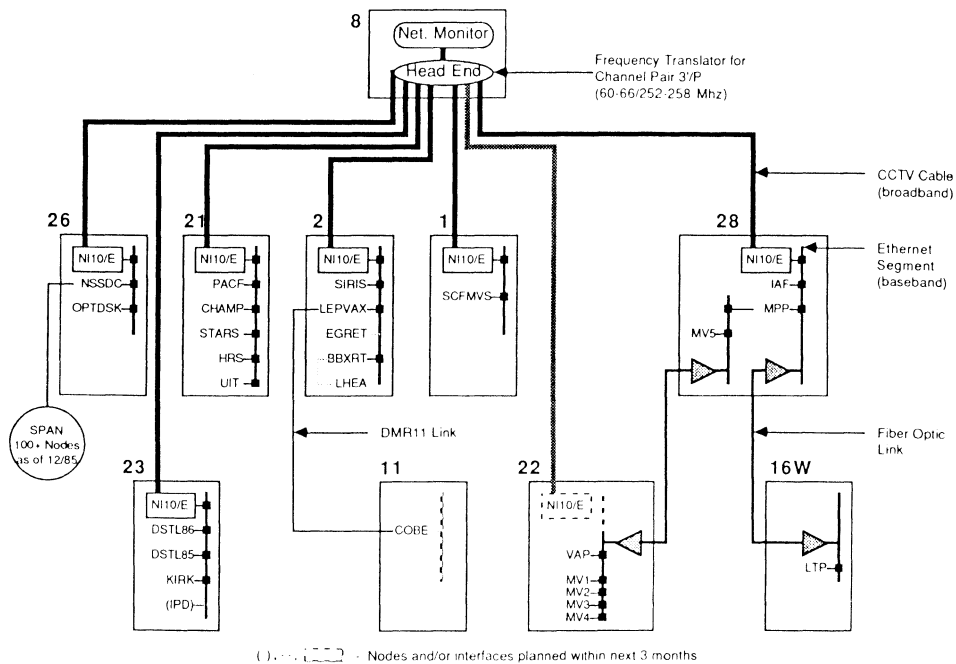


Figure 12. DECnet-Enabled Computer Networking in GSFC's LACN

SESnet DECnet Interconnections with NSESCC & SPAN

Space and Earth Sciences Local Area Network
 Ethernets, CCTV & Applitek NI10/E's of 10 Mbps Bus Not Shown

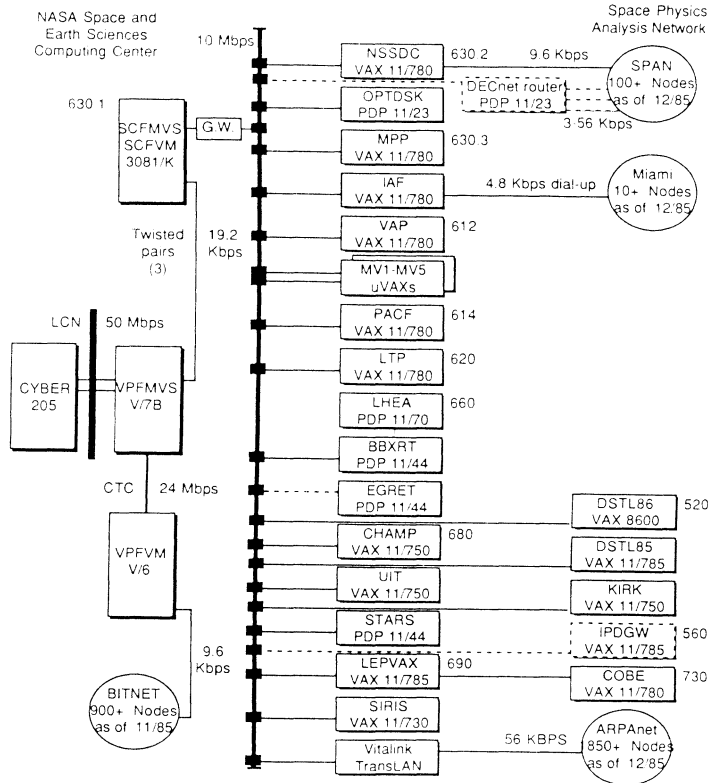


Figure 13. DECnet-Enabled Logical Topology of GSFC's LACN

Space and Earth Sciences Local Area Network (SESnet)
 10 Mbps Digital Communications Subnet
 with Nodes Running TCP/IP
 as of March 27, 1986

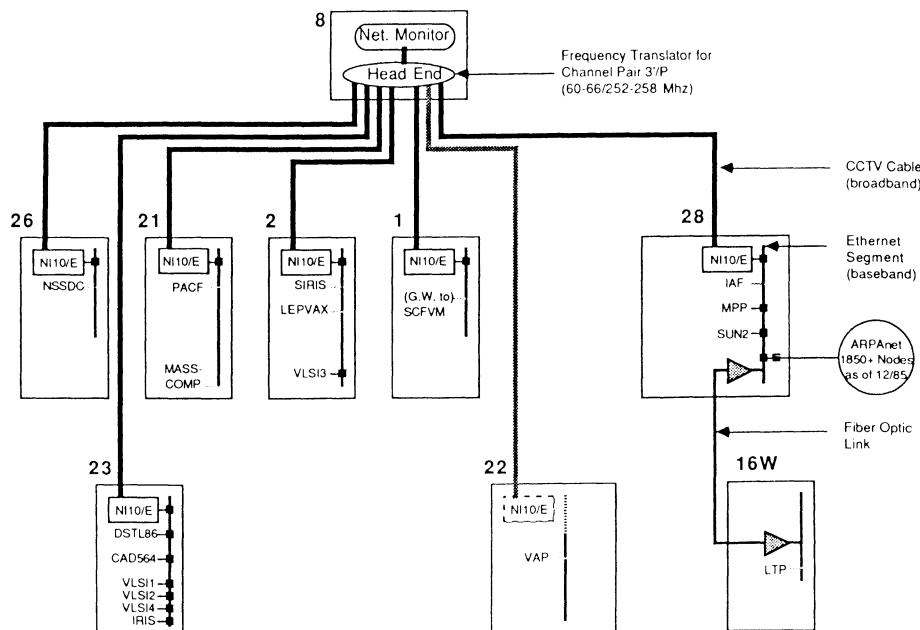


Figure 14. TCP/IP-Enabled Computer Networking in GSFC's LACN

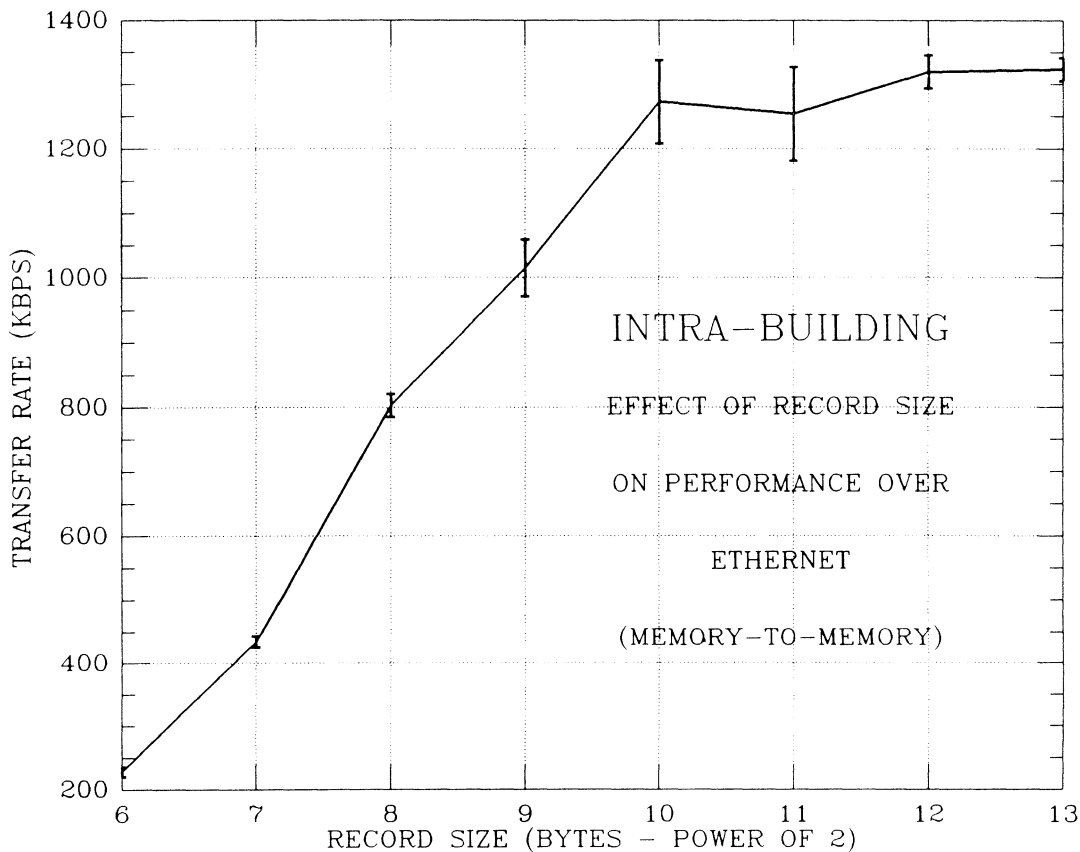


Figure 15. Memory-to-Memory Throughput Performance Using Different Record Sizes

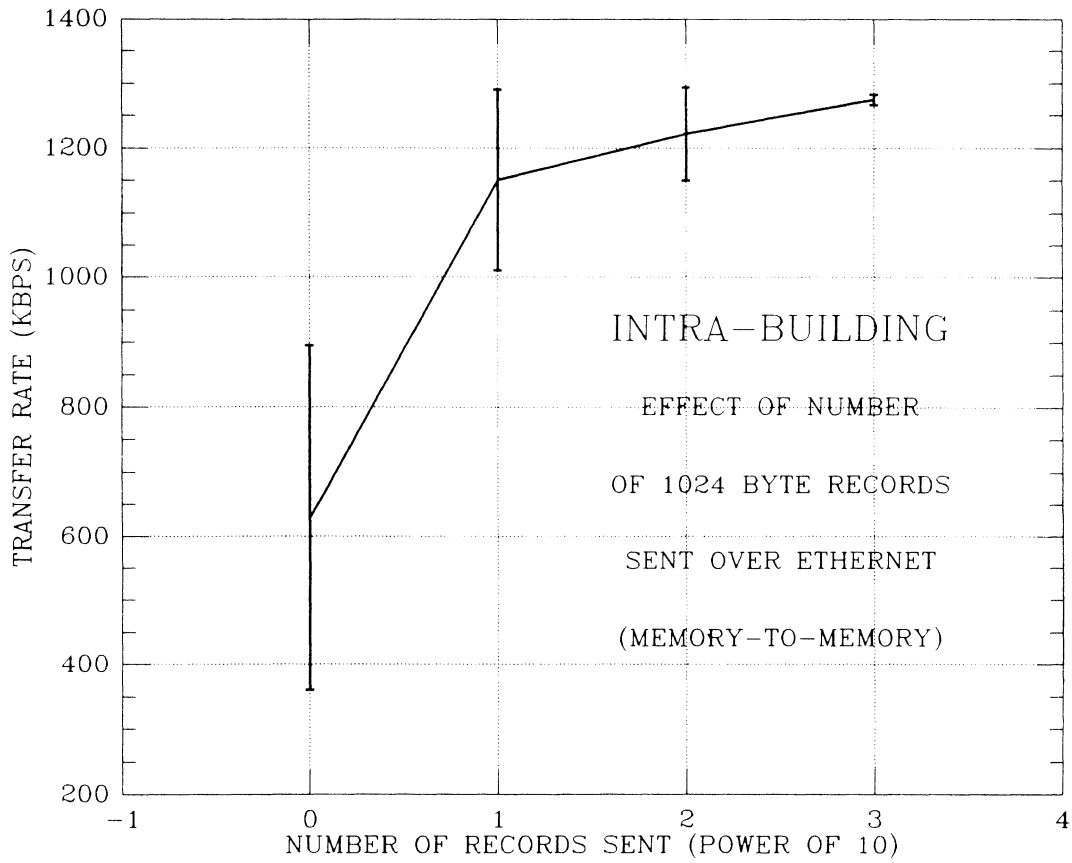


Figure 16. Memory-to-Memory Throughput Performance Using Different Record Counts

SESNET PERFORMANCE DATA
Interlink Throughput Performance
 Record size X Duration (Number of records sent) X Rate
 (Kilobits/Second)

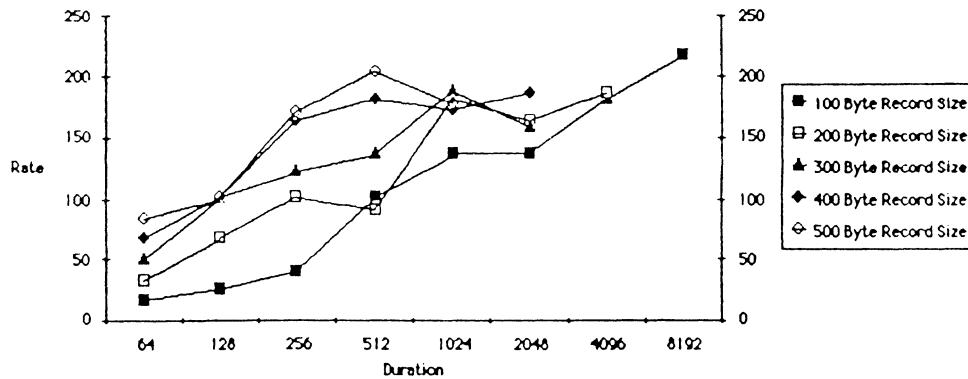


Figure 17. Inter-Building Disk-to-Disk Throughput Performance Through the Interlink Gateway

IBM 3270 TECHNOLOGY USING A BROADBAND LAN

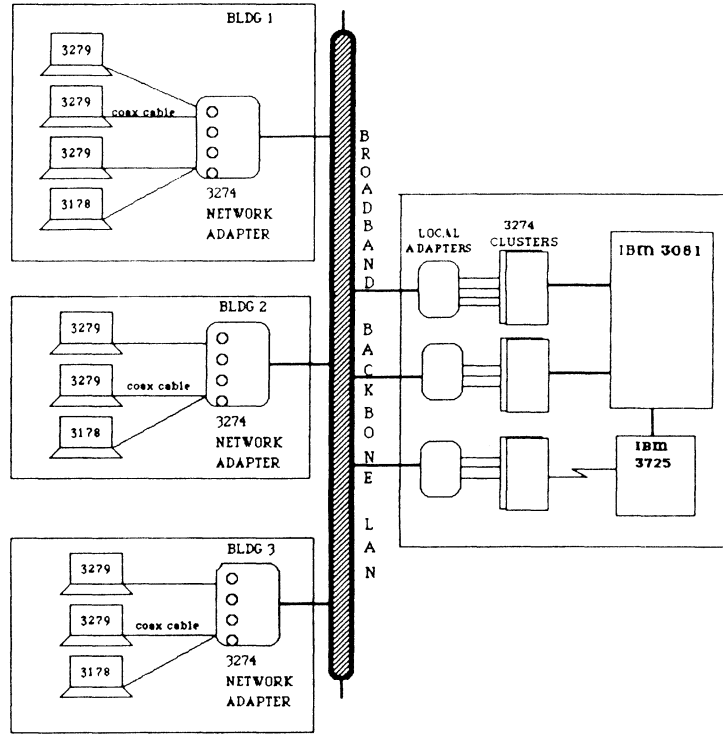


Figure 18. IBM 3270 Technology Using GSFC's Broadband LAN

CODE 630 INSTALLED ETHERNET AREAS
(AS OF 2/26/85)

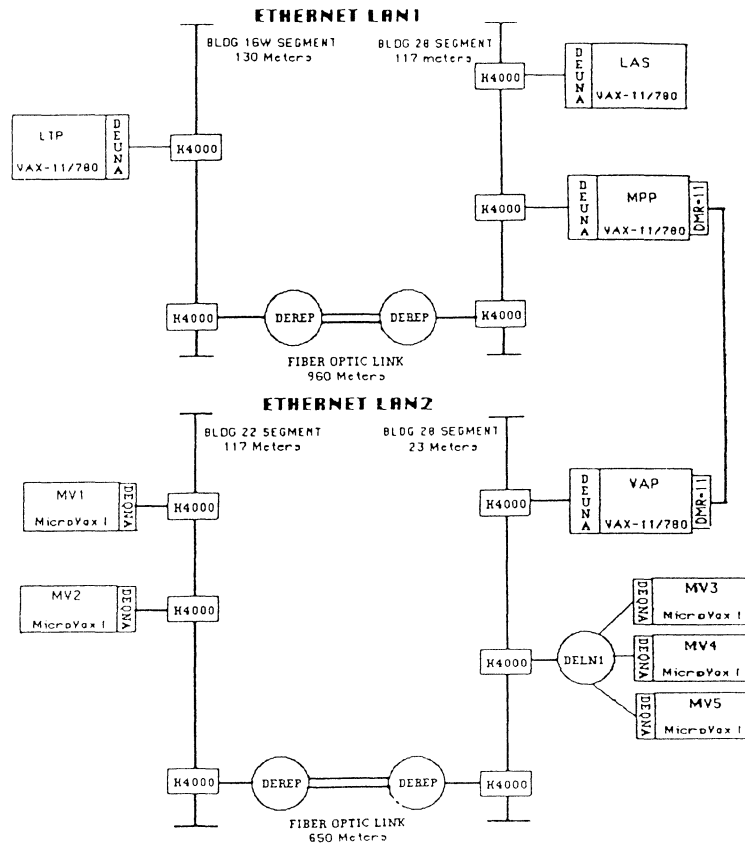


Figure A1. Initial GSFC LACN Links Using Fiber Optic Cables

IEEE 802.4 FREQUENCY ALLOCATION
 FOR BROADBAND LANs
 (EIA TR40.1)

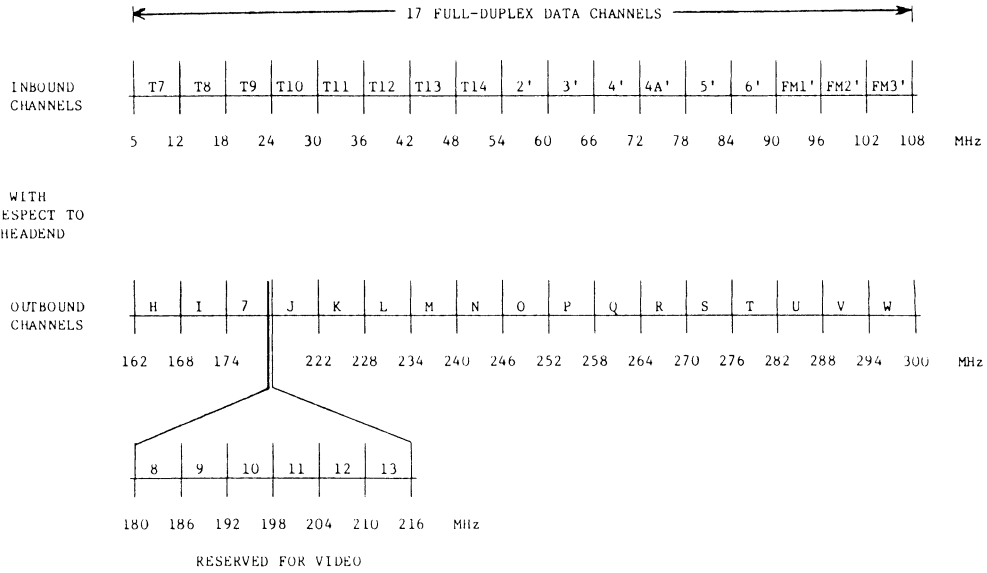


Figure A2. Mid-Split Frequency Allocations for 5-300 Mhz Band

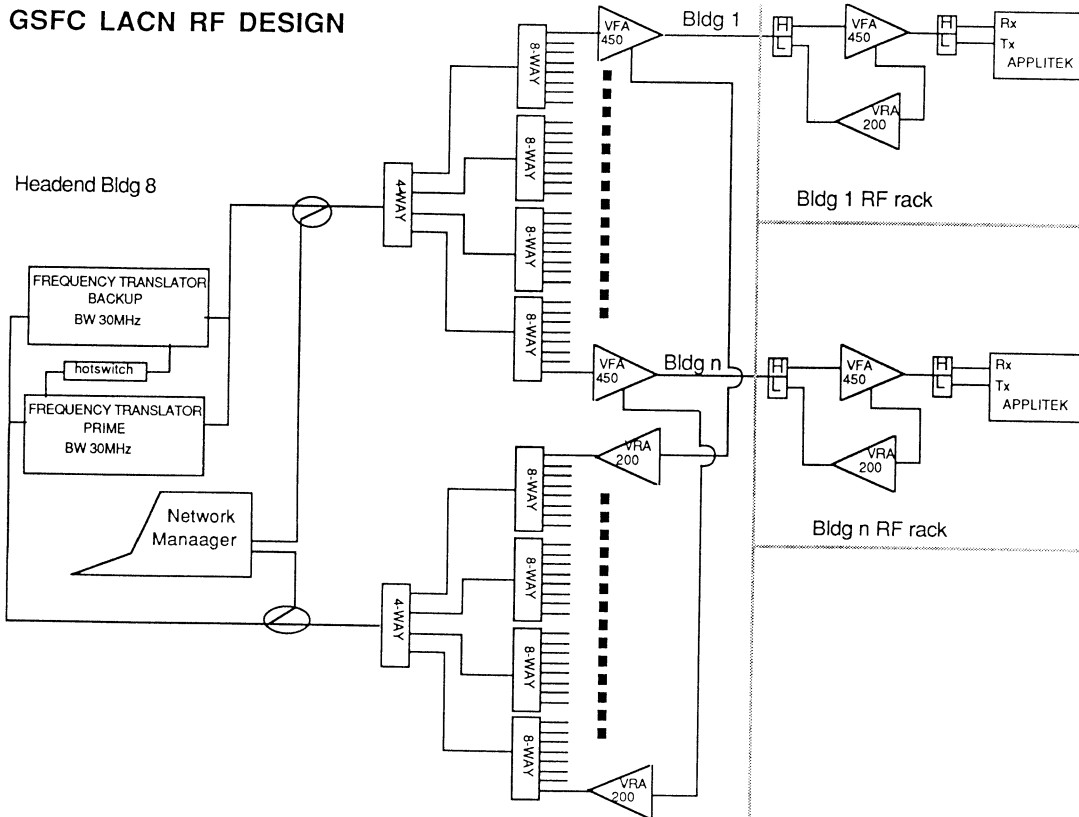


Figure A3. GSFC LACN RF Design

OFFICE AUTOMATION SIG

ALL-IN-1: A New Road to Effective Applications

Barclay Brown
Digital Equipment Corporation
Charlotte, NC

As application development costs rise and the need for applications becomes more acute, data processing professionals are searching for better ways to develop the many applications their users yearn for. This paper discusses, in a semi-technical style, the capabilities of Digital's ALL-IN-1 to help you develop data processing applications quickly and easily by freeing you from the more mundane parts of the application process.

Webster's New World Dictionary defines the word *application* as a noun, meaning a putting to use, also indicating continued mental or physical exertion.

As the 1980's become the years of new applications of computing and information technology, Webster's dictionary definition of the word application seems to fairly hit the mark in describing the experiences of many software development groups. Fortunately, there are on the horizon new tools for application development that promise greater programmer productivity, enabling applications to be developed more quickly and efficiently, and allowing for easy later modification. So-called fourth generation languages, application environments, screen handling systems, report writers, graphics packages, data management tools, data base systems and even automatic application generators have been making the rounds in software circles lately.

A Tool of the Trade

In this article, we will discuss one such tool. You have probably already heard of it, but you, like many, including even users themselves, might not be aware of its ability to aid in the application development process. I'm talking about Digital's

ALL-IN-1, the leading integrated office system, with over 35% of that market to its credit. ALL-IN-1 achieved this distinction with clearly superior word processing, electronic messaging, time management, desk management and communication capabilities, but there is yet another important side to this comprehensive system. A conceptual understanding of ALL-IN-1's underlying application architecture will reveal some astonishing tools available to application developers.

To say it another way, the standard equipment of ALL-IN-1: word processing, electronic messaging, file cabinet and time management are actually applications that were created using ALL-IN-1's built-in application development tools. These very same tools are available to your application developers, who, with a little practice, can use them to prototype or create new applications quickly and easily! As a by-product, these custom applications become part of your existing ALL-IN-1 office system.

The Application Road: Country Lane or Autobahn

If we look at the real costs of developing computer applications, we begin to find that the preliminary phases of specifying, designing and prototyping, and the later phases of support, training and future

enhancements consume much more time and resources than the system coding itself. Digital's experience in bringing software to the market bears this out. Most of the costs are incurred before or after the actual coding of the application.

Using an application development tool such as ALL-IN-1 allows you to move an application from the idea stage to the prototype stage quickly--you get a feel for how the application will look and perform. We call it creating a user interface. This application shell can then be demonstrated and when constructive input is received, the shell can be easily modified and presented again. It is much easier to evaluate an application in its real environment--on the screen of a terminal, than in a dry functional or design document. When the final design is agreed upon, there is no need to scrap the prototype user interface in order to begin coding. The user interface can simply be fleshed out with whatever features are necessary to give the application its complete functionality.

Since ALL-IN-1 gives you--the application developer--access to the same tools that the developer of say, ALL-IN-1 Word Processing had, the applications you develop can look and act just like ALL-IN-1's standard applications. The user interface will be consistent, keystrokes will be similar, and on-line help and computer based instruction lessons can be added easily--your application will appear as if it is an integral part of ALL-IN-1, which as you will soon see, it is indeed!

How the ALL got into ALL-IN-1

ALL-IN-1, as evidenced by its standard equipment applications, serves as a Grand Central Station for many other tools and packages in the rich VAX/VMS application software collection. Together with ALL-IN-1, these packages form the VAX Information Architecture (VIA). *Via*, in Latin, means way or road so VIA is literally an application highway down which your application may travel, making use of the many roadside services and conveniences, and arriving at its destination complete, safe and most of all, on time.

I would now like to conduct a tour of this highway, pointing out the various capabilities of ALL-IN-1 in the application development environment. The discussion will be only slightly technical in nature, since I feel that you want to know exactly what might make ALL-IN-1 the right choice for your application. We will consider in turn the several types of ALL-IN-1 forms, data management, ties to other applications, scripts, help, computer-based

instruction, and ALL-IN-1's built-in callable applications.

What's on the Menu Today?

Of the dozen or so different types of ALL-IN-1 forms, the **menu** form is by far the most common. Most probably the first thing you see when you enter ALL-IN-1 is a menu. ALL-IN-1 menus present the user with a set of options as well as some background information such as the user's name, job title, the date and perhaps a current item or document. To explain how menus work, I must digress for just a moment and tell you about how ALL-IN-1 uses FMS.

The VAX/VMS Forms Management System, FMS, is a software system belonging to the VIA product set which provides for the creation, modification and display of screens of information. An FMS form consists of two parts: the information that appears on the screen of the terminal and what is called *named data*, which does not appear on the screen. The named data portion of a form is a set of data items, each with a name. FMS allows this data to be stored with a form but does not interpret or process this data in any way. ALL-IN-1, however, makes extensive use of this named data area to store functions, directives and qualifiers which serve to direct the processing of that form by ALL-IN-1.

Now, let's get back to menu forms. If you look at the named data of an ALL-IN-1 menu form (you can look at the named data of any form by pressing **GOLD N**), the first thing you will see is a **.TYPE** directive specifying the type of MENU. The other items on this same line, each beginning with a slash (/) are called *form qualifiers* and are used to tell ALL-IN-1 how to load or specify the various informational fields on this menu. In Figure 1, the **/CHOICE** qualifier is used to tell ALL-IN-1 that the user's choice will be entered into the field named **CHOICE**, while the other qualifiers specify that the user's name and the day of the week are to be displayed in the **USER** and **DAY** fields, respectively. Following the **.TYPE** directive are other named data items. Each of these tell ALL-IN-1 what to do when the user types one of the menu options shown on the menu screen. It's very simple. In Figure 1, we see that if the user types an **E** on this menu, ALL-IN-1 will begin to execute (do) the script called **EDITREC.SCP**. We will talk more about functions and scripts later, but this example should serve to illustrate how easy it is to define menu options.

Once you set up this list of menu options in named data, ALL-IN-1 handles all the menu processing and flow control for you. It provides the user interface, menu tree structuring, automatic **EXIT SCREEN** key processing and help support as well as menu chaining functions, all without additional coding.

It is also important to note that not all menu options available to the user need appear on the screen. You can cause certain sets of menu options to be accessible from specific menus, or from anywhere in ALL-IN-1. Adding, removing or changing menu options is probably the simplest kind of customization you can do with ALL-IN-1.

Before I touch on the other types of ALL-IN-1 forms, I want to say just a word about how easy it is to modify forms. Included with the ALL-IN-1 system is a menu, very similar to the word processing menu, which allows you to select, create, edit and delete all kinds of ALL-IN-1 forms. This menu uses some of the capabilities of FMS, but also adds some functions not available from FMS, such as the ability to edit named data with your chosen ALL-IN-1 text editor. You can access this Forms Development menu by typing **FD** from any ALL-IN-1 menu.

```
Name: .TYPE
Data: MENU /CHOICE=CHOICE /USER=USER
      /GET=DAY,OA$DAY
Name: E
Data: DO EDITREC
```

Figure 1

Transactions Made Easy

The second major kind of ALL-IN-1 form allows you, the application developer, to handle files, records and data transactions easily. It's called the *entry form* and, as its name implies, it is used to enter data into files. To create an entry form, just lay out a field on your form for every field in the data file. Arrange the fields in the order you want them in the record in the file itself. Then, using the **.FILE** directive in named data, you will specify the name of the data file and the name of the field to be used as the primary key for the file. Figure 2 is a sample of some named data from an entry form.

```
Name: .TYPE
```

```
Data: ENTRY /MODE=UPDATE
```

```
Name: .FILE
```

```
Data: MYDATA.DAT,KEY1
```

Figure 2

Once you have created this entry form, you can use the ALL-IN-1 **CREATE** function to create a new, empty data file with a layout that exactly matches the entry form. You have just created a full transaction processor for that data file! When you ask ALL-IN-1 to display this new entry form (with a **FORM** function), ALL-IN-1 does all the work of handling the user's interaction with the data file. It works like this.

First, ALL-IN-1 makes the key field reverse video and prompts you to enter the key field or fields. ALL-IN-1 validates this against the data file, and if the record exists in the file, all the other data fields are filled in from that record. Then ALL-IN-1 asks you to select a transaction: **add, change, delete, inquire** or **copy**. If you select **add** or **change**, you can navigate around the fields in the form, in any order, and change fields. When you press **RETURN**, the record is written back to the file. **Copy** allows you to duplicate any record in the file.

Using ALL-IN-1 entry forms eliminates all worries about managing data files. Multiple entry forms designed for use by different kinds of users can all manipulate records in the same file. Multiple users can access the same data file concurrently. You can even allow users to input fields in an order different from the data file's field order.

Sometimes the need for a simple application can be met by using only ALL-IN-1 menu and entry forms. More often, though, menu and entry forms are just a part of the entire ALL-IN-1 application necessary to meet your business need.

In Search Of Excellent Data

In most applications, the next logical step after allowing users to input, change and delete records from data files, is to allow them to search these data files in some flexible manner. This ability is provided by ALL-IN-1 *select* forms. Select forms allow the ALL-IN-1 application developer to set up a

flexible way to allow users to get listings of data records that match certain search criteria.

A select form typically has two areas, though these two areas may be on two different forms. The first area is a scrolled region meaning that chosen entries scroll through like scenery through the portal of a cruise ship. The other area contains a set of fields into which the user will enter the selection criteria. These fields will be matched against the data file by means of a *record selection expression*, and may be any subset of the total number of fields contained in a record from the data file. As shown in Figure 3, the named data of a select form looks a little more complex than the menu or entry form. But it's still pretty simple.

The word **SELECT** tells ALL-IN-1 that this is a select form and that what follows is to be taken as a record selection expression (RSE). This RSE is a boolean expression and compares the contents of the fields on the select form with the data in fields in the data file. In this case we are comparing the fields number and name with their counterparts in the data file. The == notation means exactly equal, while the <=> indicates a containing comparison. Thus, a record would only match this RSE if the number matched *exactly* and the name *contained* what was entered on the select form.

Records that match the rse are displayed in the scrolled region at the top of the select form according to the **SEL_STYLE** function. In this case, the name is to be displayed followed by the number. The **/CHOICE** qualifier tells ALL-IN-1 to allow the user to choose one of the displayed entries by line number. ALL-IN-1 stores the key of the item chosen in a special symbol for later use by the application.

Name: .TYPE

```
Data: SELECT FOR CUSTOMER_ENT WITH .NUMBER
      == NUMBER AND .NAME <=> NAME DO
      SEL_STYLE .NAME .NUMBER /STYLE = CHOICE
```

Figure 3

Other Interesting Form-ations

Menus, entry forms, and select forms, are probably the most commonly used ALL-IN-1 forms in most applications. There are, however, several other kinds of ALL-IN-1 forms that may come in handy when designing applications. For instance, the *argument* form is a simple form with fields into

which the user can enter information. This information is then passed to ALL-IN-1 via ALL-IN-1 *symbols* (we will discuss symbols a bit later) and may be used to build records later to be written to a file, or to specify other input to an application. Argument forms are in essence multi-purpose information gathering forms.

Calc forms are used primarily to allow the user to do various kinds of pre-set calculations. The *desk* form, a general purpose variation of the calc form, is a full-featured desk calculator. *Help* forms present the user with helpful paragraphs of information written about each ALL-IN-1 application. *List* forms are used to display text information to the user, screen by screen. *Edit* forms allow the user to edit text files and documents using one of ALL-IN-1's built-in text editors or word processors. You can use these various kinds of ALL-IN-1 forms, in any combination, to create your own specific application. The form types are designed to do most of the work for you, making your development task easier and faster.

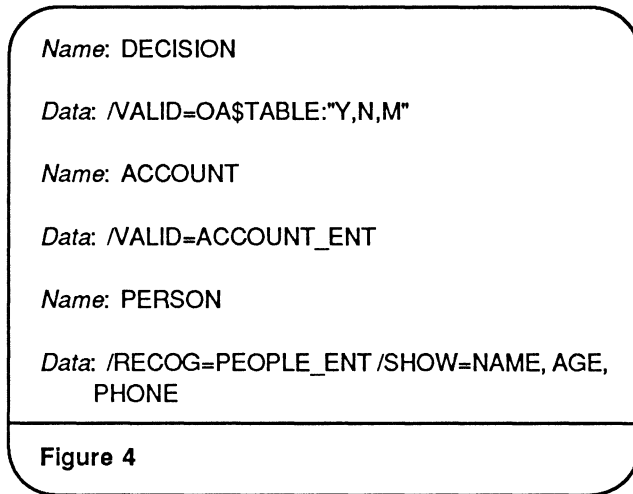
In order to use a form in an ALL-IN-1 application, you must first put that form in a *form library*. The VAX/VMS Forms Management System allows you to create and maintain these libraries. Each ALL-IN-1 user, then, can be given access to one or more forms libraries. This ability to selectively grant access to forms libraries is an important security feature. Suppose you have ALL-IN-1 different applications on your system for sales, for services and for engineering. As would usually be the case, all the menus, entry forms and other forms for the sales application would be in a form library separate from the services and engineering form libraries. A sales manager would be granted access to only the sales form library, while an engineer might only be granted access to the engineering form library. An application developer might be given access to all of the form libraries. In this way, you can have as many applications as you want on your system, and limit their use to specific sets of users.

Do you recognize this data?

The next stop on our tour of the ALL-IN-1 application highway brings us to the subject of data *recognition* and *validation*. As you have probably seen, entering information into fields on forms is the primary (though not only) way of gathering information from users to be used in applications. Often, the application designer would like to allow only certain entries in a field. Some of the many

ALL-IN-1 field qualifiers make this an easy feature to include in your applications. In the simplest case, we would like to prevent our user from entering anything but a **Y** for yes, an **N** for no and an **M** for maybe. In the named data of Figure 4, the field named **DECISION** shows how to do this with the **/VALID** field qualifier and the **OA\$TABLE** symbol.

Note that this named data could be a part of an entry, argument or select form. Field qualifiers for validation and recognition operate the same on all of these kinds of forms.



Many times you would like to allow the user to enter any of the values contained in another file. For example, in an invoicing application, you need to prevent the user from entering any account number that is not on file in the accounts file. In Figure 4, the field **ACCOUNT** is validated in this way. **ACCOUNT_ENT** is the name of the entry form that points to (remember the **.FILE** directive?) the accounts file. This simple **/VALID** qualifier causes ALL-IN-1 to take whatever is entered in the **ACCOUNT** field and compare it to all the entries in the accounts file. If the entry is not found, an error message is displayed and the user is not permitted to leave the field until a valid entry is made.

Extending the validation concept, a user would like to be able to see a list of the valid entries for a field. The **/RECOG** field qualifier makes this easy. Again referring to Figure 4, we have added **/RECOG** and **/SHOW** qualifiers to this field. When the user moves onto this field, he/she may press **GOLD L** to display all the entries in file pointed to by **PEOPLE_ENT**. The **/SHOW** qualifier determines which fields in the people file are displayed, in this case the user will see the person's name, age and phone number. If the user

knows the entry begins with a **B**, he/she can enter a **B** in the field before pressing **GOLD L**. This produces a list containing only entries beginning with **B**.

In any case, once this recognition list is displayed, the user can choose one of the entries on the screen by its line number and the entry will be copied into the field. Recognition and validation are prime examples of application development capabilities built into ALL-IN-1 that increase consistency among applications and provide user friendly features without a heavy burden on the application programmer!

The ALL-IN-1 Function Family

We have already alluded to ALL-IN-1 *functions*. Each ALL-IN-1 function has a brief, unique name and performs a specific function useful to the application developer. For example, there are ALL-IN-1 functions to create files, compute formulas, display or edit text, search data files, manipulate symbols, handle form libraries, prompt the user for information and even convert data files from one format to another. There are a total of 152 ALL-IN-1 functions available for the application developer.

Many ALL-IN-1 functions require information to be passed to them in the form of ALL-IN-1 *symbols*. Symbols are a lot like variables in a programming language, but they are in many ways, easier to use. Symbols have names chosen by the application developer, and can contain up to 255 characters of alphanumeric information. Symbols are the main vehicle used to move data between ALL-IN-1 functions and fields on forms. Field qualifiers such as **/GET** and **/PUT** are used to move data between specific fields and symbols.

Sticking to the Script

Although multiple ALL-IN-1 functions can be invoked from named data, it is usually necessary to construct more elaborate procedures to accomplish certain application tasks. It is for this purpose that ALL-IN-1 *scripts* were created. There are two kinds of scripts: *do* scripts, invoked by the ALL-IN-1 **DO** function, and *script* scripts, invoked by the ALL-IN-1 **SCRIPT** function.

Do scripts are in essence procedures consisting of ALL-IN-1 functions intermixed with script directives such as **.IF**, **.LABEL** and **.GOTO**, giving the script writer programmer-like power. But remember that many ALL-IN-1 functions perform

elaborate tasks that would take many lines of code in a "normal" programming language. This means that complex procedures can be usually be written in small, straightforward scripts.

Script scripts are similar to *do* scripts, but are used for different tasks. *Script* scripts can simulate keystrokes as if they were typed by the user. This makes this kind of script ideal for *user defined procedures*. Sequences of operations that are performed often can be automated by creating a script that acts like a robot programmed to press the keys for you.

The Computer as a Teacher

The other major application of *script* scripts is in the area of computer-based tutorial lessons. Scripts can interact with both ALL-IN-1 and the ALL-IN-1 user independently, as well as present windows of information overlaid on ALL-IN-1 forms. Using these capabilities, you can develop computer-based tutorials that use the application itself, instead of simulating the application. Extending this concept a bit further, you can write tutorials that guide the user through a real task. For example, the lesson on creating documents can guide the user through the process and when the lesson is over, the user has actually created a real document.

Both ALL-IN-1 standard equipment applications and other ALL-IN-1 applications developed by Digital include these tutorials for all major user functions. Once again, you, the application developer, can make use of the same capabilities to create computer-based training courses for applications you develop using ALL-IN-1.

Help Is On The Way

In recent years there has been a trend toward on-line documentation of application systems and away from large printed document sets. Computer-based tutorials are a step in that direction, as is on-line help. ALL-IN-1 provides an extensive on-line help system that can be used to create and process help for applications you develop within the ALL-IN-1 framework. To do this, write your help text using your favorite word processor. You may write text paragraphs for of any length for whole forms, menu options, and individual fields. You will then copy this text into ALL-IN-1's help library tagging each paragraph as to the form, option or field it covers. You can also define cross references in

your help. For example, when the user presses the help key for the **create document** menu option, you might would also like to offer him help on *text editors*. When you do this, your user will see the subject *text editors* under the *additional topics* heading in the help window.

By just creating your help text and putting it into the help library, you have provided a powerful on-line assistance tool for your users. When the users press **GOLD H** anywhere in the application, a help window will appear on the screen containing your help text. Below this, a list of *additional topics* will appear containing related topics chosen by ALL-IN-1 in addition to the *additional topics* you have specified as described above. ALL-IN-1's help system is yet another work-saver for the application developer!

Produce Your Own Junk Mail

Quite often office automation applications involve the creation of specialized documents, containing both standard text and data items from files. A common example is the mass mailing sales letter. You would like to send the same text to everyone whose name and address are stored in a data file. Or you might want to create legal contracts, using selected standard paragraphs, but filling in the client's name and some other salient facts from a client data file.

To allow you to perform these tasks, ALL-IN-1 provides a *merge* facility. It works like this. You can include, in any text file, *merge directives* that specify a data field from any file pointed to by an ALL-IN-1 entry form. Note that this capability is more powerful than a conventional list processor, which only allows data to be included from only one file.

```
<OA$DATE>

<CLIENT.NAME[OA$MERGE_KEY]>
<CLIENT.ADDRESS[OA$MERGE_KEY]>

Dear Sir:

Please call me at <PROFIL.PHONE[OA$USER]>.

Sincerely,

<PROFIL.FULNAM[OA$USER]>
```

Figure 5

Figure 5 is a simple example of this feature. First the symbol **OA\$DATE** is used to bring in the current date. Then, a client's name and address is pulled from the client data file pointed to by entry form **CLIENT**. Lastly, the user's phone number and name is pulled from the ALL-IN-1 user profile using the symbol **OA\$USER** as key into the profile file. By the way, this notation for pulling fields from data files is known as a *data set reference* (DSR) and is used elsewhere within ALL-IN-1 applications to give the application developer access to data files on a field by field basis.

You may have noticed the **OA\$MERGE_KEY** in the first few DSR's above. Using this symbol indicates that ALL-IN-1 should use the next item in the list file as the key into (in this case) the **CLIENT** file. A list file is simply a text file containing a list of the keys pointing to the records to be merged with a form document, like the one in Figure 5. Whatever the number of items in this list file, this same number of merged output letters will be produced.

You can create a selection list document by using ALL-IN-1's text editor, and typing in the keys to all the records you want merged. But there is a much easier way. The **/LIST** form qualifier can be placed on a **SELECT** form, causing ALL-IN-1 to write the keys of the records found by the **SELECT** form's *record selection expression* to a selection file. This file can then be specified when its time to perform the merge. The actual merge is performed by use of the ALL-IN-1 **MERGE** function, which requires that you specify three files. As you might guess, you must specify the form file (like the one in Figure 5), the selection list file (created by a **SELECT** form), and an output file, into which will be written your merged letters, ready for printing and mailing!

Application Access Roads

Over the years, Digital has developed many powerful VAX/VMS applications for many purposes. Rather than re-develop comparable applications for the ALL-IN-1 environment, we've chosen to provide strong links between these applications and ALL-IN-1, allowing you, the application developer, to use their functions, invisible to your user.

A prime example of this kind of integration is *Datatrieve*. Datatrieve is a powerful and very popular data query, report-writing and data manipulation language. With Datatrieve, you can easily produce a variety of reports related to the data you have stored through ALL-IN-1's entry

forms. You may be wondering, how can I use Datatrieve if I am running ALL-IN-1? Well, the Datatrieve code has been linked into the ALL-IN-1 image. This means that you have immediate access to all of Datatrieve's functions, just as you do to ALL-IN-1's functions. ALL-IN-1 even has special functions that allow you easy access to Datatrieve from ALL-IN-1 scripts and named data. For instance, you can call a Datatrieve procedure from an ALL-IN-1 script, or bring a Datatrieve variable into an ALL-IN-1 symbol.

DECgraph is another example of a fully integrated application. The easiest way to produce a graph of your user's data is to use Datatrieve to produce a load file in DECgraph's format, containing the data to be graphed. Then you can use the ALL-IN-1 **DECGRAPH** function to pass this information, along with a graph description file, to DECgraph. The resulting graph can be stored in the ALL-IN-1 file cabinet, and later printed or mailed.

You would think that ALL-IN-1 would have a fully integrated spreadsheet system as well. It almost does. Digital's DECcalc is a powerful spreadsheet system that can be used with ALL-IN-1. But unlike Datatrieve and DECgraph, it is not linked into the ALL-IN-1 image itself. So how *can* you use it? Understanding the answer to this question will not only allow you to use DECcalc, but will also enable you to utilize other applications that live outside of ALL-IN-1, from within your ALL-IN-1 applications.

Let me explain. ALL-IN-1, like any other program on the VAX, runs in the user's main process area. But VAX/VMS allows each user to have multiple *sub-processes* subordinate to this main process. ALL-IN-1 gives the application developer the ability to easily run *any* VAX/VMS application in this sub-process without leaving ALL-IN-1. In addition, the ALL-IN-1 application can transfer data to and from this sub-process application though easy to use *mailboxes*. Of course, the user never needs to know about these multiple processes, but you, the application developer, can see how this makes it possible to take *any* VAX/VMS application and make it appear as a part of ALL-IN-1. Depending on that particular application's ability to be called, you can pass parameters and data (perhaps collected from the user on an argument form) to it, and receive data back from it, all without the user's awareness.

A simple but powerful example of using the sub-process is the *DCL command procedure*. DCL is the Digital Command Language, and as you may know, DCL command procedures can perform many tasks that would otherwise require a programming

language. In addition, DCL procedures running in ALL-IN-1's sub-process can make use of ALL-IN-1 functions, and can pass information to and from ALL-IN-1 applications using symbols. DCL procedures are often a good way to specify the interaction between outside applications and the ALL-IN-1 environment.

One more thing: if you have written an application according to VAX/VMS calling standards, you may be able to link your code right into the ALL-IN-1 image, making access to your code even faster from an ALL-IN-1 application.

A Closer Look at the Standard Equipment

You've seen the ALL-IN-1 word processing menu and have probably used it. Have you ever wondered how it was developed? Well, the application code for the word processing, electronic messaging and file cabinet menus is really pretty simple, thanks to a family of ALL-IN-1 functions, known as the *file cabinet sub-functions*. These functions do things such as create, edit, delete, refile, cross-file, duplicate, send and select documents and mail messages. Now, why would I be telling you about that? Simply this.

The very same set of functions that we used to create the word processing and electronic messaging systems in ALL-IN-1, are available to you for use in your applications. For example, you may want to give the user a document, associated with a particular record in a data file. Digital did this in the ALL-IN-1 System for Sales and Marketing by using these file cabinet functions to create the document, store its name in the appropriate data record, and then let the user edit it with the word processing editor.

In addition to *file cabinet* functions, you also have access to a full set of *mailing* functions and a full set of *calendar* and *time management* functions. So if you have an application that is used to track sales calls, you can have it post follow up actions on the sales reps' calendars, reminding them to take care of their customers on the appropriate days.

Let's take this concept one step further. You may find that some of the scripts we have written for say, creating a document, do much the same task you need to do. Insofar as is possible, we have made these scripts that drive the standard equipment ALL-IN-1 applications, callable from

your applications. This means you may be able to use some of the work that has already been put into common application needs, further reducing your investment in application development.

Do We Still Need Programmers?

That heading was just to get your attention. Surely we do still need application programmers, or developers, as we like to say, but their jobs are changing somewhat. As you can see, even if you did not follow all of the details in this article, the job of the ALL-IN-1 application developer is less of a coder, and more of a system designer and integrator. ALL-IN-1 applications are a collection of various high-level, powerful pieces, and the real task in implementing an ALL-IN-1 application is deciding which pieces to use and how to connect them. The "coding" itself is minimal.

Because of this modular approach to applications, I believe we will start to see ALL-IN-1 application libraries, supplied by vendors or created internally, containing various applications or application pieces. Items from this library could be taken and modified to suit your specific purpose, thereby even further reducing your development time.

Though I have attempted to be fairly complete, I have only touched the surface of the totality of what is available to the application developer through ALL-IN-1. For more detailed descriptions of the many features and functions of ALL-IN-1, you may wish to consult the ALL-IN-1 Application Programmer's Reference, a three volume, 1000 plus page reference work available from Digital.

Wrapping It Up

If you are a non-technical or semi-technical data processing person, and have stuck with me this far, I want to thank you. Though you may not have understood every detail, I hope you have gained an appreciation for this new approach to data processing applications. As Tom Peters has observed, the *DO IT, FIX IT, TRY IT* approach to business is *in* and the *ANALYZE IT, COMPLICATE IT, DEBATE IT* approach is on the *way out*. If we can quickly prototype applications and test their usefulness in real situations, we can not only save money in the development process, but more importantly, we can be assured of producing applications and putting them in the hands of users *before* they (the applications) become obsolete.

User Communications for Office Automation Systems

Peter LaQuerre
Digital Equipment Corporation
Maynard, Massachusetts

Abstract

This paper discusses major issues facing user communications in the office marketplace. Traditional documentation sets are no longer sufficient to meet the needs of this market. In particular, this paper defines an overriding issue in DIGITAL's office automation environment: how we provide user communications for complete office automation systems.

Introduction

Office Systems Documentation (OSD) is a group of writers, editors, and publications personnel responsible for providing documentation for several of DIGITAL's office automation products. Our goal is to provide accurate, useful, and timely documentation packages for Business and Office Systems Engineering (BOSE) systems.

In the past few years, our responsibilities have changed from the task of documenting individual products to documenting products that can work together as complete office automation systems. Each of the subjects discussed in this paper is a direct result of this one overriding issue.

To provide a user communications package for a system, we must:

- Understand and define the term *system*
- Produce core documentation
- Carefully design complete user interface packages
- Provide more on-line assistance
- Build systems that are *user-friendly*

This paper will discuss each of these concepts and how Office Systems Documentation is implementing them.

What Is a System?

Every science seems to have its own specific definition of the word *system*. It makes sense then that the computer industry would have a few of its own definitions and that DIGITAL would have several.

If we add a qualifier to *system*, we can clarify what kind of system we are discussing. For computer users in general, a *computer system* can be many things. Most users would consider a computer system as a mainframe or minicomputer. This system is stored in an air-conditioned computer lab and is blamed for any problems you might have in the office.

For our purposes, the qualifier is *office automation*, as in a complete *office automation system*. In this category would be ALL-IN-1 in its many forms, and the WPS-PLUS/DECpage combinations, when they are integrated in a single VAX installation.

These products are systems because they are an "assemblage or combination of" two or more software tools, all accessible from a common interface. In other words, you can access more than one type of office automation software from the same menu system and from the same documentation set.

For an office automation system to be successful, the user interface must be consistent throughout the system. That way, users can learn how to use the *system* and not each individual component.

WPS-PLUS as a System

In the past, we documented *standalone* products. Standalone products are components not attached to a system. As you can imagine, it is easier to write documentation for a standalone product than it is to write documentation for a system.

For example, when we wrote the documentation for WPS-PLUS/VMS V1.1, we were documenting a standalone product that had no connections to other software products. We didn't have to contact other writing groups. We didn't have to imagine how our documentation would match other books in a system documentation set. We could concentrate on the job of documenting the product.

WPS-PLUS/VMS V2.0 introduced us to an office automation system. We now had to keep in mind that users would be installing DECpage as an integrated part of WPS-PLUS.

To make WPS-PLUS and DECpage work together, the engineers and planners designed DECpage as a menu option. To make the documentation more consistent and easy to understand, we worked to make the writing styles and packaging of the DECpage manuals consistent with WPS-PLUS.

The days of documenting a single, standalone word processor for the VAX/VMS environment were over.

WPS-PLUS as a System Component

Besides being a system in itself, WPS-PLUS is also considered a component—the document processing component—of other systems.

For example, WPS-PLUS/ALL-IN-1 is the document processing component of ALL-IN-1. In this capacity, WPS-PLUS is a part of the ALL-IN-1 system and the documentation, product interface, on-line Help, and on-line training must be integrated to make the package feel like a system.

To make this possible, we had to use several of the tools and concepts described in this paper. Namely, we had to use the concepts of core documentation and produce a consistent, useful on-line Help system.

Producing Core Documentation

Currently, WPS-PLUS runs on several versions of VMS, ALL-IN-1, Rainbow/MS-DOS, P/OS, and IBM's DOS. Because WPS-PLUS is available in so many configurations, there is interest in making the software and the documentation:

- Cost-effective
- Easier and faster to produce
- Fully international and easy to translate
- Easily integrated into software systems

However, none of these goals were seen to be as important as presenting to the user a single and consistent approach to learning WPS-PLUS regardless of the implementation.

The first step toward a solution involved two realizations:

- We realized that WPS-PLUS is basically the same wherever it operates. We call this *sameness* the *core* of WPS-PLUS.
- We realized that the differences between environments and versions of WPS-PLUS appear, largely, in isolated areas of functionality.

Therefore, we knew that we would have to devise a documentation strategy containing a core that was largely environment and version independent.

The solution that developed represents a highly modular approach to documentation. It attempts to identify those aspects of functionality that remain the same across all WPS-PLUS systems. As this functionality is the *core* of WPS-PLUS, it forms the basis of what we call the *core* concept in WPS-PLUS documentation.

Core Software = Functionality that is always present, regardless of the version of the software or the operating environment

Core Documentation = Documentation of that functionality

Non-core Software = Functionality that varies across operating systems

Non-core Documentation = Documentation of those variations

This process creates general purpose, reusable modules of text and ensures consistent presentation of the software across multiple systems. It gives us greater flexibility in building systems of communications products and makes it easier for customers who are translating systems or putting together systems from DIGITAL and third-party components. This topic of "open" or "loosely-coupled" systems is very closely related to other important ("hot") issues for both software and documentation: consistency, customization, and integrated user communications.

Core modules may vary in size from individual phrases and sentences to entire books. An example of a book that is more than 90% core is *WPS-PLUS List and Sort Processing*. At the other extreme, primers, tutorials, and installation guides (because they are system-dependent) are likely to have less core material. The simplest illustration of this concept may be seen in the WPS-PLUS product names: WPS-PLUS/VMS, WPS-PLUS/ALL-IN-1, WPS-PLUS/Rainbow, and so on. The core name is the same in all three cases. Only the appendage that describes the implementation environment changes. You might think of /VMS, /ALL-IN-1, and /Rainbow (or any of the other PC-based products) as non-core elements in this case.

The issue becomes more complex when dealing with functionality differences. For example, *WPS-PLUS Editor Functions* documents the WPS-PLUS editor and every version contains the same information — except for specific references to VMS, ALL-IN-1, or PC-based features.

More specifically, *WPS-PLUS Rainbow Editor Functions* documents WPS-PLUS on a personal computer with floppy disk drives, and the user must be aware of the physical location of documents. To select a document, WPS-PLUS/Rainbow users must enter B: before the document title (if the document is stored on Drive B:). The B: preface is called the path name of the document.

As a result, *WPS-PLUS Rainbow Editor Functions* reminds users to include the path name when they select a Library or Abbreviation document. This type of difference appears only in *WPS-PLUS/PC-based Editor Functions* and not in the VMS or ALL-IN-1 versions of this manual.

The following equations may also help make these concepts more concrete:

The core approach to documentation is ideal for systems because it:

- Gives consistency in organization and format to users migrating to and from different operating systems or implementations
- Factors out generic material to create general purpose, reusable modules of information
- Presents to the user a single and consistent approach to learning the product regardless of the implementation
- Eliminates needless repetition in instructional material
- Captures and preserves well-designed, written, and tested modules of information

Since core documentation clearly benefits everyone, it is important to continue to produce and extend this movement towards user communications packages that are increasingly modular and generic (core). This approach gives users greater consistency in the same way that software achieves consistency from modular and generic code. We should be able to produce core documentation as long as engineering produces "core" products.

Designing a Complete User Interface Package

What Is User Interface?

At a similar DECUS session last year, my supervisor, Sue Franklin, defined user interface. She explained that user interface is "often described as the point where man and machine meet." It is "the medium through which the user receives information about the software's functionality."

To engineers who build Office Automation products, the user interface is one of many challenges in a complex problem. But to the user, the interface IS the product.

We usually define the user interface as having four parts:

- Documentation
- On-Line Help
- CBIs
- The Product Interface

Each of these parts is not effective by itself, but when they work together, they form a complete and useful package for the user.

Traditionally, we in documentation have been involved in only one or two of the four user interface ingredients. First, we have always been the people who write the manuals. Later, as on-line Help systems became more extensive, we took on the role of writing the text for the WPS-PLUS and ALL-IN-1 on-line Help.

Still later, we have become involved with the CBIs for the products we document. By controlling these three parts of the user interface, we can create consistent, integrated, user communication packages for our products.

Like on-line Help, the product interface has long been considered the responsibility of the engineers. The program developers write the code and user interface engineers help them design the product interface.

That system worked fine in the past, but now our Office Automation products are more powerful and complex. We are documenting systems instead of individual products. The user interface engineers needed some help from the different groups who help build the system. Developers, computer architects, and writers.

Why Writers and Editors?

Recently, documentation groups have been asked to help design the entire user interface package for some of our future office automation products.

The logic is three-fold:

- 1 Writers and editors are familiar with the needs of users who must write memos and reports.
- 2 Writers and editors tend to recognize global issues that affect more than just one product.
- 3 Writers and editors tend to picture how the product will be presented to the user.

Besides these strategic reasons, designing user interface also accomplishes a more selfish goal for the writers and editors. By helping design the product interface, we can better plan our writing and editing resources and isolate problem areas well before the writing effort begins.

Understanding the Needs of Office Automation Users

Products such as ALL-IN-1 and WPS-PLUS are built for users who work in offices. Office workers rely on memos, reports, and papers that must be distributed to other workers.

Writers and editors in BOSE also write memos, books, and papers and distribute them to other workers. That's our job. Therefore, we understand the needs of people who use word processors.

As much as possible, the writers and editors who work on the WPS-PLUS documentation use the product themselves. We create memos, use DIGITAL tools to create high quality output, and distribute information via electronic mail, or allow reviewers to copy the information from library directories.

In the past, our role as user interface designers was informal. When one of us began documenting a certain feature of WPS-PLUS, we would notice something in the software that didn't seem right. After questioning the developer and talking with other planners, our suggestions were sometimes incorporated into the product.

Now, we are formalizing this role. As we continue planning for future versions of WPS-PLUS, DECpage, and other OA tools, writers and editors are being included in the planning for these products. We are involved in detailed discussions about functionality, usability, and screen design.

We realize we are not expert user interface designers. We rely on our gut instincts and our own experiences to make design decisions. But together with the user interface engineers, who are trained in designing the user interface, we can work to create productive and easy-to-use products.

Recognizing Global Issues

Last year at DECUS, Sue Franklin described the core documentation concept and how we designed our documentation so we can use large sections of it for more than one version of WPS-PLUS, whether it runs on VMS, ALL-IN-1, or a PC.

One of the goals of the core concept is to make the user see the consistency and usefulness of a product that is available on many different machines. Our goal is to make the documentation reflect that consistency and make it easier for users to migrate from one WPS-PLUS environment to another.

But before the documentation could accomplish that goal, the software had to accomplish that goal. Many of the informal suggestions made by writers to the engineering and planning groups had to do with inconsistency between the WPS-PLUS products. The writers and editors, who had to describe each of the products in a consistent and concise way, noticed the global implications of some engineering decisions.

Now, through more formal channels, the writers and editors are having their say in the entire user interface design.

Picturing the User Interface

When you tell writers or editors how a future product might function, they usually react by picturing how the software might be described to the user.

Many of us believe this is a good test for software ideas. If a product's interface is hard to explain in the documentation, there is a good chance it needs redesigning. The ideal software should not require a huge documentation set. Instead, a well-designed interface should be able to handle most of the problems a user runs into and the documentation, on-line Help, and CBIs should supplement the screen interface.

This perspective—the writer's point of view—can sometimes help engineers and planners recognize problem areas of the UI design.

What Is On-Line Assistance?

The term *on-line assistance* has no formal definition. Instead, it is the general term for a group of tools we use to display user information on the computer screen. The idea is to minimize the need for bulky hardcopy documentation and to reduce the time it takes to locate specific information.

The End of the Software Reference Manual?

When we begin to discuss on-line assistance, someone always asks whether on-line information will replace our need for hardcopy manuals. We imagine the clean, paperless office of the future—no software manuals on the shelves or strewn about the desk, only the computer terminal resting on an uncluttered desk next to a cordless telephone.

I've heard this question asked many times at computer conferences and even at recent meetings here at DIGITAL. The answer is usually as follows:

While on-line assistance is very useful for many specific purposes, there will always be a need for certain types of hardcopy information.

Four Types of On-Line Assistance

We can divide the current on-line assistance packages into four categories:

- On-line Help
- CBIs and tutorials
- On-line documentation
- System messages

On-Line Help

The most important type of on-line assistance is on-line Help. More than any other type of on-line assistance, Help is becoming a necessary part of office automation systems.

A successful on-line Help system can:

- Do its job better than a hardcopy manual
- Supply context-sensitive, specific information
- Save time for the user

Just as hardcopy manuals are well-suited for certain types of information, on-line Help is well-suited for brief, concise instructions explaining how to get a job done.

For example, imagine you are selecting a WPS-PLUS document. As you begin to type the title in the title field, you realize you have forgotten the title of the document. You wonder if there is a way to see an index of the documents in your file cabinet folder without leaving the WPS-PLUS form.

In the days before on-line Help, you would have to reach for a reference manual or quick lookup guide. Either of these manuals contain the information you need, but the process of locating the manual, finding the information, and using the information is time-consuming.

With on-line Help, you can find the specific information you need by pressing a keyboard function. A window appears at the top of the screen and as you scroll through it, you can read specific information about selecting document titles.

To design a Help system, we must keep several things in mind. To be effective and useful, a Help system must be:

- Written in a short and concise manner
- Designed so it is context-specific; it must supply the user with information about the current topic

When a WPS-PLUS or ALL-IN-1 user presses the Help key, the information must be short and to the point. It should not give background information or extensive information about other functions the user is not using. It should tell the user how to get the immediate job done.

Since we have to fit the information into a relatively small area of the screen, we avoid large paragraphs of information and instead rely on information that pertains only to the immediate task.

All Help messages should contain references to other Help topics or to the hardcopy documentation. That way, if the Help message is not sufficient to get the job done, the user can go directly to where the Help is available.

The references to documentation link the on-line information with the hardcopy text, creating a more integrated user communication package.

Computer Based Instructions (CBIs)

Computer Based Instruction is another specialized type of on-line assistance. Like on-line Help, it has a specific role to play in providing information to the user.

While on-line Help supplies the user with direct and immediate information about a very specific topic, CBIs and tutorials supply the new user with an introduction to the system, as well as lessons on specific topics.

Some characteristics of CBIs and tutorials:

- They are self-paced
- They are always available
- They fit well into the user communications package

Like the Getting Started manual, CBIs are there whenever you need them. Depending on your previous experience with computers and office automation equipment, you can take all or some of the lessons. As you become more experienced with the product, you can take specific CBI lessons as you need them.

The CBIs also offer an alternative to the hardcopy Getting Started manual. Some users feel comfortable with an on-line set of lessons and may learn faster using the CBIs rather than lessons in a Getting Started manual.

Once again, this helps us form a complete user communications package.

On-Line Documentation

There are several types of on-line documentation. For WPS-PLUS and ALL-IN-1 we are familiar with three specific types:

- Documentation stored on-line and delivered with the product
- Customizable documentation
- Documentation available through an on-line delivery package

Documentation Stored On-Line

New technology will soon make it possible to deliver entire documentation sets on-line. When the product is delivered, the customer prints out the documentation and distributes it to the users.

This method of on-line documentation:

- Saves typesetting and printing costs
- Allows customers to print only the documentation they need

Currently, we use this method for certain types of documentation.

For example, WPS-PLUS/VMS V2.0 was shipped with a feature called XAL (External Application Link). This feature allows technically-oriented programmers and consultants to run VMS applications from within WPS-PLUS. However, to set up this XAL feature, you must be familiar with VMS command languages and some programming skills.

Instead of including this specialized, technical document in the user documentation, we used WPS-PLUS to create the document and stored it on-line for users and companies who are interested. By contacting the software specialist representative, the company can learn how XAL works and print the documentation from a special directory inside WPS-PLUS.

Customizable Documentation

Customizable documentation is documentation stored on-line with instructions on how to change the content and organization of the documents.

Large companies who use ALL-IN-1 for specific purposes appreciate this capability. They can change the documentation to match their specific office tasks and train their employees in a consistent, personalized manner.

Refer to Tom Skelcher's DECUS presentation on customizing ALL-IN-1 documentation.

Documentation Accessed On-Line

Some types of software provide on-line access to entire documentation sets. Like the Help systems and CBIs, these systems are effective for certain types of information in certain user environments.

The concept is essentially the same used in the design of the WPS-PLUS and ALL-IN-1 Help systems. By pressing certain keystrokes, you "turn the pages" of the on-line manual to display the information you want. DIGITAL's VTX Electronic Publishing system is an example of a product designed to display large amounts of information previously produced in hardcopy format.

On-line documentation packages are well-suited for information about operating systems and programming tools because technically-oriented users tend to appreciate on-line facilities since they are experienced computer users.

On the other hand, on-line documentation systems push delivery systems to their limits. Displaying large amounts of information on the screen is a challenge that requires much design work and user interface training. Otherwise, the time it takes to find the information on-line will equal the time it would take to read it in a hardcopy manual.

For office automation systems, speed and ease-of-use for non-computer specialists is very important. Therefore, on-line Help systems and CBIs that supply concise, direct information about getting the job done are more important than supplying large amounts of on-line documentation.

System Messages

System messages are an important and often overlooked part of the user information package. Without these messages, users wouldn't understand the errors they make. Because they supply the user with important information, they must be considered a part of the user information package.

Previously, the engineers were solely responsible for error messages. They understand the coding and routines that call the error messages so they maintained control of that part of the user interface.

This will be true in the future as well, but new on-line assistance programs may offer entry into the Help system if you don't understand the system messages you receive.

In the past, documentation has tried to include a list of possible error messages in the hardcopy documentation. But an on-line Help system, hooked directly into the error message would be much more effective. In OSD, we are looking into that necessary and useful connection.

Summary of On-Line Assistance

Each type of on-line assistance has certain advantages in providing certain types of information. The trick is to decide where each type is useful and how they can work together to form a complete user information package.

The following chart is an example of how I might categorize the usefulness of each type of on-line assistance. Often, it is useful to imagine how different types of users might use on-line information:

	New Users	Office Users	Computer Specialists
Use on-line Help?	Yes	Yes	Yes
Use CBIs?	Yes	Yes	No
Use on-line documentation?	No	No	Yes
Use system messages?	Yes	Yes	Yes

This is only an example of the questions we need to ask ourselves as we examine new and existing on-line information technologies.

The User Friendly Puzzle

The strategic goal of DIGITAL'S Business and Office Systems Engineering group (BOSE) is "end user information productivity." OSD is responsible for providing the user communications segment of that goal — greater and faster user productivity.

We have identified and discussed several communications issues in support of that goal:

- The systems approach to documentation
- Core documentation
- The definition and role of the user interface
- The definition and role of on-line assistance

What has been the point of all this identification and discussion? What puzzle are we trying to solve? We are trying to solve the puzzle of communication. We are trying to transfer a body of knowledge about a product from our heads to the user's head through an effort called, perhaps loosely, user communications. We are trying to discover how best to transfer this knowledge.

The best way of transmitting knowledge has been debated for centuries. Today, the best ways are called "user friendly." This attribute is also known as "ease-of-use" and "easy-to-use."

I suspect that whether or not a system and its accompanying user communications package is user friendly is, like beauty, in the eye of the beholder.

I suspect that if users have questions and find the answers easily, they say that the system is user friendly. And, if not, they believe the system is not adequately meeting their needs, are slower to learn the system, and are slower to use the system to its fullest functionality. This is not by way of getting us off the hook. We have made progress. A sign of our progress may be that our foreign language translators find our documentation and on-line Help "too friendly" and are insulted by our air of informality.

User communications packages can never make a complex system look simple. ALL-IN-1, WPS-PLUS, and DECpage are complex systems. With that complexity, the user gets greater power and functionality. That complexity also demands that the user expend more effort during the learning process.

This complexity also means that we must build ease of use into systems at the design stage. User friendliness is not a concept that can be layered on a product like a final coat of varnish. This means that we must work to design systems that are:

- Simple (not simplistic), intuitive, and logical
- Consistent and, therefore, predictable
- Customizable and open

The messages for us are clear. We have learned much about building user communications packages for office automation products and I believe that we continue to move in the right direction. Learning to document systems, not individual products, defining and clarifying the role of the user interface, orienting toward user tasks, learning to define and build integrated user communications, and struggling to solve the "user friendly" puzzle all support our common goal of increased user productivity.

PERSONAL COMPUTER SIG

PUTTING THE READER BACK IN MANUALS;
COMPUTER MANUALS AND THE
PROBLEMS OF READABILITY--V 2.0

By

Thomas L. Warren
Department of English
Oklahoma State University
Stillwater, OK 74078-0135

ABSTRACT

At the Fall, 1985 DECUS Symposium, I presented a paper in which I analyzed samples from computer manuals based on reader access. Following a review of how humans read and understand, I summarized the results of applying readability, stylistic, and design tests to the samples. I concluded that the manual excerpt that ranked high in style and readability rated low in design, and the excerpt that rated high in design rated low in style and readability. The full paper appears in the Proceedings of the Digital Equipment Corporation users Society, Fall, 1985 Symposium. The present paper uses the same tests but a different sample plus applying one more stylistic test. I again conclude that writers and designers do not use all the tools available to them to put readers back into manuals.

This paper extends one that I did for DECUS at the Fall, 1985 Symposium (6). In that paper, I focused on instructions--prose that the writer meant for the reader to use while performing some sort of action. I looked at the page layout and style--especially the command forms of the verbs--because instructions are nonfiction prose where the reader normally does not read linearly, that is, beginning at the first word on the page and progressing to the last. We read novels, poems, short stories, newspaper features, etc. in this manner. Readers normally read instructions in a random fashion because they are interested in performing a particular action and need help doing it.

I used a single command ("Move a Block of Text") from nine wordprocessing programs and analyzed them for layout and style. I can give you a flavor of that presentation by looking at some samples. Given a situation where "Move a Block of Text" is not a basic wordprocessing command that the reader must learn (such as "Insert Text," "Delete Text," "Setting Format," and so forth) the reader probably will turn to it on those occasions when moving the block is important. With that in mind, let's look at Figure 1.

INSERT FIGURE 1 HERE

This text is relatively easy to get into. The writer and designer have used

plenty of white space so that there is not a lot of text on the page to confuse the reader. When readers use instructions, they read, turn to perform, and then return to the text to pick-up where they left off before moving to the next step. Some readers will hold a block of text in memory as they perform the step, returning for reminders or when the action is complete.

We can see also that the writer and designer have used boldface type and have numbered the steps, both practices that help the eye quickly return to the text and the next action.

Finally, a section on "Helpful Information"--but without numbers so that the reader knows this list provides information and not more steps to follow.

Compare this text with the next, Figure 2 How easily can the reader get into and out of this text?

INSERT FIGURE 2 HERE

This text features long lines (from margin to margin meaning very little white space), with the verbs buried in the sentences. Readers must read the entire section before performing the actions, and then must perform from memory. It is certainly a hard text for the reader to read, perform, and return to the place left.

The results of the analysis were that Figures 3 and 4 tested to be the best--

ironically, Figure 3 text tested best for layout but not for style and Figure 4 for style but not for layout.

INSERT FIGURES 3 AND 4 HERE

Before I discuss the new text, I need to review the tests from that previous paper (layout and style). For the layout, I used such matters as line length, justified right margins, justified bottom margins (i.e. a fixed number of lines per page), type size, spacing and other white space matters, headings and subheadings, etc. For the stylistic analysis, I used computer programs: "Grammatik," "Comment," and "Readability Calculations." (1, 2). My conclusions were that the texts needed to do well in both areas in order to keep the readers involved.

In the present study, I looked at what I took to be a non-instructional text--one that ostensibly was communicating information. I wanted to learn what differences writers and designers might use to convey information rather than instructions. Figure 5 shows that text.

INSERT FIGURE 5 HERE

We can notice some features of this text without reading it. First, it runs from margin to margin on the page, with lines that are about 5 inches long. Second, we see a visual--a copy of a screen showing elements that this section of the manual addresses. Third, there is a "Note," indented from both the right and left margins. Fourth, there are at least two levels of headings with three subheadings being questions.

Having determined that, I looked more closely at the text and how it communicated with the reader. My first conclusion (an obvious one) was that the text was meant to be read in a linear fashion. The reader was to start at the top of the left-hand page and read across the line, then come back to the left margin and begin again. The reader was to process the text in this fashion, anticipating a subject of the sentence followed by the verb.

This conclusion suggested that the writer assumed the reader would process the text in the traditional way. That is, the reader would first sense the image on the page as input to the mental system. This input would then be filtered (because the mind cannot really process all the bits of information that the eye is capable of sending through). Once past this physiological filter, the mind searches for patterns; the reader, drawing on cultural conditioning, begins to extract meaning from the arrangement of the words. As an example, consider these two sentences:

[1] John hit the ball.

[2] The ball hit John.

Sentence [1] conveys a different meaning than sentence [2]--and the reader recognizes that it does because of the positioning of the words. The reader will also look upon these next two sentences as also conveying different meanings:

[3] Strike the key to start the process.

[4] Striking the key will begin the process.

Clearly, two different purposes are at work here. In [3], the writer expects the reader to perform an action, while in [4], the writer wants the reader to be informed without necessarily performing an action.

Thus, at this stage of processing, the reader looks at our sample text and assume the writer wants to convey information without requiring a physical response. At least that is the impression the reader will get from the first paragraph. A different impression awaits the reader in the "Note" because the writer wants the reader to perform an action ("... press the Tab key"). This intent carries on into the next section where the heading asks a question and the first sentence following leads the reader to an action. Notice that the reader is commanded to "Press . . .," and then told "if you want to change the format" (my emphasis). If the reader anticipates command-response, a situation normal in manuals, he or she will press the key before coming to the conditional. Having such a mindset suggests the next step in processing information--selection.

When we approach any piece of writing (or communication for that matter), we expect something. Perhaps we expect information or entertainment or surprise or being told to do something. Readers normally come to manuals because they need information, especially how to do something, and so expect to be told something. When they encounter a sentence that opens with a command ("Press the Y key," for example), the response is to perform. When the action depends on other factors and those factors follow the command verb, readers will respond to the verb and then analyze the conditional. Normally, the reverse should be the order.

With this brief summary as background, we can now turn to a closer analysis of the new sample. How is the reader to respond to this first section? I have already noted the long lines (some 5 inches). Most researchers agree that line length does play a major role in the reader comprehending the text. The studies suggest something closer to 3 inches for a comfortable line length, because the eye must move more when the line is long, and the more the eye works physically, the more tired it becomes, especially if the reader is constantly shifting from text to keyboard and back again. The eye, in these cases, must relocate its original leaving point in order to resume. If there is sufficient white space, numbering, and other designer aids, the eye can return to the spot and not become confused.

Another interesting point about this text is that page 1-23 has justified right-hand margins, and 1-24 does not (note how the right-hand margin on 1-23 is regular

and irregular on 1-24). The eye could easily become confused by this. Studies show that the unjustified right-hand margin presents a more psychologically appealing line because the reader gains a greater sense of making progress when the lines are not justified. The reader also does not have to remember a part of a hyphenated word.

The reader, as I mentioned above, responds to the first line following the first question (a command) only to find a conditional. Two of the four headings (three are questions) open with the command "Press . . . ," while the other two use the conditional "If you" The reader is apt to become confused.

Past these factors, I analyzed the text sample using the same tests as I used before ("Readability Calculations," "Grammatik," and "Comment"[4, 1, 2]). Table 1 shows the results when running the text through the "Readability Calculations" computer analysis.

INSERT TABLE 1 HERE

Figure 6 shows a composite bar graph of the grade level scores the sample achieved on the nine readability formula calculations.

INSERT FIGURE 6 HERE

Figure 7 shows a Fry graph from the readability calculations.

INSERT FIGURE 7 HERE

These results show, as they did when I reported them on the other texts, that readability formulas may have reliability when compared with themselves, but are not very reliable when compared with other readability formulas. Regardless of that, there are some interesting points to make about the readability results. The formulas agree that anyone with an eighth-grade education could easily understand the material. One formula (Dale-Chall) even suggests that the reading level is fourth grade and below. Because that formula uses vocabulary and the other eight formulas rely on some kind of count (usually syllables and sentence length), the vocabulary must be relatively simple. In point of fact, it contains a number of words that could cause confusion (the clearest example, I believe, is the word "terminate," used in a question/heading and discussion).

I was interested in this vocabulary question and printed out the words that Dale-Chall identified as not being on the basic vocabulary lists (a list, I might add, that has not been updated since the 1940s). Table 2 lists those words and indicates the number of times each occurs. Note that of the 47 words listed, only 4 were defined in either the Getting Started or Reference Manual sections of the documentation.

INSERT TABLE 2 HERE

A scan of this list shows that these words are rather common words, so the reader should not have trouble understanding most of them. There are, of course, exceptions--"terminate" being the most notable. But were they words that the assumed reader of the document really would

know? To answer that, I looked at the assumed reader of the materials. Who were the people the writers and designers had decided were the readers? I found two answers: one in Getting Started and one in the Reference Manual sections:

Intended Reader

This guide is intended for the first-time user of the Samna Word III word processing package. The purpose of this guide is to show you how to start Word III, and how to write and edit documents you use every day, such as reports and memos. This guide also tells you how to use the Samna Word III tutorial program. (Getting Started, p. viii)

Intended Reader

The purpose of this guide is to present an overview of the product and to describe each procedure and key you can use with Samna Word III. (Reference Manual, p. ix)

Not all that helpful, but the Getting Started statement at least identifies the reader as a "first-time user," so that few assumptions are likely about vocabulary. The problem, however, is that the Reference Manual section makes a different assumption and puts that assumption into practice by limiting definitions.

I also wanted to run the other computer tests ("Grammatik"[1] and "Comment"[2]) that I previously. I have summarized the results of all the tests and placed them in the Appendix. What these new tests showed was a longer average word per sentence (expected because I assumed information text and not instruction), a greater number of forms of "to be" (again because of the nature of the text), and considerably lower transitions (surprising because I assumed this text to be coherent).

One final (and new) test I ran was one called "PC-Style"(3). Several categories agreed with the previous tests, and those that disagreed do not suggest strong conflicts. (See Figure 8)

INSERT FIGURE 8 HERE

CONCLUSION

As before, I have been concerned with the way the writer and the designer approached the reader in the manual. I looked at both the design of the page in the manual and the style the writer used to convey the information. I am left, again, with two conclusions:

1. Writers and designers do not take full advantage of the ways they can accommodate the reader.
2. When readers must work at getting the information they need, they tend to avoid extracting it.

I am concerned that the writers and designers do not use style and design to encourage the reader's attitude that the text is accessible--and hence acceptable. All too often the reader comes away from the text with a feeling of bitterness or anger that the information needed was not readily available.

Is it any wonder that readers consider the manual the last resort when the information is not accessible. When I see pages like the ones I have shown here, I wonder if the people responsible for budgets rather than writers and designers are in charge of the documentation. Certainly, documentation is not a high profit item, yet how much good will is lost because the documentation was an after-thought, or was put together as the product was moving to the shipping dock? When budget matters take precedence over people matters in documentation, it is little wonder that the manual is the last thing read.

REFERENCES CITED

1. Aspen Software Company. "Grammatik" (Includes "Random House Proofreader," Version 1.15 [1982]), Version 1.84. Aspen, CO: Aspen Software Company, 1981.
2. Barker, Thomas T. "Comment." Lubbock, TX: Texas Tech Microcomputer Company, 1984.
3. Button, Jim. "PC-Style." Bellevue, WA: Buttonware, 1986.
4. Micro Power & Light Company. "Readability Calculations: According to Nine Formulas." Dallas, TX: Micro power & Light Company, 1984.
5. Samna WordTM III: Documentation. Marlborough, MA: Digital Equipment Corporation, 1984.
6. Warren, Thomas L. "Putting the Reader Back in Manuals: Computer Manuals and the Problems of Readability." DECUS Fall Symposium, Anaheim, California. Marlboro, MA: Digital Equipment Corporation Users Society, 1986, pp. 457-482.

Move

Move

MOVE

Purpose

You use the MOVE command to move a specific block of text from one place to another. MOVE lets you delete the original block of text and insert it anywhere else you want.

Moving a Block of Text

1. Position the cursor at the beginning of the text you want to move.
2. Press the Do key.
3. Press the M key (for Move).
4. Shade the text you want to move.
5. Press the Return key.
6. Position the cursor where you want to insert the text.
7. Hold down the Ctrl key while you press the Insert Here key.
8. Press the Return key.
9. Press the N key (for No) if you want SAMNA to insert the text with its stored format. Press the Y key (for Yes) if you want SAMNA to insert the text with the current format.
10. Press the Return key.

Figure 1. Sample page from Samna Word II manual.

Helpful Information

- The text you move is deleted from its original location. If you want to move a block of text and also retain it in the original file, use the COPY command.
- You can move any amount of text. You shade the text you want to move using the Word, Sentence, Line, Paragraph, Page, File, and arrow keys. However, the File key is not recommended.
- Any marks within the text are moved with the text.
- The text you move is temporarily saved, along with its format, in the TEMP file.
- The TEMP file holds one block of text at a time. Therefore, when you move (or copy) text, SAMNA replaces the contents of the TEMP file.
- If you want to save text stored in the TEMP file after you finish moving or copying, give the TEMP file another name. The text is now safely in the file with the new name.
- You can display and edit the TEMP file.

Moving a Block

The block move command (^KV) **moves** all the text in the marked block to the **cursor position**, deleting the original at its old position. If no block is marked when the command is given, or if either marker is hidden, an error message occurs (Appendix B).

The destination may be in the middle of a line, if desired — for example when rearranging sentences in a paragraph. Just put the cursor where you want the block moved to. The cursor is left at the beginning of the moved text.

The beginning and end markers **move with the block** and remain displayed. After inspecting the result, type ^KH to hide the block markers -- both to remove the distraction from the screen, and to protect against block commands typed by accident. If you wish to use the same block markers later, just type ^KH again.

The block move command moves **exactly** the characters you have marked, and does no automatic reformatting. Thus, text reformatting is often required after a move. After rearranging sentences, for example, use paragraph reform (^B, Section 4) to re-establish the margins. You may also notice that you included too many or too few spaces or carriage returns at the beginning or end of the block. These errors are easily corrected with a few regular editing commands.

After a block move, the command ^QV will move the cursor to the place the block came from. It's a good idea to inspect here after moving, as you may have left too many spaces or carriage returns behind, or you may need to reform the paragraph. Note that any place markers 0-9 in the marked block do **not** move with it—they remain at the place the block came from.

For an example of moving a column block, see Figure 6-1.

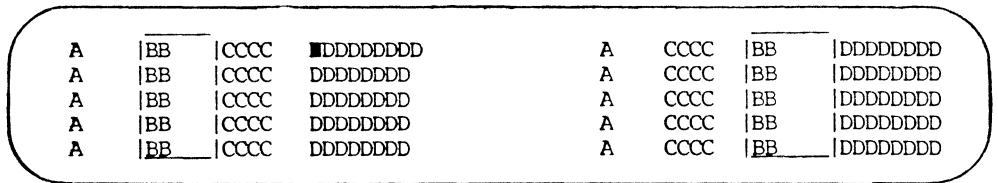


Figure 6-1. Moving a Column Block

Figure 2. Sample page from Wordstar manual

Copying or Moving Text

You use the copy or move procedure to save a specific block or column of text and insert it in a different location in the file. This is useful when you need to repeat the same information or to copy Format Lines. You can copy text to a temporary buffer or to a stored file.

Copying or Moving a Block of Text

To copy or move an entire block of text:

1. Position the cursor at the beginning of the text you want to copy.
2. Press the Do key.
3. Type the letter C (for copy) or M (for move).

When you move text, it is deleted from its original location. When you copy text, it is not deleted from the original location, and therefore exists twice in your file.

If you are storing the text in a separate file:

- Press the File key.
- If you do not want to use the default file name TEMP, type a file name.
- Press the Return key.

NOTE

If you do not specify a file to store the text in when you copy it, Samna stores it in a temporary buffer. This buffer can hold only about one full page of text. Therefore, if you have a large block of text to copy, you should store it in a separate file.

4. Shade the text you want to copy. Figure 3. Sample
pages from Samna
Word III manual
5. Press the Return key.
6. Move the cursor to where you want to insert the copy.
7. Hold down the Ctrl key while you press the Insert Here key. If you stored the text in a file:

- Press the File key.
- Type the name of the file, unless you used the default file TEMP.
- Press the Return key.

Samna asks:

```
Which format should be used?  
Type Y to use the current format. Type N to insert the  
stored format.  
Will the text be inserted into the current (displayed)  
format? Yes/No (N)  
Is the text you are inserting a column? Yes/No (N)
```

8. Respond to these questions and press the Return key.

NOTE

To move text with its original format, you must save the text in a separate file.

6.4.2 THE CUT COMMAND

This feature allows you to remove any amount of text from a document. Using the Paste command described later in this chapter, you may then move the cut text to another position within the same document or to another document within the same Document Directory. You may also elect to do nothing with the text you have cut, thus deleting it from your document. The Paste command in this case allows you to recover the last block of text you deleted in this manner.

6.4.2.1 Basic Concepts

Think of the Cut function as performing the same operation you would perform with a knife on a paper document. At the point where you would begin your cut in the paper document, you place the Select Marker in MASS-11. Moving the cursor in MASS-11 is similar to running your knife around the text you want to remove, working towards the end of the section. Finally, at the opposite end of the text from where you started, you executed the MASS-11 Cut function, which is similar to lifting the section of text from the document. Unlike the knife and paper operation, however, you are not left with a gaping hole in your document. MASS-11 automatically moves the text below the cut up to meet the text above the cut, so that there is never a hole left by the Cut operation.

The cut text is stored in a temporary holding area, or "paste buffer". The paste buffer contains the cut text until another piece of text selected with [SEL] is cut or copied, until you change Document Directories, or until you exit MASS-11. The amount of text that can be cut at one time is limited only by the disk quota allocated to your account by the System Manager.

6.4.2.2 Rulers in Cut Text

If the text you select has any rulers embedded in it, these rulers will also be stored in the paste buffer with the text. If you paste the cut text into another location, these rulers will be inserted into the document along with the cut text. When you cut text with rulers from a document, the last ruler which occurred in the cut text will be placed in the document at the point of the cut. This will preserve the format of the text which remains in the document below the point of the cut.

Figure 4. Sample page from Mass-11 manual

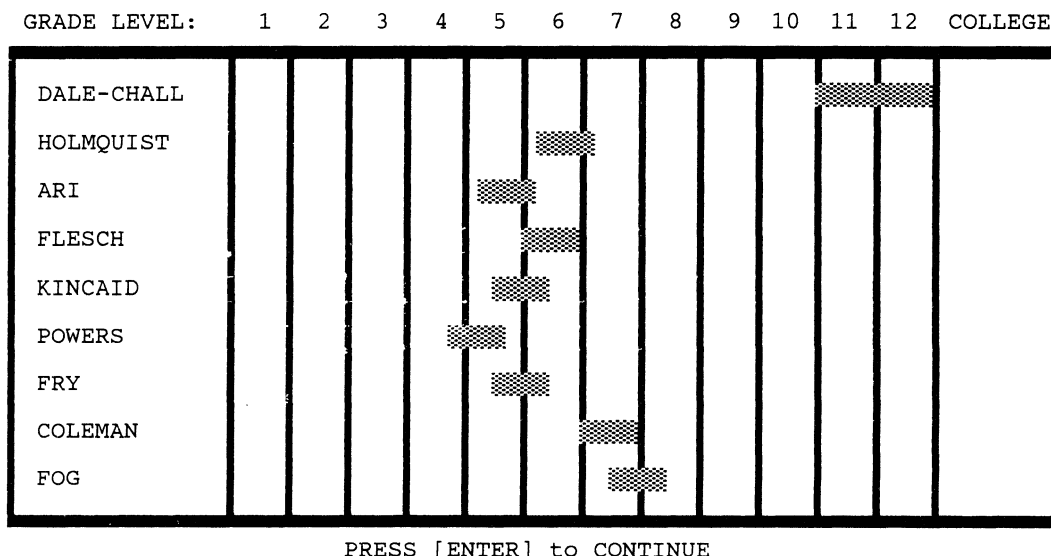


Figure 6. Comparison of reading levels for sample text.

Default Options

Screen 1-1 shows the list of default options you can keep or change. If you do not type an answer to an option, Samna uses a default answer as shown to the left of each option. An explanation of each option follows Screen 1-1.

```

To change standard options, Tab to each position.
Type in the change and then press Return.
-----
Col 2 Ln 6 Pg 1 File
-----
HERE IS YOUR DEFAULT PAGE

Remove the write-protect tab if you are using floppies
Tab to each option to make your selections then touch Return

(N) Do you wish to change the default format?
(.) Type the terminator you want to use for the numeric tabs. (Use . or ,)
(N) Do you wish SAMNA to automatically create back up files?
(N) Do you wish to have the print option for repage set for "yes"?
(C) Type the desired print option you wish to use as a default.
    C:continuous paper  S:single sheet paper  F:sheet feeder
(Y) Is your feeder a single-tray feeder? Answer Y (for yes) or N (for no).
(B) Type the identifier for the drive you wish to use as the default. (A - P)
    ( ) Type the default directory name.
(T) Do you wish to have the more extensive HELP level 2 as your default help?
(I) Type the print wheel number you wish to use as the default. (1 - 6)
    1:DEC LA50/100 Multinational  2:DEC LA50/100 Multinational
    3:DEC LA50/100 Multinational  4:DEC LA50/100 Multinational
    5:DEC LA50/100 Multinational  6:DEC LA50/100 Multinational
(Y) Do you wish to display the SAMNA borders on the screen?
(H) Do you wish to have character dots on your screen?
(I) Type the appropriate number for the keyboard you want for a default.
    1:English  2:Swiss/German  3:Swiss/French  4:Swedish  5:Spanish
    6:Dutch  7:British  8:Canadian  9:Austrian/German  10:Belgian/French
(I) Which style represents how negative numbers should be shown in math? (1-3)
    1: -12,890.98  2: 12,890.98-  3: (12,890.98)
(Q) How many decimal positions do you want to use as a default in math?

```

Screen 1-1. The Default Options Screen

NOTE

For each option setting that you want to keep, press the Tab key. This moves you to the next option.

Do you wish to change the default format?

Press the Y key (for Yes) if you want to change the format. Otherwise, press the Tab key; the settings remain as they are.

Your margins, tabs, and page specifications are preset. These settings are automatically displayed on the scratchpad and on any new file you create.

Samna displays the current default format at the top of the screen above the default screen. If you answer Yes to this question, the cursor moves to this format line, and you can change any of the settings.

Type the terminator you want to use for numeric tabs.

Press the comma (,) key if you want the terminator to be a comma.

When you tab to and type at a numeric tab stop, Samna automatically aligns numbers on the terminator. The terminator is usually a period (.). This means that your numbers are aligned on the decimal (in columns) under the numeric tab stop. Changing the character on which Samna aligns the numbers to a comma (,) is useful if you are typing in another language using statistical data.

Do you wish Samna to automatically create back up files?

If you specify Yes, Samna automatically makes a back-up copy (exact duplicate) of each file you save. The back-up copy always has exactly the same file name as the original file. However, Samna assigns a file type of .BK1 to the back-up file.

If you back up the same file again, Samna renames the file called *file.BK1* to *file.BK2*, and creates a newer back-up file with a file type of .BK1.

Do you wish to have the print option for repage set for "yes"?

If you press the Y key (for Yes), Samna changes the page break locations in your file during print.

The Repage option automatically adjusts the number of lines per page so that the page lengths are consistent. If you do not usually repage on the screen before printing, specify Y. You can always selectively change your setting for a particular print job.

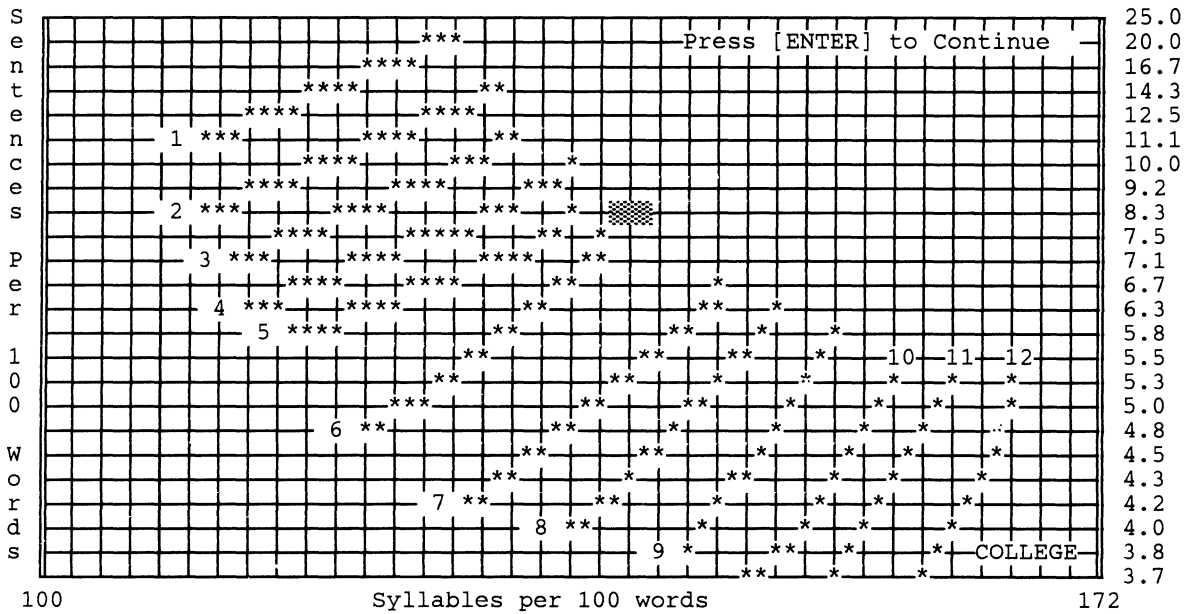


Figure 7. Fry graph from sample text.

PC-Style report for: B:REFERENC.GDE

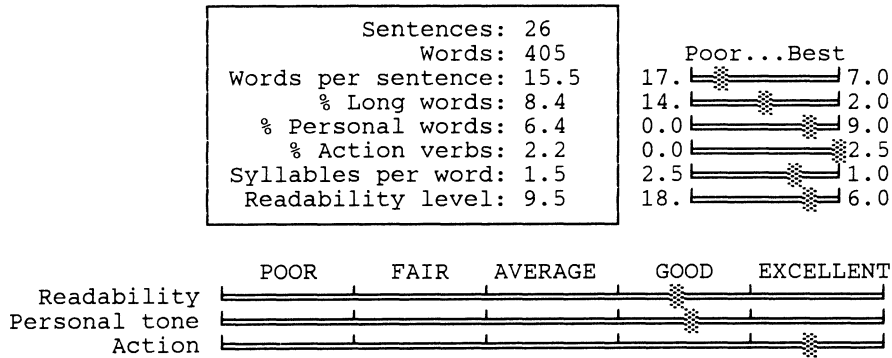


Figure 8. Analysis of sample from PC-Style

Table 1. Analysis of sample text

CURRENT PASSAGE		TEXT11	
398	WORDS	557.07	SYLLABLES
28	3-SYLLABLE WORDS	139.97	SYLLABLES PER 100 WORDS
30	SENTENCES	7.54	SENTENCES PER 100 WORDS
	8.12		FOG READING LEVEL
	74.96		FLESCH READING EASE SCORE
	6		FLESCH GRADE LEVEL
	5.20		POWERS READING EASE
	6.85		HOLMQUIST
	5.72		ARI
	6.10		FLESCH-KINCAID
	7.59		COLEMAN
	11-12		DALE-CHALL
DO YOU HAVE MORE MATERIAL? (YES OR NO)?			

Table 2. Words listed by Dale-Chall readability formula

Words used in the manual sample not found in Dale-Chall listing.

(#) = Number of occurrences

* (p. #) = Word defined in manual on page # in Getting Started

\$(p. #) = Word defined in manual on page # in Reference Manual

adjusts (1)	option (6)
aligned (2)	options (3)
aligns (2)	original (1)
assigns (1)	particular (1)
automatically (6) \$(p.1-24)	per (1)
BK (4)	period (1)
character (1)	preset (1)
comma (3)	press (5)
consistent (1)	renames (1)
create (2)	repage (3) \$(p. 1-24)
creates (1)	Samna (10)
current (2)	scratchpad (1)
cursor (1) *(p.1-6) \$(pp.1-6,1-8)	selectively (1)
data (1)	specify (4)
decimal (1)	specifications (1)
default (6) \$(p.1-23 to 1-27)	statistical (1)
displayed (1)	tab (5)
displays (1)	tabs (2)
duplicate (1)	terminator (5)
explanation (1)	type (5)
formats (5)	typing (1)
locations (1)	usually (3)
margin (1)	Y (4)
numeric (3)	

APPENDIX

PROGRAM	SECTION	TEXT11	AVERAGE FROM TEXT1-TEXT7; TEXT9
READABILITY	# words	408	373
	3-Syllable Sentences	28	22.5
	Syllables	559.00	505.25
	Syll./100 words	137.01	137.25
	Sent./100 words	6.86	6.8
	FOG Reading	8.57	9.0
	Flesch Ease	76.13	75.1
	Flesch Grade	6	6.2
	Powers Ease	5.16	5.2
	Holmquist	4.19	6.5
	ARI	5.94	6.4
	Flesch/Kincaid	6.25	6.6
	Coleman	7.24	7.3
	Dale Chall	4th or less	9.3
GRADE LEVEL	Dale Chall	1-4	9.3
	Holmquist	4.25	5.8
	ARI	6.00	5.8
	Flesch	6.00	6.2
	Kincaid	6.35	6.0
	Powers	5.35	4.7
	Fry	6.95	5.7
	Coleman	7.25	6.6
	Gunning Fog	8.00	8.9
GRAMMATIK	Av. Sent. Length	15.5	14.6
	Av. Word Length	4.5	4.2
	Longest Sentence	44.0	30.3
	Shortest Sentence	6.0	6.0
	"To be" Verbs	9.0	6.25
	Prepositions	57.0	46.7
COMMENT	Content	71.6	69.5
	"To be" (30%)	4.62%	12.1%
	Prepositions (2/Sent.)	2.0	2.1
	Transitions (20%)	11.54%	24.0%
	"Th" Openers (9%)	19.23%	14.0%
	Vagueness (1%)	1.23%	0.9%
	Short Sentences (30%)	46.0%	42.7%
	Long Sentences (15%)	4.0%	2.6%
	Problems Noted	6.0	5.9

RT-11 SIG

TSXLIB: Updated for TSX-Plus V6.0

N. A. Bourgeois, Jr.
NAB Software Services, Inc.
Albuquerque, NM

TSXLIB is a library of FORTRAN callable routines that implement the TSX-Plus system services which are unique to TSX-Plus. The library has been updated to include all TSX-Plus unique services through TSX-Plus V6.0.

Introduction

Like RT-11, TSX-Plus offers the MACRO-11 programmer a number of system services. These services are implemented via both the RT-11 programmed requests (for those services common to both RT-11 and TSX-Plus) and raw EMT instructions (for those unique to TSX-Plus). RT-11 makes its system services available to the FORTRAN programmer through the system subroutine library, SYSLIB. TSX-Plus also honors the bulk of the service requests in the SYSLIB routines. TSXLIB, however, makes the TSX-Plus unique EMTs available to the FORTRAN programmer.

These TSX-Plus library routines provide facilities to support communication lines, detached jobs, device allocating and deallocating, file structured device mounting and dismounting, communication between running programs, job privileges control, job status monitoring, program performance analysis, real time program execution, shared run time systems, shared files, special files information, spooler control, communication between running programs and a terminal, program control of the terminal, ODT activation mode, user name control, windowing, and several miscellaneous EMTs.

The TSXLIB distribution kit, DECUS #11-490, includes the MACRO-11 source modules for all the routines, a user's manual in machine readable form, an indirect command file to build the library, and the implemented library.

General Description

These TSX-Plus library routines provide facilities to support communication lines, detached jobs, device allocating and deallocating, file structured device mounting and dismounting, communication between running programs, job privileges control, job status monitoring, program performance analysis, real time program execution, shared run time systems, shared files, special files information, spooler control, communication between running programs and a terminal, program control of the terminal, ODT activation mode, user name control, windowing, and several miscellaneous EMTs.

The standard FORTRAN subroutine calling sequence shown below may be used to access all of the routines in the TSX-Plus library.

```
CALL RTNAM ( ARG1,...,ARGn )
```

These routines that return only one value are also callable as FORTRAN functions. This is as follows:

```
IRET = RTNAM ( ARG1,...,ARGn )
```

The application program is first compiled as follows:

```
.RUN SY:FORTRAN  
*MYPROG=MYPROG  
**C
```

Then the routines are linked with the application program as shown below:

```
.RUN SY:LINK  
*MYPROG=MYPROG,TSXLIB/F  
**C
```

or

```
.RUN SY:LINK  
*MYPROG=MYPROG,TSXLIB,SYSLIB,FFULIB  
**C
```

The application program is now ready for execution.

```
.RUN DK:MYPROG
```

The "TSX-Plus Reference Manual" describes how the EMTs implemented in this TSX-Plus library are accessed from a MACRO-11 program. However, the FORTRAN/MACRO interface described in the "RT-11 Programmer's Reference Manual" may also be used to access the routines in the library.

In the sections that follow, the name of the MACRO-11 source file containing the listed routines is given in the title of each of the tables.

Communication Line Support

Table 1 lists the TSXLIB routines that offer certain communication line support from within a running program.

```
IRDRCL Redirect a communication or timesharing  
line.  
ICNTSP Control the speed of a communication or  
timesharing line.
```

Communication line support.
DLLINE.MAC
Table 1.

Detached Job Support

Table 2 lists the routines that provide detached job support from within an executing program.

```
ISTDJ Get the status of a detached job.  
KLDTJB Kill a detached job.  
STDTJB Start a detached job.
```

Detached job support.
DETJOB.MAC
Table 2.

Device Allocating and Deallocating

System services are available to allow a running program to control the allocation of a device for exclusive use. Table 3 lists these routines. Once a device has been allocated to a job, no other job is allowed to access the device. A device allocated to a primary line or associated virtual line may be accessed by any of the associated lines.

```
IALOC Allocate a device.  
IDALOC Deallocate a device.  
ICALOC Check the allocation status of a device.
```

Device allocating and deallocating.
EVALOC.MAC
Table 3.

Device Mounting and Dismounting

It is possible to mount and dismount a file structured device for directory caching from within a running program. The routines listed in Table 4 provide these capabilities.

DISMNT Dismount a file structured device.
MOUNT Mount a file structured device.

Device mounting and dismounting.
MNTDEV.MAC
Table 4.

Interprogram Message Communication

TSX-Plus provides an optional facility that allows running programs to communicate with each other. Table 5 lists the routines that support this interprogram message communication.

MSGREQ Post a read request for a message.
MSGSEND Send a message to another job.
RCVMSG Try to receive a message from another job.
RCVMSW Wait for a message from another job.

Interprogram message communication.
MSGCOM.MAC
Table 5.

Messages are transferred between programs by using named message channels. A message channel accepts a message from a sending program, stores the message in a queue associated with the channel, and delivers the message to a receiving program on its request for a message on the channel. Message channels are separate from I/O channels.

Each active message channel has associated with it an ASCII character name that is used by both the sending and receiving programs to identify the channel. The names associated with the channels are defined dynamically by the running programs. A message channel is active when messages are being held in the queue associated with the channel or if a program is waiting for a message from the channel. When message channels become inactive they are returned to a free pool and may then be reused.

Once a message is queued on a channel, that message will remain in the queue until some program receives it. A program's exiting to the keyboard monitor does not remove any pending messages. This allows one program to leave a message for another program that will be run at a later time.

Job Privileges Control

Table 6 lists the routines that permit an executing program to determine and make certain changes in its own executing environment.

IPRIV Determine or change the job privilege sets.
ISTPRV Set the current job's priority value.
NONINT Set the [non]interactive job status.

Job Privileges Control
JBPRIV.MAC
Table 6.

Job Status Monitoring

A facility is available for one job to monitor the status of one or more other jobs. Once a job has declared to monitor another job, a completion routine is executed in the monitoring job whenever a change of status occurs in the job being monitored. The services provided for job monitoring are listed in Table 7.

IESMCN Establish a job status monitoring connection.
ICNMCN Cancel a job status monitoring connection.
IBSTCH Broadcast a job status change.

Job status monitoring.
JBSTMN.MAC
Table 7.

Miscellaneous EMT Support

Table 8 lists the routines that support the several miscellaneous EMTs provided by TSX-Plus.

GETREG Get the word and byte values stored in a processor register.
IHERR Enable abort on certain errors.
ISERR Inhibit abort on certain errors.
ISPY Return values from within the simulated RMON (SYSLIB routine).
ITSLIC Determine the TSX-Plus license number.
ITSLIN Determine the TSX-Plus line number.
MEMSET Set the memory allocation.

Miscellaneous EMT support.
TSXMSC.MAC
Table 8.

ODT Activation Mode Support

ODT activation mode may be turned on and off from within a running program. Table 9 lists the routines that support this feature. In this mode TSX-Plus considers all characters to be activation characters except the digits, the comma, the dollar sign and the semicolon.

RSTODT Reset normal activation mode.
SETODT Set ODT activation mode.

ODT activation mode support.
TSXODT.MAC
Table 9.

Performance Analysis Support

For many applications the keyboard monitor level performance analysis control provided by TSX-Plus is adequate. Specifically, in cases such as analyzing the performance of an overlaid program it is necessary to have control over the performance analysis feature from the running program. The routines listed in Table 10 provide just this capability.

INTTPA Initialize for a performance analysis.
ISPPA Stop a performance analysis.
ISTPA Start a performance analysis.
TERMFA Terminate from a performance analysis.

Performance analysis support.
PRFANL.MAC
Table 10.

Real Time Program Support

The real time program support provided by TSX-Plus allows multiple real time programs to be run concurrently with normal time sharing operations. The basic functions provided by this facility are listed in Table 11.

A program must have operator privilege to use any of the real time routines. The real time facilities are available to both normal jobs controlled by time sharing lines and to detached jobs. Detached jobs started by time sharing users have operator privilege only if the user starting them does.

A basic facility required by many real time programs is the ability to access the I/O page which contains the peripheral device registers. A normal time sharing job does not have this access. It is instead mapped to a simulated RMON. This allows programs that directly access offsets into RMON to run correctly.

A real time program can access the I/O page in one of two ways. It can map the I/O page into the program's space, or it can leave the simulated RMON mapped into the program's space and perform peek, poke, bit set, and bit clear operations into the I/O page. It is much more efficient to directly access the device registers by mapping the I/O page into the program's space than to use the routines to perform each individual access. However, this technique will not work if the program must also directly access offsets into RMON. The correct way to access offsets into RMON is with the SYSLIB routine, ISPY, which will work even if the I/O page is mapped into the program's space.

CNVAPA Convert a virtual address to a physical address.
 IBICIO Bit clear a value into the I/O page.
 IBISIO Bit set a value into the I/O page.
 ICNINT Connect an interrupt vector to a completion routine.
 ICNRTN Connect a service routine to an interrupt vector.
 IPEKIO Peek at a value in the I/O page.
 IPOKIO Poke a value into the I/O page.
 IRLINT Release an interrupt vector connection.
 ITCRTN Trigger a completion routine from an interrupt service routine.
 IUNLKM Unlock a job from memory.
 LKANMY Lock a job into any memory.
 LKLCMY Lock a job into low memory.
 MPIOPS Map the I/O page into the program space.
 MPRGN Map a region of virtual memory to a specified region of physical memory.
 MPRMPS Map the simulated RMON into the program space.
 RLCTL Relinquish exclusive control of the system.
 STPLRV Set the user mode processor priority level.
 TKCTL Take exclusive control of the system.

Real time program support.
 RELTIM.MAC
 Table 11.

The TSX-Plus real time support facility provides two methods to connect real time interrupts to TSX-Plus jobs. The first of these methods uses the mechanism of completion routines to perform the connection. The second method provides a more direct connection between interrupts and service routines in TSX-Plus jobs. Once a connection is established the routine, completion or service, is executed each time the specified interrupt occurs. It is possible for several interrupt vectors to be connected to the same completion routine in a job but it is illegal for more than one job to connect to the same interrupt vector.

With the first method an interrupt causes a completion routine to be queued for the job that was connected to the interrupt vector. This method is very general and allows the completion routine to call for system services (execute EMTs). Also the job does not have to be locked in memory. On entry to the completion routine, R0 contains the address of the vector that caused the completion routine to be entered.

The second method sets up conditions for the job being called and enters the interrupt service routine. This approach minimizes the overhead in entering the service routine. Hence, the service routine method can process interrupts at a greater rate than can be processed using the completion routine method.

There are restrictions imposed on a program using the service routine method. Specifically, the following conditions must be met to directly connect to an interrupt:

1. The job must be locked in memory before connecting it to the interrupt vector and must remain locked in memory as long as the interrupt connection is in effect.
2. The processing done by the service routine should not be very lengthy since other interrupts may be queued up during execution of the service routine.
3. Only two system services may be legally called from within the service routine. They are the SYSLIB routine, RESUME, and the TSXLIB routine, ITCRTN.

The interrupt service routine runs in user mode and uses memory mapping that has been established for the job before the interrupt occurs. Access to the I/O page via the MPIOPS routine is possible provided this mapping has been set up in the mainline code before the interrupt occurs.

If an interrupt service routine needs to perform a system service other than the RESUME call listed above, it should trigger a completion routine via a call to ITCRTN and have the completion routine call for the required system service(s).

An execution priority may be specified for each completion routine. This is not the same as the

hardware selected priority of the interrupt. All completion routines are synchronized with the job and run at hardware priority level zero. The completion routine priority is used to schedule completion routines for execution. The available priority levels are 0 through 127. The execution of a real time completion routine for one job will be interrupted and suspended if an interrupt occurs that causes a higher priority completion routine for another job to be queued for execution. However, a completion routine for a given job will never be interrupted to run another completion routine for the same job even if a higher priority completion routine is pending.

Completion routine priorities one and larger are real time priorities. They are higher than the priorities given to time sharing jobs and will always preempt the time sharing jobs. Completion routine priority zero is not a real time priority but rather a very high normal priority. Such zero priority completion routines are time sliced in the normal fashion. If a completion routine enters a wait state it relinquishes its real time priority. Jobs that have real time, interrupt driven completion routines need not be locked in memory.

In time critical, real time applications where a program must respond to an interrupt with minimum delay, it may be necessary for the job to lock itself in memory to avoid the time consumed in program swapping. This facility should be used with caution since if a number of large programs are locked in memory there may not be enough space left to run other programs.

A running program may gain exclusive access to the system to perform some time-critical task. The program may then relinquish this exclusive access when it is not needed. A running program may also set the user mode priority level and map a region of virtual memory to a specified region of physical memory.

Shared Run Time System Support

TSX-Plus provides a facility that allows shared run time systems or data areas to be mapped into the address spaces of multiple time sharing jobs. Table 12 lists the routines that support this feature.

IASRNT Associate/disassociate a shared run time system with a job.
 MAPRNT Map a shared run time system into a job's region.

Shared run time system support.
 RUNTIM.MAC
 Table 12.

Memory space can be conserved by having several jobs access a common copy of a run time system rather than having to allocate space within each job. Shared run time systems are never swapped out of memory. When a job is associated with a run time system, a portion of the job's virtual memory is mapped so as to allow access to the run time system.

Shared File Support

Table 13 lists the routines that offer access to the shared file record locking facility provided by TSX-Plus. This is useful in situations where programs being run from several terminals wish to update a common file. Through the record locking facility a program may gain exclusive access to one or more blocks in a shared file by locking those blocks. Other users attempting to lock the same blocks will be denied access until the first user releases the locked blocks.

ICKWTS Check for writes to a shared file.
 IDCLSF Declare a file to be shared.
 ISVST Save the status of a shared file.
 IUALBK Unlock all locked blocks.
 IUSPBK Unlock a specific block.
 LKBLK Try to lock a block.
 LKBLKW Wait for a block to lock.

Shared file support.
 SHRFIL.MAC
 Table 13.

The recommended procedure for updating a shared file being accessed by several users is as follows:

1. Open the file.
2. Declare the file to be shared.
3. Lock all blocks which contain the desired record.
4. Read the locked blocks into memory.
5. Update the record.
6. Write the updated blocks to the file.
7. Unlock the blocks.
8. Repeat steps three through seven as required.
9. Close the file.

Special File Information

TSX-Plus allows a running program to obtain certain status information about a file and to set the creation time of a file. Table 14 lists the routines that support this facility.

- IFLINF Obtain some information about a file.
 ISTFTM Set the creation time of a file.

Special file information.
 TSFILS.MAC
 Table 14.

Spooler Support

Table 15 lists the routines that provide spooler support available to an executing program.

- ISPBLK Determine the number of free blocks in the spool file.
 ISPCTL Control the release time of a spooled file.

Spooler Support.
 SPOLER.MAC
 Table 15.

System Status Information

Information typical of that returned by the SYSTAT keyboard command is made available to a running program by the routines listed in Table 2-16.

- ICONTM Determine the connect time for a job.
 ICPUTM Get the CPU time used by a job.
 IEXSTS Get a job's execution status.
 ILNSTS Check the status of a line.
 IPGNAM Get the name of the program being run by a job.
 IPPNUM Get the project-programmer number for a job.
 MEMPOS Determine the position of a job in memory.
 MEMUSE Determine the amount of memory used by a job.

System status information.
 SYSTAT.MAC
 Table 16.

Terminal Communications Support

The routines that allow a running program to communicate with a terminal are listed in Table 17.

- TRMIN Accept a string of characters from the terminal.
 TRMSG Send a message to another terminal.
 TRMOUT Send a string of characters to the terminal.

Terminal communications support.
 TRMCOM.MAC
 Table 17.

Terminal Control Support

The terminal control support routines are listed in Table 18.

- BRKCTL Establish break sentinel control.
 HIEFOF Turn off the high efficiency terminal mode.
 HIEFON Turn on the high efficiency terminal mode.
 IACTCH Check for pending activation characters.
 ITRCTL Perform lead-in character type terminal control functions.
 ITRERR Check for terminal input errors.
 ITRTYP Determine the terminal type.
 TIMOUT Set the terminal read time out value.
 TTCRTN Establish a terminal completion routine.

Terminal control support.
 TRMCTL.MAC
 Table 18.

User Name Support

An executing program may set and change the user name for the connected job. The routines for this facility are listed in Table 19.

- GTUNAM Get the user name associated with the current job.
 STUNAM Set the user name associated with the current job.

User name support.
 USRNAM.MAC
 Table 19.

Windowing Support

TSX-Plus offers a full screen process windowing system that allows the system to remember the contents and status of the terminal display. A running program may redisplay windows on demand. The routines for this facility are listed in Table 20.

When windowing is used, the system monitors all characters sent to the terminal and maintains an updated screen image display in memory. Terminal attributes such as line width, reverse/normal video, application keypad mode, and etc. are saved along with line attributes (double wide, double high) and character attributes. The attributes retained for each character consist of blinking, bold, underlined, reverse video, and character set information (ASCII, UK, DEC supplemental, or graphics).

The most common use of windows is to completely refresh the display when switching among sub-processes to avoid the confusion of mixed displays. For some program applications, it may be useful to utilize multiple windows from the same job.

- ICRWND Create a window.
 IDLWND Delete a window.
 IPRWND Print the contents of a window.
 IRSWND Resume window processing.
 ISLWND Select a window.
 ISPWND Suspend window processing.

Windowing Support
 WINDOW.MAC
 Table 20.

RSX-11 SIG

"MACHIAVELLI": AN ENGINEERING APPLICATIONS DEVELOPMENT ENVIRONMENT IN DECUS "C" UNDER RSX 11M PLUS

Richard Wittenoom
Richard Wittenoom & Associates Pty Ltd
Consulting Chartered Engineers
West Perth 6005, Australia

ABSTRACT

The paper describes the software development environment in the author's consulting engineering firm in Perth, Australia, where a broad spectrum engineering applications system is being developed on PDP-11 hardware using DECUS "C" under the RSX 11M Plus operating system.

1.0 WHY DEVELOP?

The decision to commence a program of computer related development was made by the author's firm in the late 1970s as a hedge against the uncertain future in consulting engineering work in the firm's area of expertise. The goal set was to automate all possible aspects of small consulting engineering practice.

In the preceding ten years the firm had worked through successive mining development booms producing thousands of similar drawings, many of them standard, generally relating to town planning and services development in mining towns. With the cyclic nature of this work (boom or bust) and the resulting problems of staffing, financing and management, intensive use of computers appeared to offer a way to enable a small, skilled, base load work force to extend its productivity to handle the profitable peak periods. At the same time there appeared to be no suitable software available which would allow automation of this work area (as opposed to computer assistance).

2.0 DEVELOPMENT DIRECTIONS

Development concentrated first on an automated drawing production system. Although limited in its initial implementation, this was conceived as a general purpose technical system, for which generation of drafting output was only one possible result. The original system has been used in production with considerable success for the past 4 years, during which time options for future development have been progressed. Because of the envisaged "behind the scenes" role extending across many aspects of consulting practice, the evolving system has been given the name "Machiavelli".

Despite the pragmatic initial goals, an overriding design decision was made that the system should be as un-bounded as possible. This implied indirect definition capabilities, data dependent or situation dependent access or processing modes, and generally a measure of intelligence. The long term implications of this approach were not fully realised initially but have become apparent as systems evolved with prototype development and use.

Another design decision was that the system should model an engineering object as an un-bounded data set, rather than as a particular "view" or side effect. The definition of a sewer line or a structural member may thus take the form of a complex database, rather than as graphics primitives with associated attributes.

Such things as the set of graphics primitives required to draw it, or the stiffness matrix used for structural analysis, become derived views of the object as known to the system.

While in a typical CAD system objects are defined directly in terms of their physical representation (e.g. edges, surfaces, wireframes etc), objects in a Machiavelli system can be any tangible or intangible thing, concept or relationship, provided they can eventually be mathematically defined. The production of any specific attributes is regarded as an end (or a side effect) of processing, rather than as the representation of the object itself.

Another basic difference is that any part of the data in the set associated with an object may be defined indirectly in terms of data in sets associated with other objects. In this context the fact that another object is "referenced" implies that there will exist (or can be obtained) accessing and processing functions available to select from the data in the other set and return the selected data or a derivation of it to the higher level object.

The ability to learn during operation is seen as essential. An object can thus be considered to be defined by the union of:

{ {the information actually stored}
and
{information required at the time} }

and the system needs to be capable of both recognising the need for and asking for any additional data, rules or processing required in a particular situation.

An implementation decision was that early development should concentrate on automation of output rather than optimisation of input (particularly graphic input). As a result the production system now in use relies largely on pre-processing of batch input data under system control rather than human interaction by graphics means. With the prototype "back end" system now operational a hardware based screen interface will be added in due course. This course was adopted in view of the rapid development in workstation hardware and software.

3.0 FUTURE EMPHASIS

It has become increasingly obvious that what initially appeared to be engineering specific, or even application specific processes were in fact substantially the same as processes in completely different areas. Particularly as high performance graphics becomes an integral part of almost any commercial application, the processing functionality required in commercial development is becoming asymptotic to that required for an engineering programming environment. As a result, work on apparently different applications has now been integrated wherever possible.

The goal now is to set up a single high level software development environment within which to develop a wide range of applications, ranging from basic processing and reporting to intelligent systems, and covering both commercial and technical applications.

In most cases there will be a "4GL"-type option which will allow the inputs, outputs and processing to be defined and run, either interpretively or compiled on the fly. Below this, an applications development environment will make available to the applications programmer a set of high level library functions which can be used for development of specific utilities.

The environment encountered by an application developer will be hardware and system independent, with porting implemented at low level using interchangeable libraries and macro definition modules.

4.0 WHY PDP-11?

Despite its maturity, the PDP-11 architecture still gives the most effective "bang per buck". It is a hardy environment for mixed development and production. By virtue of its known brick walls it encourages good programming practices. It is considered by the firm to be a better starting point for development over a wide range of systems. "If it will work on an 11, it can be ported to almost anywhere." The same does not always apply to development carried out in a VAX environment.

5.0 WHY RSX?

RSX-11M Plus is considered the most cost efficient general purpose operating system for a mixed commercial, technical development and production office. It is extremely functional, is increasingly making more task space available to the programmer (with I/D space, supervisor mode libraries etc) and without the overhead of virtual memory management is much more responsive than an equivalent VMS system. It is closer to a number of other operating systems likely to be development targets for the firm's software than VMS.

Task size restrictions on the 11 are being reduced by the new features offered by the "J" based processors (e.g. I/D space separation, supervisor mode libraries etc) in addition to more familiar methods such as use of multiple intercommunicating tasks. The exploitation of such options is included in the firm's development program. However the details of the implementation will be transparent to the application programmer.

5.0 WHY "C"?

"C" is a language which allows program development at a wide range of levels. A programmer can write very low level code, to exploit the system to the maximum. This is particularly so on PDP-11s, on which the language was popularised. Operations can include direct bit-bashing, and with an efficient compiler there is little advantage in writing all but the most critical functions in assembler. At the same time it can be used as a very high level language.

"C" makes it relatively easy to structure applications so that they can be ported to a wide range of systems, provided one accepts the need to plan for this. It is necessary to separate out modules which are likely to be hardware or operating system dependent into libraries which are tailored to each implementation. The overall philosophy of "C", in which the language itself is small and most system and hardware dependent activities (including I/O) occur in functions not forming part of the language, means that library functions are not regarded as sacred, and programmers approach the tailoring of low level libraries to new systems or devices without major hang-ups.

To set up an environment in which applications can be quickly developed it is only necessary to build up libraries of standard functions covering the range of procedures normally expected in the target applications. The applications modules can then be restricted to quite high level code.

A disadvantage of "C" is that because of its powerful features it can be exploited by expert programmers to produce code which is super-efficient and small, but is also cryptic and impenetrable. A novice programmer can also write "C" like Basic, complete with the GOTOs. Somewhere in the upper end of this range lies the desirable result.

Because the compiler does not enforce such restrictions as array bound checking, "C" programs are vulnerable to carelessness in programming, with "dangling" or uninitialised pointers a source of frustration. This type of problem can take some time and energy to track down. Low level development in "C" is thus better carried out by experienced programmers, rather than occasional engineering users.

6.0 ENGINEERING PROGRAMMER INTERFACE

The approach taken in the firm's development is to present the "occasional application developer" with a very high level interface termed "EPIC" (Engineering programmer interface to "C"). This provides a relatively safe level at which non expert users can develop compiled applications. The overall structure of this environment is set out in the following Table.

(HARDWARE AND SYSTEM INDEPENDENT)	
APPLICATIONS LEVEL code at this level is largely calls to lower level routines	
EPIC	
PROCESSING MODULE LEVEL application oriented library processing modules	
DATA IMPLEMENTATION LEVEL this level implements the data structures (e.g. file processing procedures, memory structures, etc (device and hardware independent)	
(HARDWARE OR SYSTEM DEPENDENT)	
SYSTEM AND HARDWARE DEPENDENT GENERIC IMPLEMENTATION LAYER Set of routines used by higher level functions to interface to the system.	
LIBRARIES	
system/machine independent 'C' library functions	system/machine dependent 'C' library functions
SYSTEM IMPLEMENTATION LEVEL operating, file control, etc, systems	

TABLE 1: "EPIC" Engineering Programmer Interface

7.0 WHY DECUS-C?

DECUS-C is available to members as a normal Library distribution (11-SP-18). The most recent update of the compiler is included in the RSX Fall 85 SIG Tape. It is now a fully functional implementation, and includes a suite of UNIX style software tools and utilities.

There are a number of commercial "C" compilers for the PDP-11. However DECUS-C offers the rare combination of an open language (with full sources) and a nominal price, thereby opening up the opportunity to program in "C" on many sites where the cost of a full license or support for a further language cannot be justified.

As a DECUS Library program, its purchase cost is insignificant in comparison with that of other similar computer languages. However as with most Library packages there is no guaranteed level of support, and the cost of upgrading in-house expertise to provide this support, (or paying for it on a contract basis) must be taken into account when comparing it with other available "C" implementations. Much of this support is, however, being provided by the Decus Membership at no cost. For example, details of bugs found by Australian users, and their fixes, have been submitted to the Spring 86 RSX Symposium Tape, together with a number of developmental modules related to this paper.

The original development of the compiler was carried out with a view to portability of low level code across all PDP-11 based systems, and was developed with a RSTS-E flavour. As a result, much of the interaction with the RSX operating system is less efficient that could be desired. However with the number of RSX users increasing it is considered that user contributed enhancements will offset such problems.

8.0 SOFTWARE DEVELOPMENT ENVIRONMENT

Components of the software modules forming the lower levels of the development environment include:

- standard command line interface (GCL),
command parser and command dispatcher
- screen windowing, editing and data
structure control functions
- workstation and hardcopy graphics functions
- report generator utilities
- command interpreter with fast binary
file compiler
- fast binary file loader
- intelligent database manager functions
- file and direct device I/O libraries
- hardware dependent libraries
- operating system dependent libraries
- device and system independent special
purpose libraries (e.g. scientific
subroutines, matrix processing, etc)

Higher level functionality is provided by appropriate shells used in conjunction with the lower level library modules.

9.0 DATABASE MANAGER

The file system is transparent to a high level applications function. I/O may be to a region in

memory, a file on disk, a RAM disk, a file on a remote network node, or a cooperating task. The routing for a particular channel is determined by a current "context" maintained by the file system. I/O functions may be either built with the task or in a separate data server task (which may be elsewhere on the network).

The database function library recognises the following file types:

- fast binary files
- terminal format files
- fixed length record files
- slab files
- index files (various formats)

Fast binary files are organised and accessed in "pages", each page being a single disk block. These files are used with a module which will load data structures or lists, already initialised with data, into memory. This is used to load precompiled processes, record mapping, database information and screen display control data structures. A command interpreter and compiler maintains and updates these files.

A "slab file" is a file addressable with a granularity of 4 bytes - i.e. it is 'longword addressable.' In this type of file records of varying length are stored in whole numbers of contiguous 4 byte "slabs". Data files in any but the simplest applications are treated as slab files.

A complex record access procedure permits "records" to be in fact complex linked lists in file. This allows for records to "own" large variable length data areas or to be extended indefinitely after initial definition. Management of record I/O is controlled by a memory resident "context", a linked list defining the record structure which is loaded from a fast binary file as required. In an extreme case, this would allow a single record to define a complete ISAM database.

Access modes to data files may be mixed, allowing processing of files in either relational or networked views or as virtual arrays, regardless of whether tuples of the particular record type exist physically as fixed length records in a file or as sets of slabs in a file shared with records of other types. A file which is accessed as part of a networked database by an engineering application may also be accessed relationally by the same (or some other) application.

Engineering applications tend to involve the need for fast network or array processing. They also often involve indirect definition. In addition, however, each type of object may have associated with it a number of processing options.

Objects in such an indirectly defined database are assigned a more complex system of identification, in which every data set (or object) is allocated a "genre", an "ordinal", and a "class. The genre indicates the "family" to which the object belongs, based on the assumption that all objects of a genre can be "resolved" to an absolutely defined current value set. The "ordinal" uniquely identifies each object. The "class" identifies a processing envelope. In addition objects may be grouped in hierarchical "families" or in the case of "many to many" relationships by a system of "knobs".

Database functions can access objects by functional specifications (i.e. genre:ordinal) or location (file:slab). In the former case the object is accessed through an index maintained for each genre. The latter mode is particularly suited to applications requiring fast access, while the (g:o) mode is more convenient for safe development and debugging of a major engineering

database involving large numbers of indirect reference pointers, particularly when the initial records are likely to change substantially. The default mode for part or all of the database can be changed once the situation is more stable, and (g:o) mode references can be changed to (f:s) mode as each is accessed.

Either of these address modes will map into the system's standard address format, a 32 bit longword which is known as a REC_ADR (record address). A single 32 bit word maps to either of the above access modes or to a 30 bit "slab" number, with the mode in use set by bits 30 and 31 as status bits. Pointers from within an object to other objects are stored within the object's data set as REC_ADRs. This pointer structure means that networked or indirectly defined disk files may be loaded into a memory region and accessed directly rather than as a virtual disk.

A functionality required by (but by no means limited in application to) engineering applications is the ability of the data base server itself to carry out processing of database contents in a manner which can be influenced by the data itself. An example is the derivation of a current data set from one or more data sets which may themselves be indirectly defined. Such processing may be a function of both the data type found and the current processing environment. This allows, for example, a data set defining a structural element to determine at different times whether it is to be designed, drawn, or specified, and by which process.

The usual way in which the processing mode may be specified is by allocating one of a number of available "classes" to each object of a particular genre. The "class" is stored in the record header and is used to determine the processing required to evaluate the current data set to be returned to a higher level object when requested. Other ways of determining processing mode may be defined as required.

This functionality could have been provided more easily in a LISP or SIMULA environment. However it was felt that the time had not yet arrived when a LISP environment will live comfortably with the types of

mixed mode processing likely to be encountered in a general engineering office system in a small office.

However it is planned that a subset of "LISP", probably written in "C", will form part of a future intelligent user interface.

10.0 OFFICE ENVIRONMENT

The office in which the development is taking place is predominantly engaged in civil-structural engineering consultancy activities with computer system development occupying a secondary role. This results in some inefficiency in the development process, but means that the development work is very much end-user driven. As prototypes are developed they are immediately put into active service. Some have been running this way for 4 years before anyone had a chance to get back and clean them up. Despite the disadvantages, this means that developed modules are immediately placed into field test in house, and thus have a fighting chance of evolving into stable, reliable products.

10.0 CURRENT STATUS

At present priority is being given to definition and planning of the database server modules and further development of existing utilities and utilities in development has been deferred until this stage has been completed. Existing office utilities in other languages will then progressively be converted to "C" and to the new database standards. Once again the priorities will be set by office work flow. However it is expected that there will be a number of systems working in the new environment within six months.

11.0 CONCLUSION

DECUS-C is being used intensively to develop a broadly based applications system covering areas from specialised engineering to general purpose applications. While there is a need for continuing development and optimisation, of the compiler and its utilities, particularly in its interaction with the RSX operating system, the language has proved to be convenient and effective in this development program.

DEVELOPING LARGE PROGRAMS ON THE PDP THE SPAWN PROCESS

BY WALTER HAYES
IIT RESEARCH INSTITUTE
ANNAPOLIS; MARYLAND

ABSTRACT

Programs can sometimes grow so large that they exceed the 16 bit 32K word allowable program space. When this happens, the programmer often turns to overlay and segmentation techniques to reduce the program size. Another technique, spawning, lets the programmer divide the program into separate portions which can be called by each other, enabling the program portions to be swapped in and executed as needed. A description of the spawn process, programming examples and reasons for its use are provided in this paper.

You've just completed a study of your new project's requirements and suddenly realize that your system's memory is much too small to maintain the huge program you need. What next? Or, suppose you need to simplify the operation of several programs by linking them together or by combining them into one program. How would you do it? You could buy a larger computer (an expensive proposition); you could separate the project requirements into smaller more manageable programs; you could overlay the subroutines (they may still be too big); or you could spawn offspring tasks. Spawning overlaid programs, a combination of the last two alternatives, appears to be the best alternative because the results provide a very workable and easily maintainable program. The best part about spawning is that it lets your program grow way beyond the memory size limitations of your system.

In our case, we spent two years developing two large prototype programs that were each overlaid, but we were still pressing the upper limits of our PDP 11/23 memory. The PDP 11/23 has 124K words total memory available, but, if virtual arrays are not used, only 32K words are available for one program. The rest is used for overlaying, multiple user tasks and system routines.

The project's objective was to allow automatic and manual creation of a full color graphic picture. One program displayed a color background picture and then allowed user placement of predetermined shapes. The other program redrew the background picture, with the additional shapes, and then let the user draw freeform designs using a joystick. The user could select any color for filling in the shapes. Each program created a different alphanumeric display and updated different files.

Our project requirements were changing. We needed to combine both prototype programs into one model for easier execution, portability, and to gain better control of the shared data files. Combined, they totalled 64.2K words, much too large for one program. Fortunately, there was one major module common to both programs, which saved some space, but we still had to figure a way to fit the rest of the program into available memory. We used the Overlay Description Language (ODL) to overlay logically independent program segments to save more memory, even though both programs were already extensively overlaid. Any reconfiguration steps taken must conserve as much memory space as possible because future additions were inevitable. Thus, it was decided to use the spawning process available under the RSX-11M operating system because it allows the most flexibility.

SPAWNING DEFINED

Spawning is the ability of a program (called the parent task) to call, activate and execute another program (called the offspring task). Spawning sets up communications between the offspring and the parent. The parent task can either be put on hold at any time to wait for the offspring to finish an execution, or it may run concurrently with the offspring. To control shared access of data it is advisable to hold the parent task's execution. Concurrent processing can be a time saver because two separate functions are being completed simultaneously. An offspring can also spawn another task, thus becoming a subparent that can wait for or run concurrently with its spawned offspring.

Spawning may occur by one of two methods. The most effective method is to have the command line interpreter (CLI) execute the offspring task. In this method, the CLI (for example, Monitor Console Routine (MCR)) is spawned by the

parent. The name of the offspring task is passed to MCR to execute. In the second method, the offspring task can be directly spawned by the parent without using the CLI. Greater control and better success may result when using the CLI, however.

Communications are set up between all connected parent and offspring tasks so that information about the status of the offspring can be returned to the parent and any other connected tasks. The status is transmitted by status block to the parent when the offspring terminates or when requested by program code. Termination of the offspring run is a significant event and triggers the parent back into action. At the same time, the status block informs the parent whether the offspring execution was successful.

The spawn process is described in the Digital Equipment Corporation (DEC) DIGITAL SOFTWARE RSX-11M EXECUTIVE REFERENCE MANUAL. The process is performed by using the DEC-supplied SPWN\$ (MACRO) or SPAWN (FORTRAN) system directives.

GENERAL PROCEDURE

The following paragraphs describe a procedure to follow when spawning tasks.

Top-level structural design is an important first step to take. Determine how many basic functions exist in all the programs you wish to incorporate. Most likely, several modules will be repeated in these different programs. Highlight the major functions. Form the tasks by grouping functionally cohesive modules that perform separate activities. Set up an overall structural design and task hierarchy to help define the communication connections. Proper design is crucial, so exercise great care in this phase. The structural design will help you determine which task will be the parent and just where control will be transferred to the offspring.

When the top-level design is completed, organize the programs' code by task. Use ODL to keep their size below your system's memory limits. Since each spawned task will actually be a separate program, you will find that many modules will be used over and over, so put all modules into programmer defined libraries for each access.

Several DEC-supplied system directives are available that make the spawn process easy to accomplish. The directives are SPAWN, GETMCR, and WAITFR. SPAWN requests the CLI to execute the offspring. The CLI requires a task instruction array of up to 79 characters for the RSX-11M system or up to 255 characters on the RSX-11M-PLUS systems. This array will contain the name of the installed offspring task and any other data you

wish to pass to that spawned task. For example, assume the offspring task name is OSP and it will perform calculations based on data derived from the parent task. The characters OSP and the data can be sent in one array.

GETMCR, called by the spawned task, retrieves an 80 character MCR command line for the tasks internal use. This may contain data passed from the parent to the offspring.

WAITFR, called by the parent task, stops execution until the specified event flag is set by the exit of the spawned task.

Any tasks to be spawned must first be installed on the system. To do this, use the 'TASK=task-name' option when task-building the spawned task. Then, prior to running the spawning program, install the task-name on the system using the INS (install) command.

PROGRAM CODING

The accompanying program listing, Figure 1, is presented to illustrate the coding required to spawn a task. It will be referred to in the following discussion. The program in this example is designed to set up an array, CMD, and send it to a spawned task, OSP. Comments will be directed to the line numbers in the right hand column.

```

PROGRAM PARENT
C
C THIS IS A TEST PROGRAM DESIGNED TO ILLUSTRATE
C THE SPAWN PROCESS.
C
C INTEGER DSW,TLEN,MCR(2),EFLG          1
C CHARACTER CMD(79),CHARJ              2
C DATA CMD(1)/'0',CMD(2)/'S',CMD(3)/'P'/ 3
C DATA MCR/3RMCR,3R.../                4
C
C TLEN=79                                5
C DO 100 J=1,7                            6
C ENCODE(1,110,CHARJ)J                  7
100 CMD(J+3)=CHARJ                        8
C
C EFLG=2                                  9
C CALL SPAWN(MCR,,EFLG,,IESB,,CMD,TLEN,,DSW) 10
C IF(DSW.GE.0) GOTO 300                  11
C TYPE *,'ERROR SPAWNING MCR...'        12
300 CALL WAITFR(EFLG)                    13
C IF(IESB.NE.1) TYPE *,'OSP ERROR NUMBER=',IESB 14
110 TYPE *,'TSPAWN REACTIVATED'         15
C FORMAT(I1)
C STOP
C END
C
C PROGRAM OFSPRG                          16
C THIS IS THE OFFSPRING SPAWNED TASK
C INSTALLED TASK NAME=...OSP
C CHARACTER CMD(80),DATA(7)              17
C
C CALL GETMCR(CMD)
C DO 100 I=1,7                            18
100 DATA(I)=CMD(I+3)                    19
C                                         20
C BEGIN PROCESSING DATA HERE
C .
C .
C .
C STOP
C END

```

Figure 1.

Line 3 places the installed offspring task name, 'OSP', into the first three elements of the array, CMD. Line 4 sets up the MCR command line interpreter letters (...MCR) into the required Radix-50 format. Lines 5, 6, 7 and 8 define the length of the array, CMD, and assign the numbers 1 through 7 to the array as shown. The ENCODE statement in line 7 transforms the integer value, J, into a character value, CHARJ, for assignment to the character array CMD. These numbers represent any data that could be passed to the spawned task for its use. The array, CMD, now contains the characters 'OSP1234567' in the first ten elements' the remaining 69 elements are not used.

Line 9 assigns an event flag number, EFLG, to be set when the offspring sends any status information or exits. According to the Executive Reference Manual, there are a total of ninety-six event flags available. Local event flags (1-32) are unique to individual tasks and are set or cleared by that task. Common flags (33-64) can be set or cleared by any task. Group global flags (65-96) can be set or cleared by any task running in a group User Identification Code (UIC). Local flags 25-32 and common flags 57-64 are reserved for system use.

There are 12 parameters in the SPAWN directive (line 10), but they are not all necessary for execution and can be skipped by inserting a comma as a place holder. The SPAWN calls the MCR command line interpreter with arguments to be used by the Executive to issue a system directive to run the offspring task OSP. The first parameter is the name of the task (in this case MCR) to be run. The fourth parameter is the event flag number common to both the parent and the offspring. The offspring status will be put in IESB, an eight-word status block, which defaults to a one word integer. Typical status values will be: 0 for a task completion with a warning, 1 for a successful completion, 2 for an error causing unexpected results, and 4 if a severe, usually fatal, error occurs. This value should be checked, and if there is an error, the status error code should be relayed to the operator for his action. The CMD array and its length, TLEN, parameters 8 and 9, are both sent to the spawned task. The Directive Status Word (DSW), parameter 12, is an integer that receives the status of the MCR execution. Generally, except in specialized cases not addressed in this article, a +1 is returned by DSW if the task was accepted and a negative value returned if the task was rejected. Checking the value of the DSW informs the operator of the exact MCR status (line 11). If the MCR spawn failed, a message will be sent to warn the operator (line 12). At this point, further testing is required to determine the nature of the error and to allow the operator a chance to take necessary action, i.e., installing a task and then looping back to try the SPAWN call again.

If the spawned task is accepted, it will run concurrently with the parent. In this case, however we want the parent task to stop and wait for the offspring to finish. This will help us control the shared data. Two routines accomplish this purpose, STOPFR(EFLG) and WAITFR(EFLG), where EFLG is an event flag number (line 13). These routines stop or block the parent task until the specified event flag number (in this case EFLG=2) is reset by the offspring. The Executive sets the event flag to zero when the task is accepted and resets it to one when the task terminates. If there is an abnormal exit, the DSW code is also recorded. The parent task must wait for the offspring to set the event flag. If the flag is zero, the offspring has not terminated nor been rejected. The parent task will continue execution after the event flag is reset.

If an error occurs within the offspring task OSP, the status block value, IESB, can be typed out to inform the operator (line 14).

The offspring task code must include a way to receive and accept the MCR command line and data we wish to send it. The FORTRAN routine GETMCR(CMD), where CMD is an 80-character array containing the installed task name and any data, is the system routine that is used (line 18). Activities included on lines 19 and 20 unravel the data passed to the offspring for its use. When the task execution is complete, however, no data is transferred back to the parent in the CMD array.

Any data calculated by the parent task and required in the offspring that cannot be sent in the CMD array may be transferred by files as long as they are closed prior to each spawn and then reopened in the offspring. The files must again be closed prior to the offspring termination, in preparation for the jump back to the parent task.

ADVANTAGES

There are distinct advantages to using the spawn process. The most obvious being that the program can grow many times beyond the size limitations of the system. Countless tasks can be developed and called by the parent or offspring, each functioning as a separate program limited only by the computer's physical memory size. The net result is that the main program runs with no operator intervention needed to execute any of the spawned programs.

Debugging is made much easier. When errors are discovered and traced to a task, only that task need be changed, recompiled, and task-built. The time savings resulting from task-building only one task versus the entire program is significant. If the parent is designed such that the offsprings are called by the operator through keyboard options, one may remove the

affected offspring task, repair the code, recompile, task-build and reinstall without terminating the parent task. In this way, the entire testing phase can be completed at a quicker pace.

Spawning stresses breaking a program into major modules. This approach is similar to the structured programming techniques we all hear so much about.

Before you attempt to spawn your own programs, here are a couple of useful hints. It is advisable to make all the tasks checkpointable by including the '/CP' switch on the task image file when task building. Parent tasks waiting for the offspring to terminate can then be checkpointed out, thus conserving pool space. On the PDP 11/23, pool space is limited and can be easily exhausted. Another memory saving technique, using the '/-TR' option with the source code file when compiling, eliminates the trace capabilities but frees up memory for a larger program.

CONCLUSION

Spawning is a very useful programming tool with many advantages and a wide variety of applications. Spawning is a relatively easy process to understand and implement. Countless tasks can be broken into separate programs to be spawned from the main program. This allows the overall program size to grow way beyond the limits of the system's memory. No operator intervention is required to run any of the small spawned tasks. Although this article does not completely investigate the concurrent processing technique, it can easily be achieved using the spawn process.

Without spawning, our project would have been very difficult, if not impossible, to complete on our minicomputer. Now that we understand the programming steps involved in spawning tasks, we often find ourselves dreaming and scheming of new uses to further enhance our graphic display model.

DEVELOPMENT OF A COMPUTER-BASED DATA ACQUISITION SYSTEM

S.K.R.Iyengar and R.P.Schmidt
John Deere Product Engineering Center
Waterloo, Iowa

ABSTRACT

This paper describes the development of a data acquisition system based on the DEC Micro-PDP 11 computer using the RSX 11M operating system for collecting and analyzing data on hydraulic components.

The first part describes the formulation of functional specifications and the attempts to apply data base management techniques to scientific and engineering applications, and in particular the use of RMS files to store a variety of engineering data.

The second part describes the menu-driven FORTRAN 77 programs and their development, installation and usage by engineers and technicians. A brief description of the hardware used, and the impact of both the hardware and operating system and software packages, such as FMS, on program development and usage is furnished.

The last part describes the performance of the RMS file structure in relation to original expectations, and the extent to which original functional specifications could be implemented.

INTRODUCTION

Historical Perspective

The John Deere Product Engineering Center is involved in the design and development of many hydraulic components used in agricultural, construction and industrial machinery.

Testing is an important part of the design cycle of such components, and typically involves installing the test component on a suitably designed test stand, subjecting it to prescribed input signals and measuring a variety of physical parameters at various locations in the test arrangement. Typical measurands are rotational speed, torque, temperatures, pressures and flow rates. Electrical transducers generating dc voltage signals which are linear or nonlinear functions of the measurands are commonly used. Such voltages can be fed into analog / digital readouts, recording equipment such as strip-chart recorders, X-Y plotters, and optionally, transmitted to a suitably equipped computer system.

A particular advantage of a computer - based system is that for steady state tests, a number of calculated quantities, such as power consumption or dissipation, and efficiency, can be calculated in real time and displayed at the test station. Even for dynamic tests, where the need for a high sampling rate precludes calculation of secondary quantities in real time, it is

possible to print or graph them soon after a test. Hence, a computer-based system reduces the turn-around time for tests, especially those of a developmental nature. There are many other advantages which we need not itemize here.

Earlier testing at PEC relied on a centralized system in which five hydraulic test stands had been connected to a mini-computer running a real-time operating system, and were assigned a set of channels of the analog to digital (A/D) converter. Each test stand had its own control keypad, but shared some peripherals like graphics terminal and hard copier and line printer with others. Since this system hardware had outlived its usefulness, and was detracting from the productivity of the test stands and operators, it was decided to replace it by a decentralized arrangement, with one computer serving, at most two test stands.

However, the software was considered mature and reliable, and as much of it as possible was to be transferred to the new systems. Wherever new technology offered the potential for improving the functionality of the data acquisition system, we decided to use it.

The next section describes the development of functional specifications, which was a major part of the project.

FUNCTIONAL SPECIFICATIONS

These were developed in two phases: first from the user's perspective and next from the system designer's perspective. We attempted to draft the first so as to describe what the system would appear to be doing, in terms used by test engineers and operators. Since the system was not expected to be completely automatic, human interactions by both operators and test engineers had to be spelled out in general terms. This way we would not constrain ourselves too early to any specific hardware / software and their idiosyncrasies.

The objectives in terms of product development were described for performance tests on each class of test components, and these were translated into the terminology of data acquisition, processing, display and storage.

Brief descriptions of testing procedures for a wide variety of tests were also written down. Since these were product - specific, it was necessary to analyze them and extract the common features, and combine them in a generic specification for all the data acquisition systems.

Requirements more specific to a product could then be 'layered' on the 'core' requirements, provided it had the necessary 'locking' features.

For example:

- Analog channel requirements for each class of test components were analyzed to arrive at a channel count meeting needs for most, if not all, testing.
- Data processing was divided into real - time and non - real - time activities; the latter was further subdivided into quasi - real - time and post - processing, depending on the criticality of the turn-around time. The task of deciding which tasks would be in the first sub-category and which in the second, was postponed until the computer system was decided upon.
- Annotation was considered very important in helping test and design engineers manage tests and data. Once again the needs for various classes of test components were analyzed and common features extracted to impose a minimal structure on the annotation. Little thought was given at this stage, deliberately, to implementation considerations. Length of annotations was suggested but kept open for further negotiations.
- Software requirements were outlined in general terms, without specifying which features were to be provided by the Operating System, which by purchased software, and which by in-house development; to the end-user, they would be essentially indistinguishable. However, some end-users would have the capability of developing program modules.

It was envisaged at this early stage itself that users would span the spectrum from test operators - to whom the system would appear to be embedded in the test stand, to engineers - to whom it would appear as one of many available general purpose computers for post - processing, to systems analysts and programmers - to whom it would be development tool over and beyond its data acquisition capabilities.

The second stage of functional specification preparation involved preparing a separate document for each

system, outlining input and output data, data processing needs in real - time and pseudo - real - time, data management, in addition to channel requirements and test setup and data acquisition needs.

Even though the systems had much in common, certain individual features were considered very important and needed to be spelled out in some detail so as to ensure that operators and test engineers were not required to use a number and variety of keystrokes to perform commonly used manipulations and operations.

Even though the operating system and utilities were not specified, language capabilities, needs for hierarchical storage of data and help features were described in general terms.

Examples of typical graphs and reports to be generated at the test station were also included in this document.

Based on the above two documents, it was concluded that, in broad, general terms, the system had to do the following:

- Maintain a periodically refreshed display of measurands and secondary quantities, thus emulating a set of digital panel meters.
- Smooth the quantities displayed by collecting a prescribed number of samples at a prescribed sampling rate and averaging. This was termed a 'segment-average'.
- In response to an operator input, store a set of values for the above quantities, corresponding to an instant of time on a permanent storage device.
- Print a completely annotated test report.
- Plot one or more standard graphs.
- Store test data and annotation in an easily retrievable fashion, without compromising integrity.
- Answer queries for test management, for a limited range of test numbers.
- Permit test engineers to do a certain amount of post processing, i.e. use the system as a general purpose computer.

The data acquisition, display and report generation activities had to be menu driven.

The keyboard had to be used as the sole device for the test operator interactions. This was considered desirable for ergonomic reasons.

Other major considerations were as follows:

- Need to handle common transducers for pressure, flow, torque, speed, position and velocity measurements. It was decided that the computer would handle only analog signals for uniformity. We would convert frequency type signals from turbine flowmeters and tachometers using frequency - to - voltage converters.
- Test stand and testing should not be solely dependent on computer system. i.e. we should be able to read values from transducers thru panel meters, record on strip charts etc., even if the computer was not in use.
- Even though post-processing of test data would be done on a larger computer, the test stand computer would be capable of handling most needs for quick - turn-around analysis.

HARDWARE

Hardware selection was based as much on perceived notions of product maturity, and expected support as on a rigorous analysis of needs. Experience of sister organizations in using DEC and other computers were evaluated before final selection.

Computer

Fig. 1 shows a view of the system as finally implemented, using a Micro PDP-11 computer² which was chosen for multi-tasking multi-user support.

Peripherals and accessories were chosen as follows:

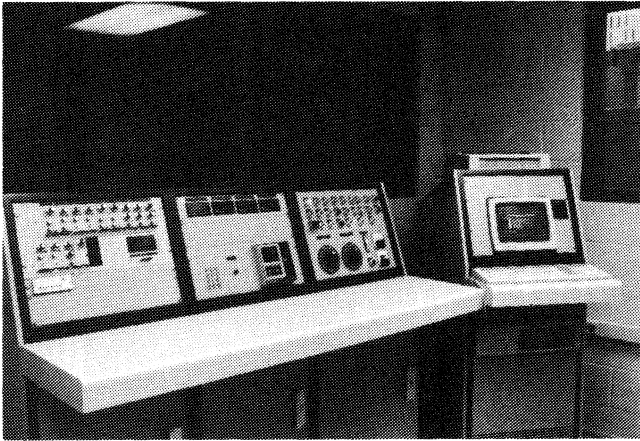


Figure 1. View of Data Acquisition System, showing arrangement

Peripherals

Storage A 10 Megabyte hard disk came with system. This was subsequently upgraded to 30 Megabyte capacity. The two 500 kbyte floppy disk drives which came with system were to be used for transferring data to and from other systems, off-line data processing, short-time archival. Networking was expected to be implemented later.

I/O Devices The VT240 terminal was chosen for two main reasons:

- TEKTRONIX emulation, so that graphics could be done at test stand²
- VT100 emulation so that FMS (Forms Management System) could be used

Choice of the LA50 printer as the hard copy device was a difficult decision to make, since the cost - benefit trade-offs of quality of reports and graphs and quick turnaround could not be quantified.

For handling analog signals from the test stand, an A/D Converter is necessary - Data Translation's 12-bit converter 2752 and clock 2769 were selected. The conversion rate of 50,000 samples per second seemed adequate for all tests, except those investigating high frequency transients. In any case, software calculations would limit the frequency of screen updates.

Purchased Software

This includes the following:

Operating System The choice of RSX-11 M over RT-11 was dictated by a perceived desire for multi-user support, and prior experience with program development on an RSX-11 M system (though not one used for real-time operations). RSX-11 M was chosen over Micro-RSX because Data Translation software package RSX-LIB needed it.

Utilities The RMS indexed file system was chosen, the next section goes into the reasons in more detail. A spreadsheet program which would handle matrices of real numbers was obtained. FORTRAN 77 of course, was needed for programming. RUNOFF was expected to be used for most text processing. FMS was chosen for menu development, though we were not sure how it could be integrated with the rest of the software, especially in a real-time environment.

We expected to use QIO's for screen management during real-time display, but found that a DECUS program UVT100 written for use with a FORTRAN spreadsheet (PORTACALC) could be adapted easily.

Data Translation's software package RSXLIB and sections of diagnostic program code served as the bridge between the A/D converter and application programs.

File Specifications

Once the decision had been made to use the DEC Micro PDP-11 computer and the RSX-11M Operating System, and FORTRAN 77 as the programming language, the next step was to layout the program modules, using the existing programs (written in FORTRAN IV), as a guide. It was felt, however, that the capabilities of the Operating System and FORTRAN 77 would be exploited more fully by using a structured approach and the RMS file structure. It was also felt that this would improve readability and maintainability of the programs.

Since retrievability of test data and test attributes in a machine readable format was considered highly desirable, it was decided to analyze the requirements for storage and retrieval using bubble charts.

It was found that scope of future testing activities could not be described in the meticulous detail used in designing business systems, and the compromise was between an unwieldy filing system designed to cover all types of tests versus a modest one covering perhaps 80% of test activity.

Retrievability and query needs for test data are somewhat different from those for business systems. Typically querying of raw test data is unnecessary. However, pointers to the location of the test data have to be synthesized from queries. For example, it may be necessary to retrieve all the raw data gathered during one or more tests, given one or more of the following:

- a unique test number
- a unique test component identification
- a test stand number
- a range for test date and time

This data should be made available for printing or processing, including graphical display by a variety of programs.

Similarly, under different circumstances, it may be necessary to aggregate test data on a specific component, possibly spanning a number of tests between given dates, and feed it to a statistics or graphing program.

Fig. 2 depicts schematically the relationship envisaged among a number of quantities relating to test data. It is seen that a vector representation is applicable to most quantities. Proceeding from the left, we move from 'raw' data to processed data. The relationship between one type and another could be one to one or one to many, as shown by the connecting lines. Time is considered a special kind of measurand, in that it is computed directly by the computer. But for this exception 'measurands' forms a subset of 'hardware sensor channels'. 'Run parameters', 'run variables' and 'test parameters' are pertinent to test management, since they enable the storage and retrieval of test data in a hierarchical fashion. Certain common kinds of processing, especially graphing, sorting and statistical analysis are facilitated by this hierarchical structure. Limitation to levels in the hierarchy to 2, i.e., test and run, was based on intuition and experience.

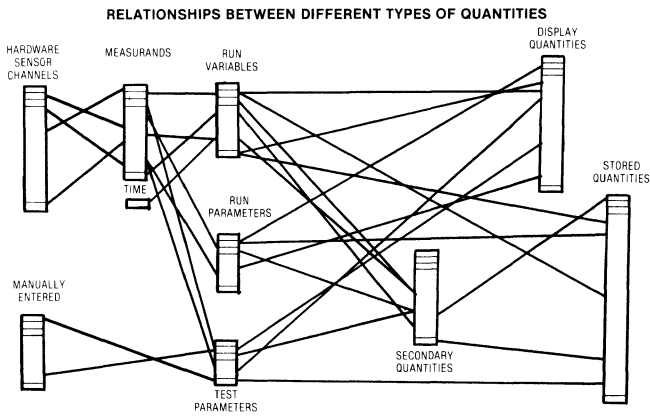


Figure 2. Relationship between Various Quantities

The category marked 'manually entered quantity' was provided to allow keyboard entry of data for transducers such as dial gages, hygrometers etc., not having electrical signals. The quantity marked 'secondary quantities' was provided to cover software channel calculations mentioned earlier. Note that the 'displayed' and 'stored quantities' are subsets of almost all the other quantities.

Data definition plays a comparatively minor role in scientific computation, but the adoption of a systematic technique was considered desirable since we intended to use RMS file management, and if possible and desirable, a data base management system at a later date.

The bubble chart was chosen as the medium for displaying relationships between different quantities, since it appeared to be adequate and appropriate as well as simple to explain to novice users of data bases.

We found that the bubble chart was primarily useful in deciding the file structure and contents of log files and test management files, rather than test data files. In fact the decision to maintain different kinds of files for test stand and test management rather than store everything in one or more (unformatted) direct access files, as had been done in the past, was motivated by the capability of data base systems to read properly structured files and permit ad hoc queries.

The bubble chart uses a single arrow to indicate a one - to - one relationship, and a double arrow to indicate one - to - many relationship. Thus a test id points to a number of run id's, and a run consists of a number of sample points. Each sample is comprised of a set of measurand and (calculated) secondary quantities. Data from individual sample points is rarely, if ever asked for, and no provisions for such retrieval were considered necessary.

Fig. 3 depicts the bubble chart drawn for storing test data. Once the structure of test data was established to be somewhat similar to that of business data, it was felt that the use of a data base population and query language was worth exploring. After trying out DATATRIEVE on some test data, we felt that a relational data base would be an overkill for storing run data. RSX-11M DATATRIEVE was not found quite suitable for handling floating point numbers and scientific notation. Also, the release we used had no graphics capabilities and the FORTRAN bridge did not look easy to use either. Even for test management data, the variety of queries could be assessed fairly accurately and ad hoc queries would not arise too often. However, by storing the test management data in indexed RMS files, it was felt that migration to a relational data base, as and when indicated by the volume of ad hoc queries, would be facilitated.

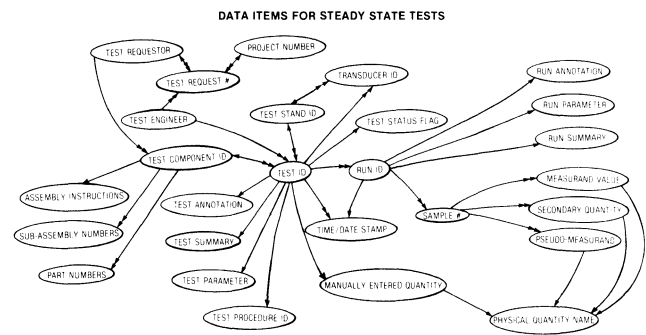


Figure 3. Bubble Chart showing data times for tests

Once we decided on using an hierarchical structure for storing test data, we had to establish a terminology for the most commonly used entities. Appendix C is a glossary containing the more important terms.

SOFTWARE DEVELOPMENT

We decided on using a resident common to communicate between a number of tasks. It would contain labelled COMMON blocks used by all programs operating in real time, and some which did not.

It was evident that a task to read the hardware channels and deposit the numbers in part of the resident common area was to be core task. Prior to running this task, other tasks would have to set up the data acquisition process and log the test. This core task would stop on receiving a suitable keyboard command. Additional tasks, which would have to run concurrently, would be needed to convert the raw data to engineering values for all channels, display them on the monitor, and process keyboard interrupts. When a run or test was completed, additional tasks would have to process user annotation and store them in the appropriate files.

Overview

Task PUMP30 is the main menu driver which displays the menu shown in Fig. 4. On booting, it is invoked by the startup command file, which also installs the resident common for inter-task communication of numbers and character strings.

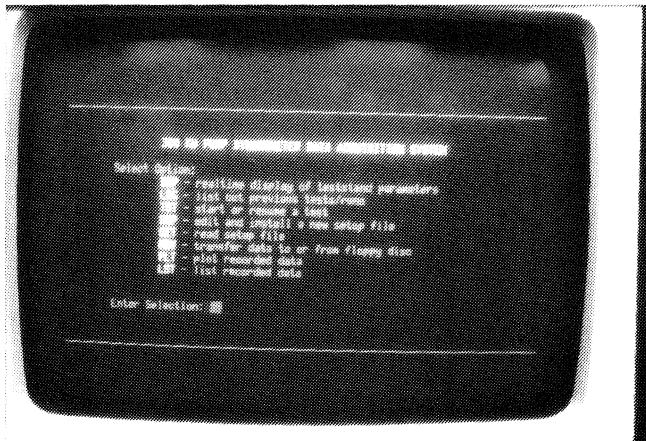


Figure 4. Main Menu

It spawns task TEST2, which controls data acquisition. Task TEST2, in turn spawns a series of tasks to obtain setup information. Fig. 5 is a timing chart of our own design, which depicts the use of various flags and indicates which tasks are concurrent.

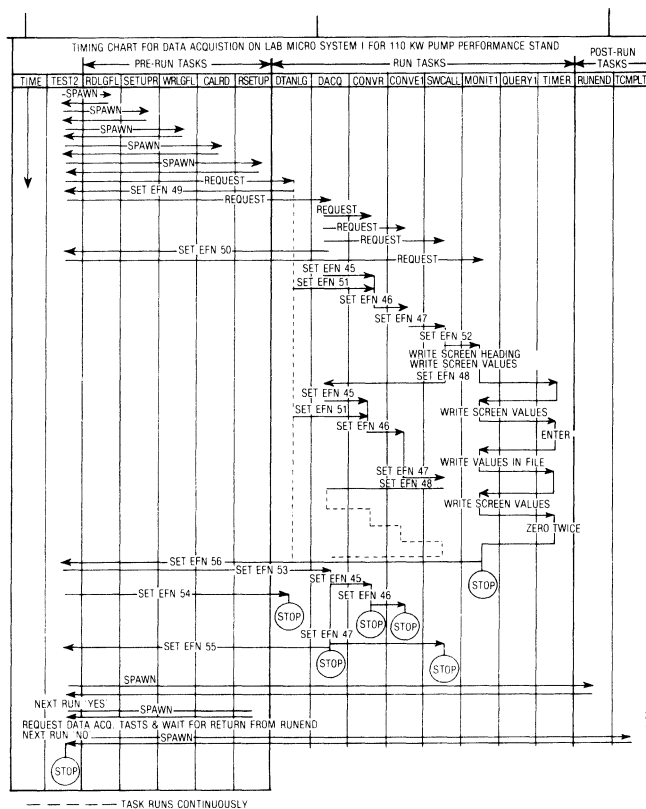


Figure 5. Timing Chart

Use of FMS to manage dialog forced us to split setup between 3 tasks. Subsequently TEST2 spawns a task to read the calibration information on pertinent transducers. Now all necessary setup and calibration information is available in the resident common. TEST2 now spawns task DTANLG, which sets up parameters for the Data Translation programs and starts reading the A/D converter output buffers.

TEST2 next spawns tasks to convert the raw numbers to engineering values for hardware and software channels. Flags are used to ensure that each of the tasks waits for the completion of the previous one, and DTANLG cannot go through the next acquisition before all hardware and software calculations are completed.

Task MONIT1 is also spawned by TEST2 and is used to refresh the screen display and wait for a keyboard action. If no action is forthcoming in 3 seconds it repeats the refresh and wait cycle.

MONIT1 uses a modification of the DECUS program UVT100 to display channel names, units and values. It cannot recognize a keyboard interrupt while writing values, since the keyboard cannot be separately addressed from the screen. Hence the need for a 'wait window'.

Fig. 6 shows a typical display. The title, hardware and software channel names and units are written only once. The values are refreshed every 3 seconds. MONIT1 also contains a reference to a MACRO11 program (see appendix B) to recognize keyboard interrupts. Specific keys on the keypad result in branching to different sections of code in MONIT1.

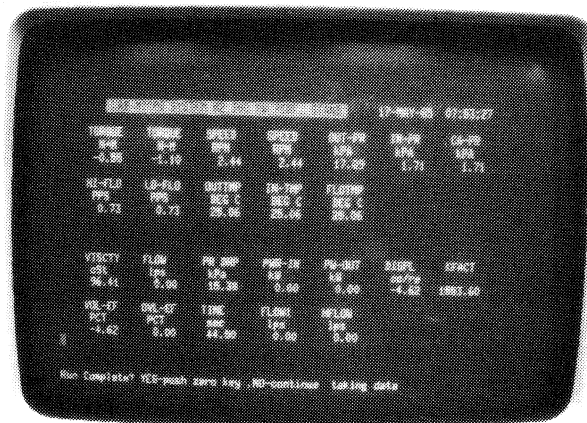


Figure 6. Typical Display

Keyboard interrupts trigger sections of code in TEST2 to stop DTANLG, and other tasks dealing with conversion of channel readings. Additional tasks are spawned to process user annotation for completion of run and test.

Pre-Test Programs

These included:

Test Setup Programs

- to generate new test setup files
- to update old test setup files

Test Setup Reader Program To display / print the following:

- channel gains
- transducer ID's for measurands
- software channel names and units
- upper/lower limits of channels (not used)

Calibration Programs to:

- to enter new transducer calibration values
- to update old transducer with new calibration values
- print Calibration data on specified transducers, or dump entire file

Post-Test Programs

These included:

Report Generating Programs Two types of reports can be generated at the test stand.

- a full report of all parameters
- a selected listing of certain parameters

Both reports are generated by picking the appropriate menu option. The user then enters the desired test number and run number. There is an option to have statistics calculated for each of the channels (software and hardware), also. The report is then printed out at the test stand printer.

Fig. 7 and 8 show a typical printed report. The extraction of information from log, annotation and attribute files should be noted. The report generation programs turned out to be among the longest in the system, and users are still developing variants.

```

LAB MICRO SYSTEM -- 300 kW Dynamometer
Test Stand ID : 240 Test Number : 1325
SAMPLE RATE/AVERAGING CHECKOUT OF LAB COMPUTER #2
Test Part No. :RA 92700
No. of points in Segment Average: 100 Sampling Frequency: 100. samples/sec.
Th. Disp. 63.00 cc/rev Rot.: F run on : 14-MAY-86 07:58:35 by : RPS

Run Number 11 of 11 runs :
10 MS, 100 POINTS
Run Date and Time : 14-MAY-86 08:01:18 Run Completion Code : S
Run Temperature: 38.0 Run Parameter : #4
Number of points in run : 10
    
```

Hardware Channels

	TORQUE N-M 1	BY-FLO L/S 2	SPEED RPM 3	CC-TMP DEG C 4	OUT-PR KPA 5	IN-PR KPA 6	CA-PR KPA 7
1	114.664	5.094	998.533	56.689	10032.74	97.634	30.301
2	114.664	5.252	1000.974	56.689	10032.74	97.634	30.301
3	114.664	4.909	1000.974	56.689	10032.74	97.634	20.199
4	114.664	4.916	998.533	56.689	10032.74	97.634	30.301
5	114.664	5.255	998.533	56.689	10032.74	97.634	30.301
6	114.664	5.255	998.533	56.689	10032.74	97.634	30.301
7	114.664	5.094	998.533	56.689	10032.74	97.634	30.301
8	114.664	5.118	1000.974	56.689	10032.74	97.634	30.301
9	114.112	4.842	998.533	56.689	10032.74	97.634	30.301
10	114.664	5.245	1000.974	56.689	10032.74	97.634	30.301

	HI-FLO PPS 8	LO-FLO PPS 9	OUTTMP DEG C 10	IN-TMP DEG C 11	FLOTTIP DEG C 12	LOADPR KPA 13	ROD-PR KPA 14
1	174.316	54.931	40.137	38.550	43.55	9696.068	7187.480
2	174.316	54.931	40.137	38.550	43.55	9712.902	7187.480
3	174.316	54.931	40.137	38.550	43.55	9712.902	7187.480
4	173.584	54.199	40.198	38.550	43.55	9712.902	7187.480
5	174.316	55.664	40.198	38.550	43.62	9712.902	7187.480
6	174.316	54.931	40.259	38.550	43.62	9696.068	7187.480
7	173.584	55.664	40.198	38.550	43.62	9712.902	7187.480
8	175.048	55.664	40.137	38.550	43.62	9712.902	7187.480
9	175.048	54.199	40.198	38.550	43.62	9696.068	7187.480
10	174.316	54.931	40.198	38.550	43.62	9712.902	7187.480

Figure 7. Typical Report

Graph Generating Program A standard graph can be generated for each set of recorded data. A standard graph is a predefined plot of two test parameters (measured or calculated) for a specific hydraulic component. Fig. 9 is an example.

Software Channels

	VISCTY cSt 1	FLOW lps 2	PR DRP kPa 3	PWR-IN kW 4	PW-OUT kW 5	DISPL cc/rev 6	KFACT 7
1	40.030	1.029	9935.104	11.990	10.223	61.832	0.000
2	40.030	1.029	9935.104	12.019	10.223	61.681	0.000
3	40.030	1.029	9935.104	12.019	10.223	61.681	0.000
4	40.030	1.025	9935.104	11.990	10.181	61.577	0.000
5	39.929	1.029	9935.104	11.990	10.223	61.829	0.000
6	39.929	1.029	9935.104	11.990	10.223	61.829	0.000
7	39.929	1.025	9935.104	11.990	10.181	61.829	0.000
8	39.929	1.033	9935.104	12.019	10.265	61.933	0.000
9	39.929	1.033	9935.104	11.932	10.265	62.084	0.000
10	39.929	1.029	9935.104	11.990	10.223	61.829	0.000

	VOL-EF PCT 8	OVL-EF PCT 9	TIME sec 10	FLOW1 lps 11	HFLOW lps 12	DELTAP kPa 13
1	98.146	85.267	9.449	-0.079	1.029	0.000
2	97.907	85.059	18.533	0.079	1.029	0.000
3	97.907	85.059	25.883	0.079	1.029	0.000
4	97.741	84.915	34.549	0.078	1.025	0.000
5	98.142	85.263	43.349	0.080	1.029	0.000
6	98.142	85.263	51.699	0.079	1.029	0.000
7	98.142	85.675	59.499	0.080	1.025	0.000
8	98.306	85.406	68.266	0.080	1.033	0.000
9	98.547	86.028	76.266	0.078	1.033	0.000
10	98.142	85.263	84.849	0.079	1.029	0.000

RUN COMMENT :OK

Report Generated on : 14-MAY-86 at 08:09:14

Figure 8. Typical Report (cont'd)

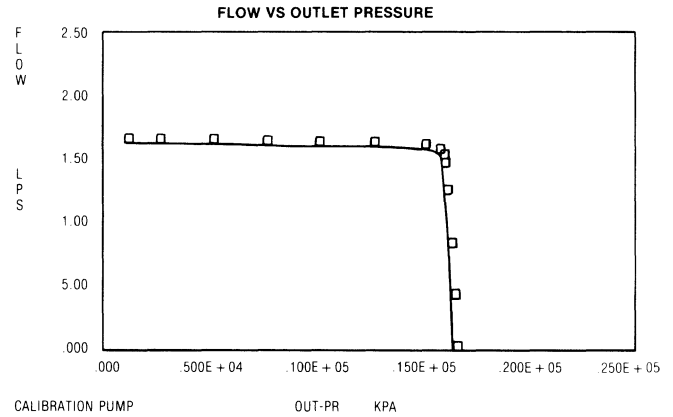


Figure 9. Typical Graph

There are two separate tasks that make up the graphing package. The first task asks for the test number and run number. The first task then obtains the data and attributes to be plotted from the RUN and RAT files. The information is stored in two files that the second portion of the plot program reads and plots from. These two files can be edited if erroneous data needs to be removed or if other test data needs to be merged into one file. The second task changes the terminal setup to TEKTRONIX mode and uses TEKTRONIX PLOT10³ commands to plot the graph.

The data acquisition system uses a DEC VT240 with plotting capabilities. A hard copy of the graph is made using the PRINT SCREEN key sending the plot to a LA50 printer.

TRAINING AND DOCUMENTATION

Common classes were held for test engineers and operators while the systems were being installed on the shop floor. Test personnel were given drafts of the User Guide and Reference Sheets for critique. Documentation was generated using well known DECUS package RUNOFF.

Technical Manual

This was written primarily for the System Manager and advanced users of the Lab Systems. Its contents are:

- Program (Task) Descriptions
- Timing Chart
- File Descriptions
- Event Flag Usage
- Dictionary of Variables

User's Guide

The User's Guide is meant primarily for the test engineer, though it is of use to the test operators after they are familiar with main menu options. Its contents are:

- Menu Usage
- Test Setup
- Task Descriptions
- File Contents

Operator's Reference Sheets

These sheets give in a cookbook fashion, the steps needed to do a cold start. It covers:

- Booting
- Keyboard Usage

PROGRAMMING STANDARDS

Rather informally, we used the following:

- FORTRAN 77 only, no DEC extensions
- Variable names in upper case, remainder in lower case
- Use of 'implicit character A-Z', and explicit specification of all variables
- Every file to contain an entry showing file name and revision date

PERFORMANCE OF RMS FILE SYSTEM

The following discussion reflects the experience of end-users in designing file structure, often without clear-cut initial specifications. We plunged into the design without any formal training or experience in file management.

RMS files were set up using the 'OPEN' command in FORTRAN 77. The description of file structures and access modes in the RMS 11 introduction (Vol. 4A of Micro/RXS Advanced Programmer's Kit) was adequate, though the examples did not have much relevance to the engineering data being stored.

Since this was the first time an Indexed multi-key structure was being used for some of the files, we were not too sure of the impact of many of our decisions.

Specifically, the effects of:

- length of records
- number of secondary keys
- bucket sizes and related parameters

were not considered during the design phase. No attempt to 'tune' the system was made.

RMS Utilities were found very easy to use, especially RMSDSP and RMSDES. However, due to problems in using files created by RMSDES in FORTRAN 77 programs, it was decided to create files from within the FORTRAN programs, using the 'OPEN' command. Our problems may have arisen due to switching programs,

RMS files between a system running RSX-11M and another running Micro-RSX. In particular, it was found that the RMSIFL utility on the Micro-RSX system could not compress files populated by the RSX-11M system. DEC indicated that differences in the prologue could be the reason for our problems.

In trying to exploit the capabilities of the RMS indexed file structure, and make our program easily readable, we made our records quite long. This posed problems in linking, which were resolved only by augmenting the ACTFIL, MAXBUF and UNITS parameters, by trial and error.

Error messages 37 ("Inconsistent Record Length"), 41 ("No Buffer Room"), and 30 ("Open Failure") were encountered frequently while trying various combinations of link parameter values.

Additionally, we found that task sizes were getting near the 32 k word limit for PDP 11 machines, and we couldn't always err on the side of safety.

In retrospect, we could have;

- used fewer secondary keys
- used shorter record lengths

to achieve efficient storage and retrieval and faster file I/O.

However, the readability of our programs would have suffered.

CONCLUSIONS

The systems have worked quite satisfactorily. Software problems did not arise during usage. Some test stand operators have used the SETUP features on the VT240 terminals to turn off and on the bell to suit their needs in different phases of testing.

The screen update of display was designed for a frequency of once in 3 seconds, but once in a while the shuffler intervenes to slow it down. Reasons are still being sought.

Our method of defining software channel calculations is very powerful but quite involved. Users get confused between the hardware channel setup, which describes the transducers being used, and conversion parameters, and the compilation of the program which performs software calculations. They expect the latter to be done 'automatically' when a setup has been invoked. Currently, they have to reboot the system to install the software calculation image.

Our attempts to involve potential users in designing the system as well as develop some of the software has proved quite successful. It is too early to tell how much use will be made of the flexibility afforded by the RMS filing system.

In using the VT240 for graphics, we have to send escape sequences to convert from Tektronix 4010 emulation mode to VT100 mode before reverting to VT240 mode. This is considered awkward, though of course, the end-user is not aware of these steps.

Summarizing our experience, the development of a Micro PDP11 based system running RSX-11M for data acquisition, resulted in a very powerful, yet flexible system, which has met the needs of test activities. Enhancements and modifications will make it suitable for an even wider variety of tests.

¹ See the Glossary for definition of special terms.

² Micro PDP-11, RSX-11M, FMS are registered trademarks of Digital Equipment Corporation. TEKTRONIX, PLOT10 are registered trademarks of Tektronix Inc.

³ PLOT10 is a registered trademark of Tektronix Inc.

APPENDIX A. FILE STRUCTURES

We will cover only the most significant data files.

A.1 TEST SETUP FILE

Type: Keyed, Indexed, Variable Length, Unformatted

Life: Permanent

Keys:

- Setup Descriptor
- Setup Date
- Setup Engineer

Other contents:

- Status Code
- Measurand Name
- Measurand Units
- Channel Number
- Transducer ID
- Gain
- Voltage to Frequency Conversion Factor
- Display Flag
- Software Channel Name
- Software Channel Units

```

PROGRAM TSTSW
C*****
C      PROGRAM TSTSW
C*****
C LAB Micro System # 2  C300 KW  Pump Stand|
C      FILE: DU0:C210,240|TSTSETUPW.PTN
C
C
C*****
C PRIMARY KEY TDESCR      XXXXXXXX
C SEC. KEY 1 STENGR      XXXXXXXX
C SEC. KEY 2 ENGR        XXX
C SEC. KEY 3 SETDAT      XXXXXXXX
C SEC. KEY 4 STCODE      XX
C*****
C      WRITE TO UNIT 1 <RMS INDEXED FILE 'DU0:C210,240|TSTSETUP.IDX'|
C*****
C Use this to create a new file:
C
C      open ( unit=1, file='DU0:C210,240|TSTSETUP.IDX', status='NEW',
C 1 access='keyed',form='unformatted', RECL=360,
C 2 organization='indexed',recordtype='variable',
C 3 key=(1:8,9:16,17:19,20:28,29:30),
C 4 initialsize=30, err=9999)
C*****
C      open ( unit=1, file='DU0:C210,241|TSTSETUP.IDX', status='OLD',
C 1 access='keyed',form='unformatted', RECL=360,
C 2 organization='indexed',recordtype='variable',
C 3 key=(1:8,9:16,17:19,20:28,29:30),
C 4 initialsize=30, err=9999)
C
C*****
2000 read(3, 1050) MEASNM(j), UNITS(j), CHNUM(j), TRID(J),
1      GAIN(J), VTOFCN(J), HCUT(j), LCUT(j),
2      INTFLG(j), DSPFLG(j), RTGFLG(j), STRFLG(j),
3      RZRFLG(j),SUMFLG(j)
C
1050 format(2a6,1X, i2,1X,A8,1X,4f10,3,1X,eal )
READ(3,1020) SWCHAN
do 2020 j= 1,swchan
2020 read(3,1150) SWNAM(j), SUNITS(j),SCHNUM(j),SHCUT(j),SLCUT(j),
1      SDPFLG(j), SSTFLG(j), SSMFLG(j)
1150 format(2a6,1X, I2, 1X,2F10,3,1X,3al)
write(1) TDESCR,STENGR, ENGR,
1 SETDAT, STCODE, SETTIM, NCHAN,
2 (MEASNM(J),UNITS(J), CHNUM(J), TRID(J),

```

```

2      GAIN(J), VTOFCN(J), HCUT(J), LCUT(J),
3      INTFLG(j), DSPFLG(j),RTGFLG(j), STRFLG(j),
4      RZRFLG(j), SUMFLG(j),j=1,NCHAN),
5      SWCHAN,(SWNAM(j), SUNITS(j),SCHNUM(j),SHCUT(j),SLCUT(j),
6      SDPFLG(j), SSTFLG(j), SSMFLG(j), J= 1,SWCHAN)

```

C
C
C
C

A.2 CALIBRATION FILE

Type: Keyed, Indexed, Variable Length, Unformatted

Life: Permanent

Keys:

- Transducer ID
- Calibration Date
- Entry Date

For turbine flowmeters:

Other contents:

- Universal Curve Polynomial Coefficients

For other transducers:

- Number of Points
- Interpolation Flag
- Lagrange Interpolation Order
- X-axis Units
- Y-axis Units
- X-axis Values
- Y-axis Values

```

C
C      open ( unit=1, file='DU0:C210,240|calstr.IDX', status='OLD',
C 1 access='keyed', form='unformatted', RECL=42,
C 2 organization='indexed',recordtype='variable',
C 3 key=(1:8,9:11,12:20,21:29),
C 4 initialsize=30, err=9999)
C
C      write(j) TRID, UIN TLS , CALDAT, REVDAT,
C      Write to RMS file
C 1 (A(j), j= 0, 5), INTFLG, NPTS, LNUM,
C 2 (XVALUE(j), YVALUE(j), J=1, NPTS)
C
C      write(1) TRID, UIN TLS , CALDAT, REVDAT, INTFLG, NPTS ,LNUM,
C 1 XUNITS, YUNITS,
C 2 (XVALUE(j), YVALUE(j), j= 1, NPTS)
C
C

```

A.3 TEST LOG FILE

Type: Keyed, Indexed, Variable Length, Unformatted

Life: Permanent

Keys:

- Test ID
- Test Date
- Test Part Number

Other contents:

- Test Title
- Test Engineer
- Test Fluid
- Test Component Attributes

C Open Test Log File

```

C
C      open ( unit=4, file=TSTLG , status='OLD',
C 1 access='keyed',form='unformatted', RECL=250,
C 2 organization='indexed',recordtype='variable',
C 3 key=(1:4:INTEGER, 5:13, 14:21, 22:38),
C 4 initialsize=30, err=9999)
C
C write incremented test number on first record of test log
C in variable N1STS
C
C*****

```

```

C
c Define setup file
c
STPPFIL = 'DU0:c210,241|TSTSETUP.IDX'
C
c Open setup file
c
open ( unit=1, file=STPPFIL , status='old',
1 access='keyed',form='unformatted', RECL=316,
2 organization='indexed',recordtype='variable',
3 key=(1:8,9:16,17:19,20:28,29:30),
4 initialsize=30, err=9999)
C
c Retrieve setup info from indexed file, using TDESCR as key
c
22999 Continue ! Come here from 79999 w default TDESCR
c
read(1,key=TDESCR,ERR=79999 ) TDESCR,STENGR, ZENGR,
1 SETDAT, STCODE, SETTIN, NCHAN,
2 (MEASNM(J),UNITS(J), CHNUM(J), TRID(J)),
2 GAIN(J), VTFCN(J), HCUT(J), LCUT(J),
3 INTFLG(j), DSPFLG(j),RTGFLG(j), STRFLG(j),
4 RZRFLG(j), SUMFLG(j),j=1,NCHAN),
5 SWCHAN,(SWNAM(j), SUNITS(j),SCHNUM(j),SHCUT(j),SLCUT(j)),
6 SDPFLG(j), SSTFLG(j), SSMFLG(j), J= 1,SWCHAN)
c
c
rewrite(4, err=9999) ZERO , TODAT, TOTIM, TPNUM,
1 NTSTS
c
c
c-----
C
c Write details in test log file
c
c
c
c
write(4, ERR=9999) TSNUM, TODAT, TOTIM, TPNUM,
1 TTITLE, ENGR, TFLUID, TSTDN, CPARM(1), CPLAG(1)
c
C Close Test Log File
c
Close( UNIT=4, status = 'KEEP' )
C
c Create Run Attribute file , once for each test
c
c Assign character sub-string to denote test-number
c for Run Attribute file name
c
write (RAFILE(17:22), '(16)' ) TSNUM
c
C write(TERMINL, 99118) RAFILE
c99118 format(1h,'RAFILE: ', a26)
c
open ( unit=4, file=RAFILE , status='NEW',
1 access='keyed',form='unformatted', RECL=210,
2 organization='indexed',recordtype='variable',
3 key=(1:2,3:11,12:19),
4 initialsize=10, err=9999)
c
Close( UNIT=4, status = 'KEEP' )
c
c Create Test Attribute file : done only once
c
C TAFILE = 'DU0:c210,241|PMPSTATT.TAT'
C
open ( unit=4, file= TAFILE , status='NEW',
1 access='keyed',form='unformatted', RECL=180,
c 2 organization='indexed',recordtype='variable',
c 3 key=(1:6,7:15,16:23,24:25),
c 4 initialsize=30, err=9999)
c
close (UNIT = 4, status = 'KEEP' )
c
61220 continue ! come from bad entry

```

A.4 TEST ATTRIBUTE FILE

Type: Keyed, Indexed, Variable Length, Unformatted

Life: Permanent

Keys:

- Test Number
- Test Date
- Test Completion Code

Other contents:

- Number of Measurands
- Measurand Name

- Measurand Unit
- Number of Software Channels
- Software Channel Name
- Software Channel Unit
- Transducer ID
- Number of Runs in Test

```

c
open ( unit=4, file= TAFILE , status='OLD',
1 access='keyed',form='unformatted', RECL=180,
2 organization='indexed',recordtype='variable',
3 key=(1:6,7:15,16:23,24:25),
4 initialsize=30, err=9999)
C
write (4 , ERR = 3999 ) RAFILE(17:22),
1 TSTDAT, TSTTIM, STCODE, NCHAN,
2 (MEASNM(J), UNITS(J), TRID(J),
3 j=1,NCHAN ),
4 SWCHAN,(SWNAM(j), SUNITS(j),
5 J= 1,SWCHAN), NPSG, SFREQ, RNUM
c

```

A.5 RUN ATTRIBUTE FILE

Type: Keyed, Indexed, Variable Length, Unformatted

Life: Permanent

Keys:

- Run Number

Other contents:

- Run Title
- Run Date
- Run Time
- Run Temperature
- Run Parameter

A.6 RUN DATA FILE

Type: Sequential, Variable Length, Unformatted

Life: Permanent

APPENDIX B. AST FOR KEYBOARD INTERRUPT

```

.MCALL SETF$C,QIOW$C,ASTX$S
QUIT:: QIOW$C IO.ATA!TF.XCC,5,,,,<UNSO>
MOV (R5)+,R2 ;THROW AWAY # OF ARGS
MOV (R5)+,SAVE
RETURN
.ENABL LSB
UNSO: CMPB (SP),#15 ;IS IT ENTER -- OCTAL 15
BNE CHK2 ;CHECK FOR SECOND CHARACTER
MOV R1,-(SP) ;SAVE R1
MOV SAVE,R1
MOV #1,(R1) ;SET THE INDICATOR
MOV (SP)+,R1 ;RETURN R1
BR 80$
CHK2: CMPB (SP),#64 ;IS IT 4 -- OCTAL 64
BNE CHKA ;CHECK FOR THIRD CHARACTER
MOV R1,-(SP) ;SAVE R1
MOV SAVE,R1
MOV #5,(R1) ;SET THE INDICATOR
MOV (SP)+,R1 ;RETURN R1
BR 80$
CHKA: CMPB (SP),#65 ;IS IT 5 -- OCTAL 65
BNE CHKB ;CHECK FOR FOURTH CHARACTER
MOV R1,-(SP) ;SAVE R1
MOV SAVE,R1
MOV #6,(R1) ;SET THE INDICATOR
MOV (SP)+,R1 ;RETURN R1
BR 80$
CHKB: CMPB (SP),#66 ;IS IT 6 -- OCTAL 66
BNE CHKC ;CHECK FOR FIFTH CHARACTER
MOV R1,-(SP) ;SAVE R1
MOV SAVE,R1
MOV #7,(R1) ;SET THE INDICATOR
MOV (SP)+,R1 ;RETURN R1
BR 80$
CHKC: CMPB (SP),#67 ;IS IT 7 -- OCTAL 67
BNE CHK3 ;CHECK FOR SIXTH CHARACTER
MOV R1,-(SP) ;SAVE R1
MOV SAVE,R1

```



```

MOV #2,(R1) ;SET THE INDICATOR
MOV (SP)+,R1 ;RETURN R1
BR 80$
CHK3: CMPB (SP),#70 ;IS IT 8 -- OCTAL 70
BNE CHK4 ;CHECK FOR SEVENTH CHARACTER
MOV R1,-(SP) ;SAVE R1
MOV SAVE,R1
MOV #3,(R1) ;SET THE INDICATOR
MOV (SP)+,R1 ;RETURN R1
BR 80$
CHK4: CMPB (SP),#71 ;IS IT 9 -- OCTAL 71
BNE CHECK ;CHECK FOR EIGHTH CHARACTER
MOV R1,-(SP) ;SAVE R1
MOV SAVE,R1
MOV #4,(R1) ;SET THE INDICATOR
MOV (SP)+,R1 ;RETURN R1
BR 80$
CHECK: CMPB (SP),#60 ;IS IT 0 -- OCTAL 15
BNE 80$
MOV R1,-(SP) ;SAVE R1
MOV SAVE,R1
MOV #-1,(R1) ;SET THE IND NEGATIVE
MOV (SP)+,R1 ;RETURN R1
80$: TST (SP) ASTX$$
.DSABL LSB
.EVEN
SAVE: .WORD 0
.END

```

APPENDIX C. GLOSSARY

Hardware Channel: A measurand, i.e., a physical quantity which is being measured by a transducer signal fed to a unique terminal on the analog end of the A/D converter.

Software Channel: A physical quantity, whose value is generated by the computer from one or more measurands in accordance with calculations stored in the computer.

Display Mode: Mode of operation of the computer system in which it emulates a series of panel meters displaying selected hardware and software channel values. Data storage in permanent memory is disabled.

Acquisition Mode: Mode of operation of the computer system in which it displays selected hardware and software channel values, and, on operator initiation, permits acquisition of Point mode or Scan Mode data.

Point Mode: Mode of acquisition of data in which, on operator initiation, a complete set of hardware and software channel values, corresponding to an instant in time, are recorded on disk.

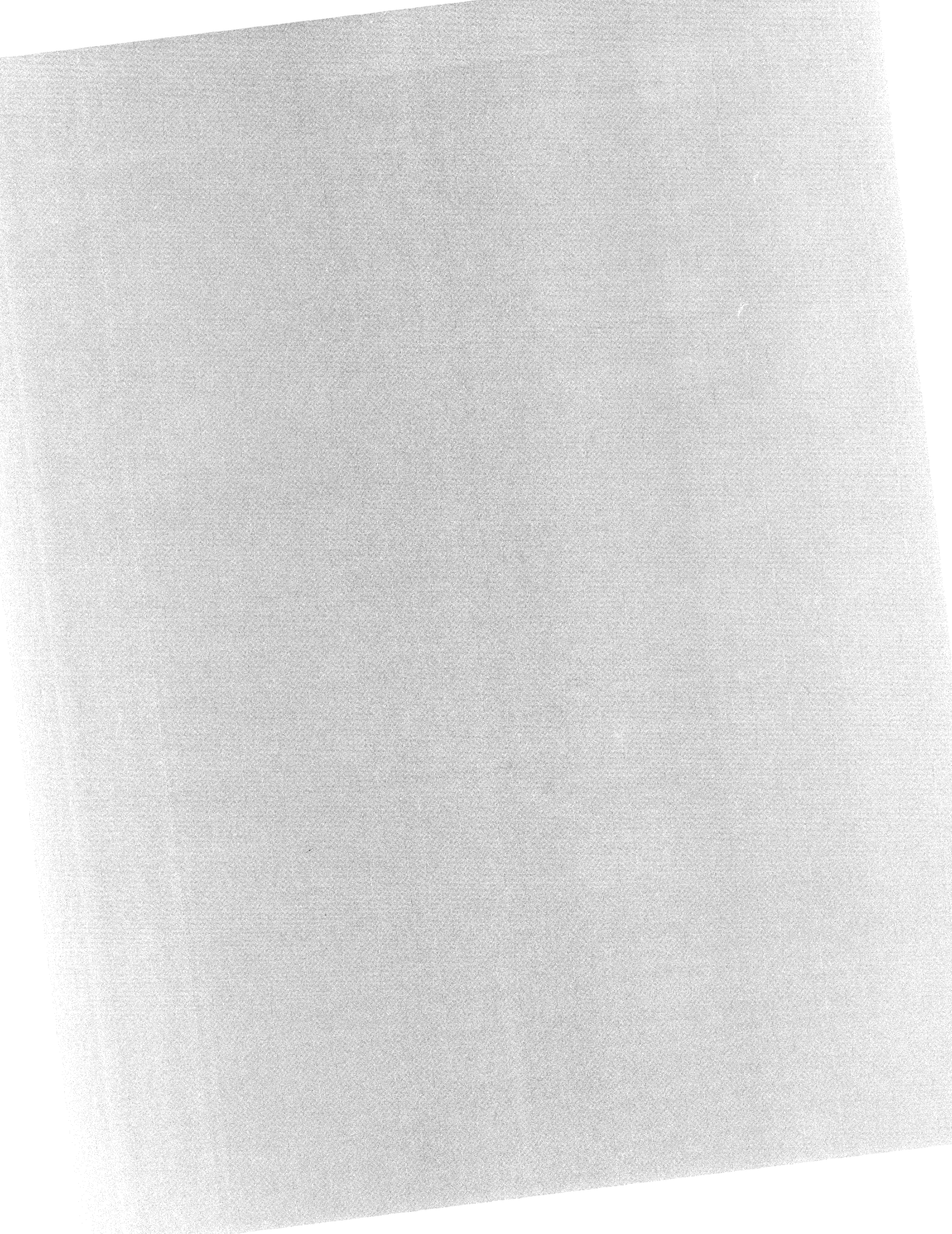
Scan Mode: Mode of acquisition of data in which, on operator initiation, a time history of raw hardware channel values only is recorded on disk, and subsequently processed to calculate engineering values for hardware and software channels.

Test: The test is the basic unit for maintaining records of test activities. Test components and setups would not be changed in the course of a test. A test would be constituted of one or more runs.

Run: A run would be a constituent of a test characterized by an optional run parameter value, nominally constant for the duration of the run. A run would consist of a number of sample points.

Sample: A constituent of a run, containing the set of numerical values of hardware and software channel values corresponding to an instant of time.

VAX SYSTEMS SIG



HSC50 Operations in a VAXcluster

Larry Herzlich
University of Texas at Austin
Computation Center
Austin, Texas 78712

Abstract

Now that the HSC50 has been installed at many sites, there are plenty of experiences that can be related to the perspective HSC owner as well as the new HSC manager. For many, the HSC is just a "Black Box" that occasionally sends messages to a console or the error logging system. This session hopefully removes some of the mystery concerning HSC Daily Operations in a VAXcluster

Introduction

This session was given from the HSC user perspective. Most of the information was gained through experiences from large VAXcluster sites and from the engineers at DEC working on the HSC50/HSC70 code.

The notes that follow are from the slides shown at the Fall Symposium except that the comments made about each topic are included.

The talk was preceded by a survey of the audience (approximately 200 people). In response to a question, 'How many are current owners of an HSCxx?', at least 80% responded. For the question, 'How many users are planning to purchase an HSCxx?', about 10 people responded.

I would appreciate any experiences, peculiarities, suggestions or comments about HSC50's or HSC70's sent to the address above.

HSC Management and Daily Operations In a VAXcluster

- The HSC Package
 - The pieces of hardware
- HSC In The Machine Room
 - Layout pitfalls to watch for
- Preparing Your Cluster For The HSC
 - A few changes for a VAX going to an HSC
- Preparing HSC Console Media
 - TU58's and RX33 floppy disks
- Understanding the Operator's Control Panel
 - What's normal, what's not
- Daily Operations

- HSC Utilities
 - Some of the common ones
- HSC50 versus HSC70
 - Some thoughts

The HSC Package

With version 3.00 of the HSC "Software", the HSC package will come in two parts: the Software package (Release notes, Console media, and User's Guide) and the Hardware (Channel cards, CPU, CI port, Console storage device, cabinet).

- Installation Guide
 - For field service
- Release notes
 - Part of the new User's manual, was most of V2.50 documentation
- Channel Cards
 - 8 Data channels on HSC70, 6 on HSC50, purchased separately
- CPU
 - Essentially a PDP 11/23 on HSC50, PDP 11/70 on HSC70
- CI Port
- Proper CI Cables
- Console Storage Device
 - TU58 cassette drives on HSC50, RX33 floppy drives on HSC70

- Console Media
 - V2.50 on HSC50, V1.00 on HSC70
- User's Guide
 - New for V3.00 - part number AA-GMEAA-TK, 300 pages

HSC In The Machine Room

- Locate central to CPU FARM, DISK FARM, TAPE DRIVES, CONSOLE and PRINTER
- Up to 45 meters from the Star Coupler
 - Set up cabling scheme early - allow for more (or replacement) disks, tapes.
 - Adding a new cabinet of RA disks should be easy; Offload a 'hot' spindle with more disks and Volume Shadowing.
 - Allow for swapping cables, unit plugs
 - If disk fails, a different drive can be used by changing plugs, ie. don't bind cable labels to unit plugs
 - Use grid diagram - cabinet, position - label like CI cables
 - Tag cables - easy to track cables at disk, CPU and Star Coupler. Hard to track cables at HSC due to high number of incoming wires.
 - Allow extra length for power - can rearrange in machine room
 - Given 15 feet of power cord,
- Console - Field Service prefers hardcopy terminal (LA12) but can use VT100 (New version of VTDPY??)
 - HSC70 running V1.00 has new VTDPY program. Was removed from V2.50 due to system crashing
- Allow for good ventilation - HSC and DRIVES are HOT!
 - 11,000 BTU/Hr. HSC50, 7,700 BTU/Hr. for HSC70
- Add a Second HSC near (or next to) the first HSC - Dual path (and Dual ported)
 - Now have FAILOVER option should one HSC fail. Cable scheme important to keep cables from getting crossed
- HSC node name - convention (0,1,...)
 - Works better for CI arbitration. Careful planning needed for adding more HSC's

- Add a Third!! HSC
 - Large disk farms need another HSC. Location can confuse cable scheme. Proposed Triangle formation. Problem: Can VMS store 3 paths to device in tables? Solutions???
- Check-out During Installation
 - Be sure Field Service does complete check-out before turning over HSC. Some tests were skipped by engineers at some sites.

HSC Channels...SDI and STI

- Connect disks and tapes over different channels
 - Recommend not all disks on one channel or all tapes on one channel - ie. buy more channel cards
- Tape channel card should be nearest the CPU cards
 - Related to prioritization on bus (State and Demand). Tape drives are less forgiving if not responded to promptly
- Watch for putting disks with expected High I/O rates on same channel
 - SYSTEM DISK
 - Volume Shadowing will offload 'hot' system disk.
 - Secondary PAGE/SWAP Files
 - Use of secondary Page and Swap files is common. Try not to put multiple Page/Swap file disks on same channel
- Four (4) or more channel cards require DC power upgrade
 - No power upgrade is required for HSC70

Preparing Your Cluster for the HSC

- Allocation classes - 0,1, ... up to 255 for CPU allocation class
 - Set up different allocation classe for HSC's serving disk farm
- Setting up the CPU's console media
 - Well documented in Guide to VAX/VMS System Management and Daily Operations manual
 - Register 2 in CIBOO/DEFBOO.COMD for Dual Ported Disks
 - High node number
 - Low node number
 - Register 3 for HSC Quorum Disk

- Quorum disk may not be necessary for clusters
- Check current version of CLxxxx.BIN on CPU console (See VMS Release notes)
 - SHOW CLUSTER /CONTINUOUS, then ADD RP_REV. For 8800 - Version 7 of CI microcode. Ask field service about prom replacement LO101.
- SYSGEN parameters - PANUMPOLL, PAMAXPORT reduced for few nodes
 - Can speed up time to poll nodes in cluster. HSC's 0 and 1, VAX nodes 2,3,4.... Reduce values from defaults.
- CONNECT/NOADAPTOR /DRIVER=FYDRIVER
 - Other terminals act as console - HSCPAD
 - SET HOST/HSC HSCnnn
 - Requires DIAGNOSE privilege
- SET DEVICE/DUAL_PORTED
 - Turns on logic in driver for failover

Preparing HSC Console Media

- Need 6 TU58 Cassettes or 2 Floppy disks (1.2MB)
 - SYSTEM
 - UTILITIES
 - DIAGNOSTICS
 - V2.50 if new HSC, else V2.00 if upgrade site
 - + enough for backups of each
 - System and Utilities on 1 floppy for HSC70. Diagnostics belong to Field Service.
- Leave ONLINE off, press INIT and FAULT to auto-configure
 - Gives new System Configuration Table. Generates a system ID based on node number and register on P.IOC
- DO NOT interchange SYSTEM cassettes between HSC's unless reconfigured
 - Parameters recorded in SCT can confuse CPU's if HSC reboots with wrong id to running cluster
- Setting the System Configuration Table (SCT) parameters manually
 - If upgrade - SHOW SYSTEM or SHOW ALL
 - ID (from register on P.ioc and node number)
 - * Serial number 0 for early HSC50's. Newer boards numbered

- Name (eg. HSC000)
 - Date (and time - Don't forget about Daylight savings time)
 - Allocation Class (255 decreasing... is one convention)
 - MAXTAPES
 - * When new HSC software installed, system allocates memory based on MAXTAPES = 24. Reduce number to reflect actual number of tape drives connected
 - MAXFORMATTER
 - * Set to 16 with new system. Reduce value for configuration but update value if new tape drives are added
- Reboot HSC to save values
- A RUNNING cluster may not talk to the HSC after ID or name changes!
 - Better to change id parameters when no circuits to HSC. Don't boot with ONLINE button on.

Understanding the Operator Control Panel (OCP)

- Press and release INIT switch to bootstrap
- Reading the State Indicator
 - Stays on while reading TU58 during Init
 - System is performing diagnostics
 - No optimization for TU58 position on tape, TU58 block layout optimized when kit was built.
 - Blinks fast while rebooting (2 times/sec)
 - Exec is running
 - Blinks (1 time/sec) while normal operation
 - After system initialization. Approximately 6 minutes for HSC50 reboot, 1 minute for HSC70
- Using the Enable/Secure Switch - Located inside door of HSC50, above controller cards for HSC70
 - Secure - pressing buttons on front panel has no effect.
 - Locking cabinet and Secure switch is sort of password.
 - Must be set to ENABLE to do anything other than SHOW
- Error reading the LED's - (Appendix B of User Guide)
 - Similar to LED's on RA disks
 - Fault light is on
 - Press and release Fault switch

- o Read Blinking LED's (Record value for Field Service!!)
- o Press and release Fault switch

Daily Operations

- Check for Crashes each morning
 - o If disks spread over 2 HSC's normally, then all disks on 1 means CRASH
 - o Check ERRORLOG
 - Console messages also sent to ERRFMT. SHOW EXCEPTION on HSC console. New User's Guide will describe faults. SOFT FAULTS will light panel but processor will continue (under V3.00)
 - o Out of paper?
 - Also check for enough paper for next day. Can disable paper-out detection (See LA12 guide)
 - o BREAK signal over console port = HALTED HSC - Use ;P to continue in some cases.
 - Related to state of processor and error traps in registers. Possibly caused by pulling plug on console terminal.
 - Clear break on HSC50 - If you think BREAK sent, leave in SECURE position, then type any 2 characters, then switch to ENABLE.
 - Clear break on HSC70 - If SECURE then no problem since BREAK is not remembered
- I/O Load balance - Dual porting plus failover makes this easy
- Rule of thumb - Less than 70% of drives on one HSC
 - o Some load for best optimization
- Monitor temperature
- Operator Control Panel in Secure mode
 - o Front door locked?
- Check indicator lights (Fault light on?)

HSC Utilities

- Get prompt by typing Control-Y at console
- Run SETSHO
 - o For HSC management
 - o Set/Show many things at once
 - o Will Timeout and exit program if left unattended - SET values not updated

• FORMAT/VERIFY

- o If you FORMAT you MUST use VERIFY
 - Certain errors not tracked during VERIFY if not FORMATTed
- o Not quite as good as factory testing
 - Factory formatter has lower level of expectancy - will flag more errors
- o Better left for Field Service

• TUCOPY

- o Does Block copy - you specify drives for source and destination of copy
- o Replace TU58 after about 1250 reboots
 - Your mileage may vary!
- o System Configuration Table template used with new SYSTEM.
 - Run SETSHO to update site specific parameters
- o Check Enable Switch position on HSC and Record tab on cassette

• BACKUP/RESTOR

- o Need TA series drives (TA78, TA81)
- o PHYSICAL backup - logical blocks copied *mostly* in order
 - HSC still does seek ordering (block clusters of size 15) so clusters may be copied out of order
- o For catastrophic recovery - ie. no other good backups
- o Only for Tape to Disk and Disk to Tape
 - Referred to as T1 or D1 rather than MFA1: or DUA1:
- o VERY FAST
 - Approximately 20 minutes - 4 minutes/tape so rewind time is significant.
- o Lacks ECC or XOR group computation recovery
- o Supports 2 tape drives
 - Allows processing to continue without rewind delay
- o Trouble with media errors causing BACKUP to abort
- o Reverse-Retry disabled in V2.50
- o Philosophical question - Is this the best method?
 - Do you need file level restore?
 - * Not available with PHYSICAL Backup
 - Can you afford the load to the CPU doing an Image Backup?

- * Image Backup usually done off-hours for least impact
- Can you afford the time that an Image Backup takes?
 - * Operator needed to monitor Backup during off-hours
- Can you afford a disk drive offline for ANY amount of time?
 - * Important volumes need high visibility at all times.
 - * Volume Shadowing solves some of this problem
- Is an HSC Backup really reliable?
 - * Not well tested in user community. Sensitivity to media problems prevents storing backups for long periods
- Others: ILTAPE, ILDISK, ILTAPCOM, ILEXER etc.

- These belong to Field service

Choosing an HSC70 versus HSC50

- Pros....
 - Eight Channels
 - Maximum number of drives now 32
 - Up to 32 drives with up to 16 tape formatters.
 - No extra power supplies needed
 - Types of CPU's in Cluster - using metric called Relative I/O figure of merit. Actual/Expected load should be deciding factor.
 - More than 3 8800's is too much for HSC50
 - Combination of 8x00 CPU's can be too much - see Sale's rep.
 - Possible space considerations - More connectivity in single cabinet
 - Console terminal - VT220 and LA50
 - Console storage - RX33 with fast reboot time
 - Better cabinet - lock front handle!, console storage not on door!
 - Faster processor + more memory
- Cons...
 - Maximum steady state data throughput same as HSC50
 - HSC70 slightly faster in terms of Requests/sec.
 - Increased price
 - We don't talk about this.
 - Combination of 11/7x0 CPU's

- Unlikely to saturate HSC50
- Saturation of Host CI port more likely than saturating HSC50 or cable
 - * Pipeline errors can appear during operation of tapes - due to HSC pre-fetch of buffers to keep TA81 streaming during heavy traffic



VAX CLUSTERING -- EXPECTATIONS AND EXPERIENCES

Gary Grebus and Michael Huffenberger
Computer and Information Services Department
Battelle Columbus Division
Columbus, Ohio 43201

ABSTRACT

The Computer and Telecommunications Center at Battelle Columbus began early in 1985 to consolidate VAX resources into a cluster and has been very successful at implementing the concept. At present eight machines, including 780, 785, and 8600 models, populate the VAXcluster at our site. Clustering provides many features and opportunities for resource sharing and workload balancing, most of which we have taken advantage of. During the first year we have observed several subtleties of implementing and operating the VAXcluster environment which may not be apparent to sites about to take the plunge. While the cluster does effectively couple resources, the result is not a nonstop processing environment, nor are the differences between constituent machines negligible. Commercial software operation and licensing are complicated in some cases. Local software is useful in cluster management. Understanding these issues has made our implementation more orderly and productive.

INTRODUCTION

The VAXcluster is a local high speed network connecting VAX hosts to each other and to clustered peripherals. New hardware is required to support this dual 70 megabit per second bus, including a Computer Interconnect (CI) which attaches to the host CPU, an SC008 Star Coupler providing a physical connection between host and controller nodes, and the HSC50 Peripheral Controller for tapes and disks. The Star Coupler must sit within 45 meters of any node, and the HSC50 is limited to 24 disks or 96 tapes on several HSC5X controllers. VMS Version 4 is required to support a VAXcluster.

The VAXcluster provides opportunities for improving system usage which undoubtedly become the basis for a site justifying a move to clustering. Fundamentally these opportunities are in Data Sharing, Improved Availability, and Load Sharing.

To enable data sharing, the VAXcluster supports direct access to clustered disks from any host, and pass-through type access to local disks owned by cluster hosts. Consequently there can be common data and system files, including batch and print queues, with access synchronized through the VMS Distributed Lock Manager.

Improved availability is provided by the fact

that users can be authorized to work from any host, and the nodes boot and fail separately. Thus reduced responsiveness on the surviving nodes is the only penalty for host failure. Dual porting the disks and tapes to multiple HSC50's can provide failover in case of controller failure.

Load sharing is achieved by virtue of distributing logons over the cluster. With shared batch and print queues, various workload types can be directed as desired by the System Manager. Operations support occurs on a cluster-wide basis.

The opportunities cited above create expectations for the site about to install a VAXcluster. In fact, while all the described features are provided, such a description is idealistic and simplified. In practice our installation has found that going to a VAXcluster requires attention to detail and an appreciation for the limits of the concept.

BATTELLE'S CONFIGURATION

Battelle Columbus set up a VAXcluster environment soon after the new hardware and software became available. It has been in operation since March 1985. The configuration now consists of:

- Eight VAX hosts, including 11/780, 11/785, 8600 and 8650 machines
- 12.75 gigabytes of RA81 disk (30 drives)
- Four TA78 tape drives
- Three HSC50 controllers
- 2 gigabytes of local MASSBUS disks accessible through MSCP server software
- Terminal service through an INFOTRON IS/4000 data switch

Our CI780, Star Coupler, and HSC50 hardware ran initially under VMS 3.7. In this mode the HSC50-supported disks were local. VMS 4.0 was then brought up on weekends for testing. After both the new hardware and VMS Version 4 were tested and stable, VAXcluster

production began.

OPERATIONAL EXPERIENCES

In general the performance history of our VAXcluster has been good. To date there is no evidence that the 70 megabit bus or HSC50 controllers have been bottlenecks. The common system files, such as user authorization, batch and print queues, and VAXmail, have been extremely useful. There are various complications we have encountered, as described below.

Software Failures. Bugs in the HSC50 firmware and with the shared file system (XQP software) have caused some system outages since startup. These were primarily early problems. Combined with HDA failures on the RA81 disks and with various TA78 failures, peripheral services have been the greatest problems we have had with clustering.

Cross-Cluster Software Operation. When the underlying architecture shifts to multiple hosts, as it does with the VAXcluster, there needs to be a corresponding adjustment of single-host software. (If operation across the cluster is desired.) Our site employs database packages with proprietary mechanisms for record addressing, outside normal VAX record services. They do not recognize the cluster, nor multiple copies of themselves on different machines. As a result a single application using these packages is confined to one host.

Analogous limitations with DEC software include lack of centralized accounting, no normalization of CPU times across nodes of different speeds, and the lack of cluster-wide logical names, event flags, mailboxes, and process control services.

Cluster Software Licensing. The cluster environment has been perplexing for commercial software vendors and contracts for clustered machines have varied, in our experience, from bargain- to premium-priced. Some vendors quote a cluster price, regardless of the number or types of machines in it. This keeps licensing simple. But many have sold into the 1 and 2 machine environment for a long time and insist on the simple markdown structures originally established for additional separate machines. The markdowns are not aggressive enough when the norm is becoming many machines rather than just a few. Thus any site feels economic pressure to support certain packages on certain hosts only,

contributing to administrative overhead and user confusion.

Where we have limited packages to certain hosts and the packages have no protection related to machine serial number, we have written local code to defer access from other hosts, protecting our side of the agreement with the vendor.

Heterogeneity in the Cluster. At first, intuition leads to an idealistic concept of the cluster where users may be shuffled around as needed, unaware of which machines are in the cluster, or indeed even which machine they are on. In practice we have not found the picture so simple. Having accumulated VAX's over the past several years, we have four different models of machines in the cluster, with decidedly different capabilities.

The variations in machines at our site, and probably for many other sites, are both intrinsic and due to our configuration strategy. An 8600 is much faster than a 780. As well, we have implemented various amounts of memory on the different machines, including 52 megabytes on an 8650. And we have been driven by economics to package certain software products on only some hosts.

Thus the machines differ enough that some applications are suited to running in only one or two places: the separate hosts become visible and individually crucial to the users once again. The system programming staff must be careful about taking hosts out of service, and about distributing users to achieve reasonable loading across the cluster.

Distributing Users for Load Balancing. Whether a cluster is homogeneous or not, some way must be found to distribute incoming users across machines. DEC can provide Ethernet terminal servers which allow the selection of a host or a generic assignment to any open port. We employ a data switch which accepts a generic "VAX" destination, or accepts specific host logical names.

The algorithm by which users are assigned is an interesting concern. One approach with a homogeneous cluster is simply to assign users round robin to hosts. In our heterogeneous cluster this would mean equal numbers of users on 780, 785, and 8600 class machines, an absurdity. Our data switch allows the round robin assignment to be weighted by host, thus an 8600 will end up with 4-5 times the users of a 780. The fact that some users must be

served by a particular machine confounds the actual workload balance, since the data switch continues to assign by the original round robin weights.

In fact these algorithms for assigning users to machines are primitive: they do not take into account what the assigned users are actually doing to the hosts where they are working. A few users could be overwhelming a 780 and it would still receive its normal proportion of additional users. A similar issue exists with load leveling across the batch queues. It is possible to see open slots in a batch queue as a basis for deciding whether another job should be added, however the actual CPU and I/O loads on the machine are not part of the equation.

The disparity in machine speed can also lead users to bypass the load leveling mechanisms in search of perceived better response. Indeed, a heavily loaded 8650 may still be more responsive than a lightly loaded 780. This phenomenon has led to under-utilization of the slower machines, and may further hamper load balancing as the cluster becomes more heavily loaded.

Performance. To date we have observed no constraints due to the bandwidth of the CI or HSC's. Our machines always seem to have a sufficient number of computable processes. The most frequent bottleneck observed has come from saturating a single spindle of disk -- predictable given a load from multiple CPU's instead of one. This is particularly true for system disks, where a tradeoff must be made between performance and the simplicity of maintaining a single copy of the system; we have multiple system disks. It becomes more important to consider both space requirements and I/O load limitations during capacity planning and daily operation. We have investigations underway aimed at better characterizing our disk I/O load in order to better configure our disk resources.

Memory might also become a consideration when moving existing machines into a cluster. It is difficult to estimate the exact impact in our case, since the VMS Version 4 upgrade happened simultaneously with clustering. A cluster node will typically have more disks mounted, with attendant use of memory for control blocks and file system caches. The lock data structures were also observed to consume more pool space in the cluster environment.

As expected, there is also some additional CPU overhead required to maintain synchronization among the cluster nodes, although the impact has not been particularly obvious. Some I/O intensive applications requiring heavy locking activities appear to take longer. Rough measurements indicate that file opens take about 5 percent more CPU time when a disk is mounted cluster-wide.

Finally, sites should not indulge the fallacy of summing up all the MIPS available across a cluster and deducing that aggregate power equals individual power. A large computational job running on a clustered 8600 may still take too long. Jobs with massive I/O requirements also may be better directed to mainframes with high speed channels.

Reliability and Availability. The VAXcluster concept provides a framework for achieving very high availability of resources. Typically when we have a system crash, displaced users will gradually reappear on the machines still functioning. The load on these machines builds until the failed host reboots, and the workload then redistributes itself. A cluster, nonetheless, is unlikely to be a nonstop processing environment; certainly ours has not been.

The multiple hosts provide redundancy but differences in capability and in software cannot be neglected totally when some particular host goes out. It is wise to provide all critical software products on at least two machines in the cluster.

Controller failures, particularly software, have at times paralyzed our cluster. This problem has been particularly aggravated by problems with support of the TA78 tape drives. During the HSC50 reboot, the user's file access is blocked. The user assumes the system is dead and disconnects, though in a few minutes the terminal would "unlock" and resume. While dual paths can be set up to disks through two HSC50's, we have not yet done so. (Smaller clusters may not even have two HSC50's.)

Disk drive failures are probably the most troublesome failures a cluster has. An outage will likely affect users across the entire cluster, compounded by the fact it may be difficult to exactly identify which users are affected. Volume shadowing is not a panacea since there will likely always be disks which, for economic reasons, will not be replicated.

VMS provides no facilities for gracefully terminating access to a failing device. We have written local software to allow a

particular disk volume to be taken out of service, bringing down only those users affected by that volume. Following a warning message, an affected user can be logged out and prevented from logging in until the necessary resource is again available. This is in contrast to the only recourse previously available, that of shutting down the entire cluster.

It is extremely desirable to have one or more disks as hot spares in case of drive failure, to circumvent relocating files. The reload time for an RA81 is penalty enough. Disk to disk copying as a recovery technique can save considerable time when compared to dumping and reloading using tape. (Spare disks also serve as a handy resource for disk compressions, not requiring the original data to be overwritten.)

Common system files--for example, a single copy of SYSUAF.DAT-- also may present single points of failure. We eagerly await the forthcoming volume shadowing facility to eliminate this problem.

CONCLUSION

Our VAXcluster experiences over the past year have been very favorable, on the whole. The availability of more powerful VAX's such as the 8600, in conjunction with the VAXcluster concept, have allowed us to expand our VAX configuration to become our major resource for centralized computing. (There are IBM and CDC hosts at our site also.) We anticipate that many other laboratories and businesses will do the same, guided by the ideals of the VAXcluster concept and by the practical experiences of sites such as ours.

HIGHEND VAX I/O BENCHMARK

Don Hamparian and Michael Huffenberger
Computer and Information Services Department
Battelle Columbus Division
Columbus, Ohio 43201

ABSTRACT

The power of the VAX 8650 has allowed sites to consider supporting larger scale applications than could be assigned effectively to earlier VAXes. As a site with machines all up and down the VAX product line, and other vendors' machines competing as well for our users' computing workloads, we have been very aware of the new potential of the VAX 8650. Of particular interest to us is the capabilities of the VAX 8650 and HSC based disks to perform I/O intensive processing traditionally thought of as "mainframe" data processing. In this paper, we provide results of benchmarking our 8650 machine using a workload consisting of simulated users querying a textual database system. Results indicate our users' growing allegiance to the VAX architecture is justified for a growing range of large and small computing workloads.

INTRODUCTION

Since the introduction of the VAX 11/780 in 1978, many VAX processors have been introduced that have raw CPU power many times faster than the original 11/780. The VAX 8650, introduced in 1985, has raw CPU performance about 5.5 times that of the 11/780. Many traditional mainframe applications such as corporate financial systems and large production databases that would never be hosted on VAX just a few years ago are now running as production systems on our VAXcluster.

Now that Digital has provided CPUs with the computing power to process such systems, our attention has turned to the I/O subsystem of our VAXcluster. We developed this and other benchmarks in order to learn about the capacity of RA81 disk based I/O subsystem.

The RA81, a 456 megabyte Winchester disk drive is currently Digital's high-end disk drive offering for the VAX line of processors. By learning about the capacity of the I/O subsystem, we were then able to better tune our I/O workload across our RA81 disk base.

BENCHMARK CONFIGURATION

Hardware. Two benchmarks were run, referred here as Test 1 and Test 2. The two benchmarks had identical hardware configurations except for the paging and swapping disks - Test 1 had one RA81 dedicated to paging and swapping, while Test 2 had 5 RA81s dedicated for paging and swapping. The hardware configuration for the benchmarks consisted of:

- VAX 8650 with Floating Point Accelerator (FPA) and 52 megabytes of physical memory
- 3 HSC50s running HSC software version 250
- 1 RA81 serving as the system disk
- 1 RA81 containing the database manager image files
- 1 RA81 containing the database data files
- 3 RA81s bound as a volume set for user procedure and listing files
- 1 RA81 dedicated for paging and swapping (Test 1)
- 5 RA81s dedicated for paging and swapping (Test 2)

We attempted to spread the disk I/O load over all three of the HSC50s. None of the HSC50s had more than 6 RA81s attached to it. Also, none of the HSC disk channel cards had more than 3 active RA81s attached to it.

Software. The software environment for both benchmarks was identical. Both benchmarks were running VMS 4.2, the System Performance Monitor (SPM) version 2.0, and Battelle's BASIS database management software. SPM is a valuable measurement and tuning tool, providing effective metrics for both benchmarks and daily system operations.

Since the database software provides its own database file management services, it is important to note that the database files are not managed by VAX Record Management Services (RMS) software. Therefore, the throughput statistics of these benchmarks may not be indicative of a RMS based database application such as Datatrieve layered with RMS indexed sequential files.

BENCHMARK STRATEGY

In this benchmark, we intended to measure the overall throughput of the 8650 in a multiuser

database environment. The benchmark application was intentionally chosen because it was I/O and memory intensive rather than CPU intensive. It was also simple enough so that we could effectively measure performance of various hardware and software subsystems without expensive custom-written instrumentation and data gathering software.

Although there are many hardware and software components to the I/O subsystem, we concentrated our analysis on the RA81 disk drives because we felt that they represented the I/O bottleneck in our benchmark. Our test results as well as daily monitoring of our VAXcluster confirmed this as well as showing that the Computer Interconnect (CI) and HSC-50 disk controller were not significant bottlenecks. At no point in our benchmarks did aggregate I/Os to all three HSCs exceed 310 I/O requests per second.

Benchmark Environment. In order to maximize system throughput and eliminate I/O contention from other VAXcluster members during our benchmarks, the following steps were taken:

- Our VAXcluster was completely idle. No other nodes on the cluster were running at the time of the benchmarks.
- All of the database files used in the benchmark were contiguous.
- All of the database manager images used in the benchmark were contiguous.
- All of the database manager images used in the benchmark were INSTALLED /OPEN /SHARED.

Due to time and cost constraints, all database "users" were emulated using detached processes. The database scripts were read from DCL procedure files. Therefore, the overall system I/O statistics do not include the overhead of terminal related I/O.

All SYSGEN parameters were tuned by the AUTOGEN command procedure. Since we have found in daily operations that increasing the size of file system and memory caches above the values set by AUTOGEN have a positive impact on overall system performance, SYSGEN parameters related to file system and memory

caching were set higher than the values recommended by AUTOGEN for the benchmarks. All SYSGEN parameters were identical for both benchmarks.

Loading Techniques. The database "users" for the benchmarks were loaded using the following techniques:

- The benchmark "users" were created using detached processes.
- All of the database scripts included what we considered to be typical database user think time. Think times were implemented as by the equivalent of "WAIT n SECONDS" statements in the database scripts.
- Since the benchmark data gathering was done in 20 user increments, the VAX 8650 was loaded by creating 20 new detached processes, waiting for each of the processes to invoke the database manager image, and then gathering resource data for approximately 15 minutes. This was repeated for each 20 user increment. Note that the measurements were done with 40 users, then 60, 80, 100, 120, 140 and 160 users. Test 2 also had a 170 user measurement taken.

BENCHMARK RESULTS

Throughput. Throughput was defined as the amount of useful work done by the entire benchmark hardware configuration per unit time. This differs from the throughput that each individual user experienced which was not measured. Throughput was measured by the database manager which includes a subsystem for monitoring the types of user database queries invoked. Since each 20 user group ran the same mix of database scripts, we measured throughput by the number of monitor records written by the database manager per unit time.

Our benchmarks yielded maximum throughput at

100 users for Test 1 and 140 users for Test 2. See figure 1 for a graph of throughput measurements for both benchmarks.

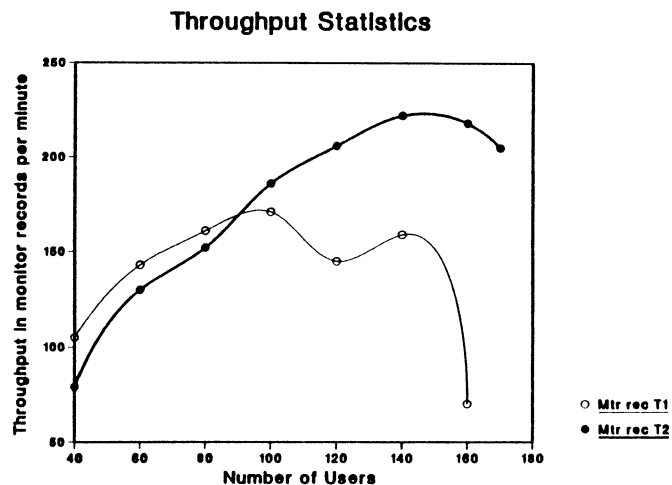


Figure 1.

CPU Utilization. Both of our benchmarks showed that the CPU was not the benchmark bottleneck. Figures 2 and 3 show the CPU mode utilization for Tests 1 and 2 respectively. Note that most of the CPU idle time was attributed to paging and swapping I/O waits. This would indicate that the paging and swapping disks were part of the overall benchmark bottleneck.

Using SPM's program counter (PC) sampling facility, we gathered PC statistics on various I/O related subsystem components. We found the following maximum PC utilizations for the following I/O subsystem components:

- RMS - 3.8% of total PC samples
- DUDRIVER - 1.5% of total PC samples
- PADRIVER - 2.6% of total PC samples

Note that the database files in the benchmarks are not RMS based.

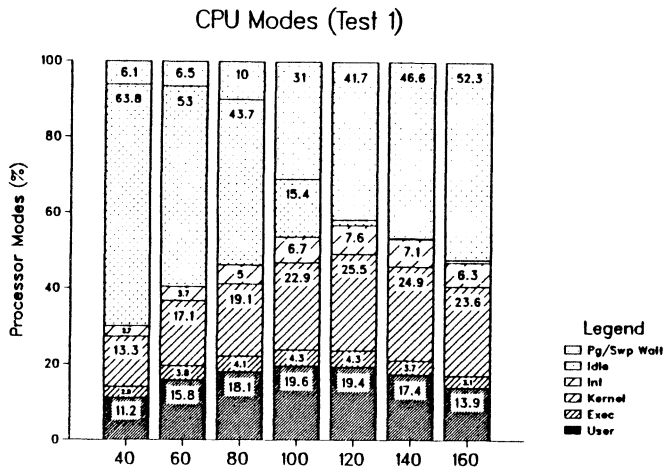


Figure 2.

Memory Utilization. Our benchmarks showed that 52mb of physical memory would allow us to run "memory rich" to about the 80 to 100 user load. "Memory rich" in this context is defined as few (less than 10) hard page faults per second (read and write page I/Os) and a free page list size that is not less than the SYSGEN parameter FREEGOAL pages at any time. Obviously, the fact that a that a memory shortage occurs above about 100 users is the reason why the page/swap disk(s) are a performance bottleneck at that point. If we had more memory, the page/swap disk(s) bottleneck would be removed and the database disk would be the performance bottleneck for both benchmarks.

I/O Subsystem Utilization. Both of our benchmarks showed that the RA81 disks were the primary performance bottleneck. No other I/O subsystem component was a significant contributor to the benchmark performance bottleneck. Figure 4 shows the overall system I/O rates for both benchmarks. The buffered I/O rates are fairly constant for both benchmarks since there were no terminals contributing to that measurement. The direct I/O rates peak at almost the same point (number of users) as the throughput statistics (figure 1). Test 2 showed a greater direct I/O rate because the paging subsystem, which was limited to one disk spindle in Test 1, was now utilizing five disk spindles.

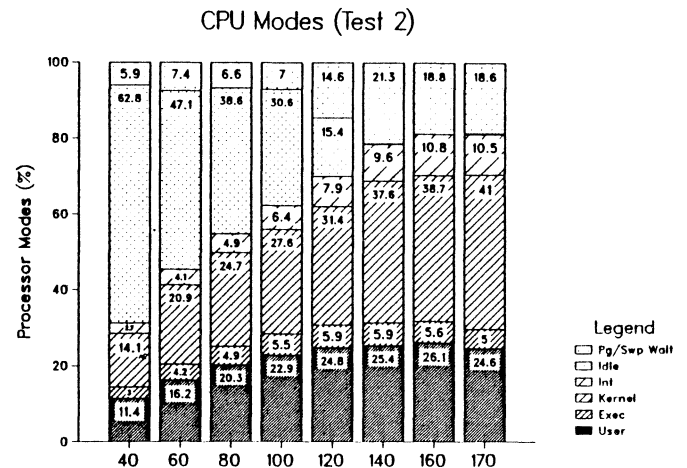


Figure 3.

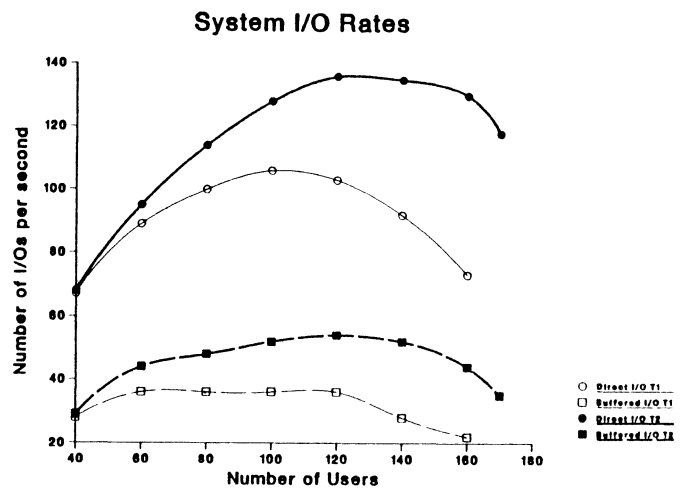


Figure 4.

Our overall I/O subsystem performance was primarily limited by the database disk in both Test 1 and Test 2. The page/swap disk was a close competitor for performance bottleneck in Test 1. Figures 5 and 6 show the I/O request rates for each disk subsystem for Test 1 and Test 2 respectively. Note that as the page/swap disk saturates in Test 1, the database disk I/O rate drops since many processes are in a page/swap I/O wait state and thus unable to do any I/Os to the database disk. In Test 2 (Figure 6), note that the 5 spindle page/swap disk I/O rate is unrestrained by spindle limitations and therefore does not significantly contribute to the I/O subsystem

bottleneck. Database disk I/O is decreasing at about the 140 user level because the many processes are suffering from long service times to the database disk resulting in fewer database disk I/O requests per process per unit time. Note that the system disk, image file disk and user disk were not significant contributors to the I/O subsystem bottleneck.

1 the database disk saturated at about 50 I/Os per second, while in Test 2 the database disk saturated at about 42 I/Os per second. The only difference we could see in our hardware configuration that could account for this difference is that in Test 2 the database disk shared a HSC channel card with one of the 5 paging spindles. Unfortunately, there is no instrumentation available at the HSC level to prove or disprove this thesis. This high I/O rate probably would not be attainable on a typical user disk since all files referenced on the database disk were contiguous. Also, since only about 5 files were referenced on the database disk during the benchmarks, time consuming track to track seeks were minimized.

Summary of I/O Request Rates (Test 1)

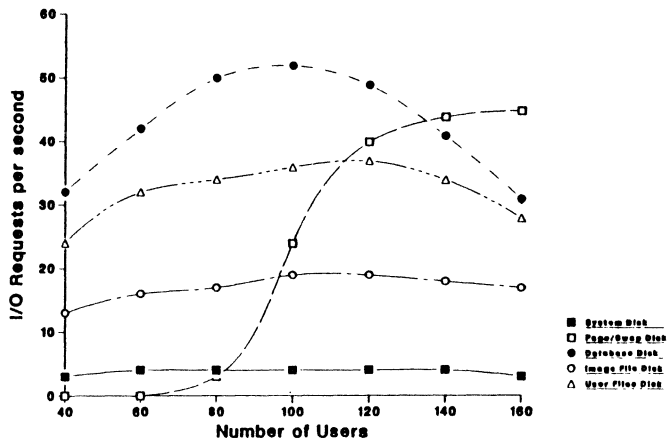


Figure 5.

Database Disk

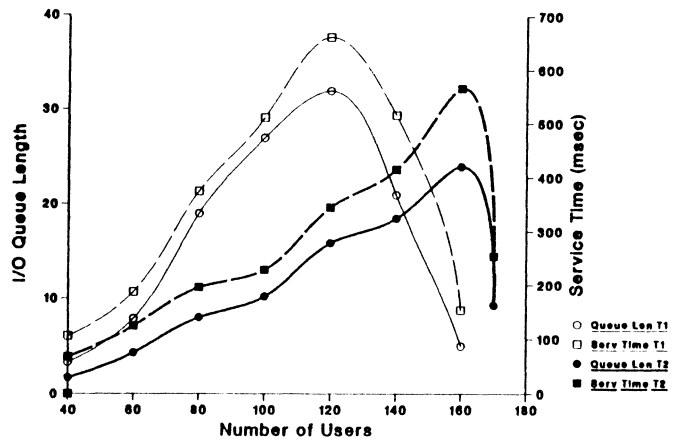


Figure 7.

Summary of I/O Request Rates (Test 2)

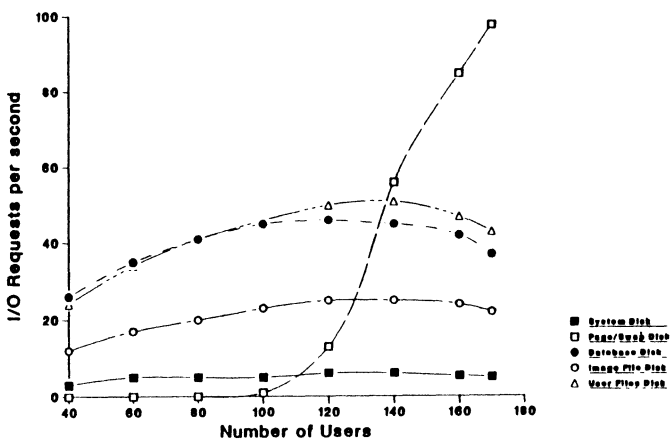


Figure 6.

Figures 7 and 8 show detailed performance information for the database disk for both benchmarks. The database disk becomes 100% busy during both benchmarks at about the 80 user level. As the database disk saturated, service times became as long as 650 milliseconds per I/O request. Note that in Test

Figures 9 and 10 show detailed performance information for the page/swap disk(s). Since Test 2 used 5 page/swap spindles, the percent busy, the queue length and the service times were calculated as the average of the 5 spindles for each benchmark user load. The I/O requests per second statistics are calculated as the sum for all 5 spindles for each benchmark user load. In Test 1, the page/swap disk spindle saturated at about the 120 user load level. This saturation level corresponded to about 45 I/Os per second being serviced on the page/swap disk. At the 160 user load in Test 1, a page/swap disk I/O took almost 8 tenths of a second to complete. Test 2 showed that there was still reserve page/swap disk capacity available throughout the entire load of users. At the 170 user load in Test 2, nearly 100 I/Os per second were being processed by the page/swap spindles.

CONCLUSIONS

I/O Load Distribution. This benchmark demonstrated the importance of properly distributing the disk I/O load across disk spindles in I/O intensive applications. A VAXcluster utilizing a shared file base also needs to be carefully tuned to prevent "hot" disks that can affect performance of the entire VAXcluster. A ballpark figure we use to determine when a disk is saturated is when the cluster-wide total I/O request rate averages above 20 I/Os per second during our VAXcluster's prime-time hours. Also, make sure that you are not overloading your HSC Channel cards. This is largely a subjective determination; no software exists to measure HSC channel card loading.

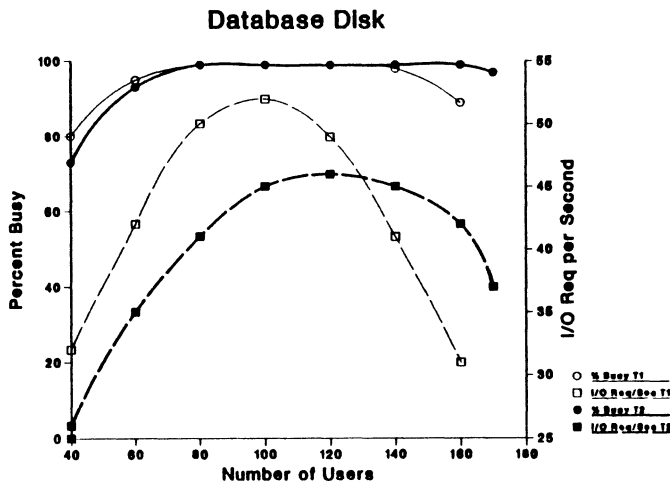


Figure 8.

The I/O rate for the page/swap disk, much like the I/O rate for the database disk, cannot be used as a maximum value for a typical user disk used for timesharing. Each page/swap spindle only had 2 open files on it and each of those files contained no more than 3 extents. Again, because of these properties, time consuming track to track seeks were minimized on the page/swap disk. A typical user disk would tend to have many small, fragmented files and most of the I/Os to such a disk would tend to be small (1 to 3 blocks) I/Os that would require expensive track to track seeks.

Page Disk

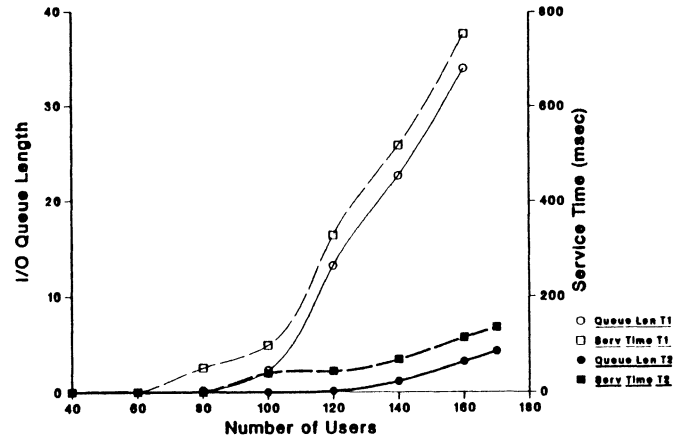


Figure 10.

The Important Disk Statistic. Many disk vendors like to quote the maximum transfer rate of their drives. Unfortunately, this statistic is nearly meaningless in a typical VMS environment. Since disk I/Os in a typical VMS environment tend to be small and randomly placed on a disk, the most important disk statistic is the average time it takes to access a block of data on disk. The maximum transfer rate of a drive may be important if utilities such as physical backup are used or for dedicated page/swap disks.

Qualifications. This benchmark was purposely selected because it is I/O and memory intensive. Our typical computing mix is less I/O and memory intensive than this benchmark. We

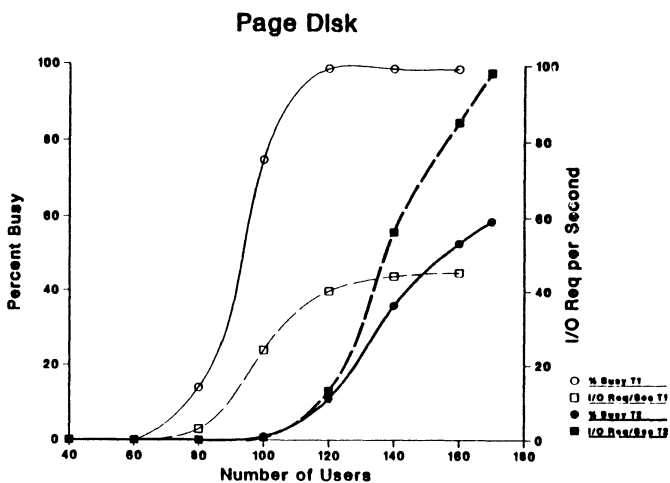


Figure 9.

could probably load an additional 20 users over the benchmark throughput maximum and achieve acceptable performance in our daily computing environment.

RECOMMENDATIONS

Distribute the I/O Load. It is critical in I/O intensive applications to distribute the I/O load across spindles and HSC channel cards. I/O load tuning is an iterative process that is done over the long term. Expect I/O load tuning to be repeated as new user applications are developed and old ones are ended. Running the VMS Monitor utility or SPM as a detached process as part of daily operations gives the system manager a good source of data for I/O load balancing and forecasting.

Distribute Page/Swap Loads. If your system is doing a lot of hard page faults and Monitor or SPM indicates that the page/swap disk is nearly 100% busy, distribute the page/swap loads to multiple spindles. Remember, if your page/swap files are on your system disk and your system has a lot of hard page faults, performance of the entire running system can be affected. Do not put your page/swap files on user disks unless the combined I/O rates of the user disk I/O and page/swap I/O do not exceed the disks' capacity.

SYSGEN Parameters. Use file system caching and free and modified memory lists as much as possible to reduce disk I/Os. File system caching is the primary method VMS provides to reduce disk I/Os without special programming or application tuning intervention. RMS file and program tuning can also reduce disk I/Os, but it is sometimes difficult to talk users into spending the time to do tuning.

Disk Compressions. Performing disk compressions will help reduce the number of physical disk I/Os required for a virtual (file system) I/O. In daily operations however, we find that the 8 to 10 hours required to do a dump to tape and reload to disk a 3 spindle volume set makes disk compressions a rare occurrence. We do not do disk to disk compressions because we do not have the luxury of having 3 spare spindles to do volume set compressions to.

Do NOT Skimp on Memory! Memory is the cheapest and most effective means of improving VMS performance. Saving a little money by skimping on memory allocation for a system is not worth the performance

headaches ahead. Remember that a VAXcluster requires more memory than a stand alone system. I/O intensive applications require more physical memory to contain file system control blocks and locks than a compute intensive application. Also, if there is not enough memory available to contain larger file system and memory list caches, the effectiveness of these caches are greatly diminished.

INSTALL images. Install all frequently referenced images. If we had not INSTALLED the database images, we would not have been able to load our 8650 with nearly as many users. Besides reducing paging from the image file, INSTALLING an image /OPEN /HEADER_RESIDENT will reduce disk I/Os and speed up image activation time.

SUMMARY

We have found the 8650 and VMS to be a powerful combination for large-scale "mainframe" type I/O intensive processing. By balancing disk I/O load across a system and VAXcluster, performance can be greatly enhanced. We now feel that a new generation of faster disk drives need to be introduced to catch up with the power of the 8650 and similar power VAX processors. Once disk technology catches up with the new VAX processors, the VAX will be a true high-end data processing system able to compete effectively in all classes.



A Robust VMS LOGOUT Driver Activator

Larry L. Johnson
Texas Instruments
McKinney, Texas

ABSTRACT

We have constructed a data-flow model of a generic software product in the context of a general time-sharing environment. In this model, points of control for each product are defined for each "system-event". One of those events is the deletion of a process (specifically, LOGOUT). The current implementation of the activation of our logout driver (which calls per-product logout functions) is weak. An unprivileged user can exit the system in a manner which will bypass the activation of the logout driver. By the definition of our model, logout operations are not optional. A weak activation of the logout driver is intolerable. This study was undertaken to determine the feasibility of implementing a robust logout driver activation.

This study established the feasibility of implementing a robust logout driver activation by:

1. proposing mechanisms on which a solid logout driver activation can be based.
2. producing a functional prototype based on a selection of these mechanisms.
3. proposing the mechanism and general structure of a production version.
4. outlining the course of action to produce such a version.

A comparison of the advantages and disadvantages of various available mechanisms is presented. Based on the experience derived from the mechanisms for the prototype, recommendations are made for the selection of mechanisms and structure for an eventual production version.

The considerations to be addressed by future efforts is clearly and specifically defined. The major thrust of the considerations is to evolve the current functional prototype in order to meet design goals of per-site flexibility, staying out of the way of per-machine system programming, and providing the

per-machine system programmer with tools to interface with the mechanism more effectively than if the programmer attempted to accomplish his task with "vanilla" VMS.

2 Problem Analysis

2.1 Primary Objective

The primary objective of the study is to determine and eventually implement a usable mechanism to force the execution of a system-wide logout command file.

The mechanism is to be hardy and non-defeatable by an unprivileged VMS user.

2.2 Decomposition of Primary Objective

The only common point of exit for processes of all types is through the VMS image rundown. Process deletion is a special, extended case of image rundown in VMS systems. Therefore the first objective of the study must be to intercept image rundown, and recognize the case of process deletion (rundown of the process).

Having intercepted process deletion, the process must be restored to a state capable of defeating user control and forcing the execution of a normal DCL command file in a normal user process context.

Having established a suitable environment, a mechanism is required to force DCL to execute a system-wide logout command procedure. (Simulate the "@command-file" behavior).

The command file must have a mechanism by which it can request a process deletion (without forcing its own execution once again).

The procedure must be immediately available to the process context, which means it must be permanently mapped and not suffer the effects of image rundown.

A method for loading such an image must be developed.

In summary, to achieve a solution the following subproblems must be solved:

1. Design mechanism to intercept normal VMS process deletion. (special case of image rundown interception).
2. Design mechanism to recover process from deletion.
3. Design mechanism to transfer control from user to system-wide logout command procedure.
4. Select rundown interceptor image residency.
5. Develop prototype rundown interceptor using above mechanisms.
6. Develop procedure to load rundown interceptor.

3 Discussion of Technical Issues and Candidate Solutions

The user is presumed to be intimately familiar with VMS Internals for this material.

3.1 Process Deletion Interception

In order to intercept process deletion for all processes, control must be gained each time VMS deletes a process. VMS supplies the hooks necessary for this through a dispatch vector mechanism described below.

3.1.1 VMS Dispatch Vectors.

VMS provides for the call of user-written routines at points critical in the VMS architecture: For each of these critical points VMS maintains two hooks:

1. System-wide vector pointer (S0 space).
2. Per-process vector pointer (P1 pointer page).

The critical points at which control is provided for each of the hooks are:

1. Change mode to kernel dispatching. If the VMS change mode to kernel dispatcher does not recognize a system service code, it will call user-written dispatchers through the system-wide pointer if it is non-zero. If the system-wide dispatcher does not recognize and handle the change mode code, user-written dispatchers are called through the per-process pointer.
2. Change mode to executive dispatching. Dispatching is handled analogously to the kernel case.
3. Image rundown dispatching (process deletion is a special case of image rundown under VMS). The per-process dispatch vector is called through the per-process pointer. If the system-wide dispatch pointer exists (is non-zero) it is called. The vectors are called exclusively by SYS\$RUNDWN and by SYS\$DEL.PRC.

The following are the symbolic names of the locations of each of these pointers.

	System-wide (S0 Space)	Per-process (P1 Space)
Change mode kernel	EXE\$GL_USRCHMK	CTL\$GL_USRCHMK
Change mode executive	EXE\$GL_USRCHME	CTL\$GL_USRCHME
Image Rundown	EXE\$GL_USRUNDWN	CTL\$GL_USRUNDWN

The system-wide pointers are not set by VMS. There is no established protocol for their use. There is, therefore, no chance of conflicting with VMS operations in the use of these vectors. However the vectors are truly system-wide and will be invoked by any process. Any exclusions would have to be dismissed by analyzing the calling environment at each invocation.

The per-process vectors are used by the image activator in mapping privileged shareable images. This is the supported method for providing user written change mode dispatchers and rundown services on a per-image basis.

The per-process pointers point to vector tables in P1 space. These are referred to as VMS dispatch vectors. The tables are one-half page each, located in two contiguous pages. There is a fourth set of pointers/vectors for message sections. The fourth table does not follow the format of the other three described here.

The tables are not strictly "jump tables", but mini-routines composed of JSB instructions to each privileged dispatcher and terminating with an RSB instruction. The image activator adds a JSB instruction to each of the tables as required in the encounter of any installed privileged shareable image. The number of such vectors is restricted by the size of the dispatch area (256 bytes for each table... 42 JSB sequences).

The image activator keeps track of the original (permanent) size of each vector table and, at image rundown, over-writes the JSB op-code written there at image activation with the original RSB op-code. Thus, at image rundown the vector table is reset to look as it did before image activation.

It is possible in VMS V4.x systems to make these vectors permanent by updating the location containing the original size of the vector table to the size of the table with the permanent vectors in place. (This possibility did not exist under V3.x and earlier systems. The vector tables in those systems were reset to no vectors at each image rundown.)

3.1.2 Selection of Vector Mechanism

The use of the system-wide vectors offers the following advantages.

1. No VMS established protocol. There is little chance of clashing with VMS's use of the vector.
2. Per-process modifications are not necessary. Once established, the rundown interceptor will be activated for every rundown in the context of every process, regardless of its environment.

The disadvantages of system-wide vectors are:

1. A protocol for use of the vector would have to be established. VMS's use of the per-process vector could serve as a convenient model for the protocol, making the effort straight-forward.
2. Since a logout command procedure driver is meaningful only to those processes which map DCL, ineligible processes would have to be detected so no operation would be performed, or an alternate image-based system-wide logout be performed.
3. The use of the system wide vector would require mapping the image into S0 rather than P1 space, since each P1 space may have the image mapped into different address ranges. P1 space could be used to keep per-process data. This would require a check on each activation to assure that the P1 data area has been initialized, and the initialization performed if it has not.

Per-process S0 allocated areas could also be used, established on any access indicating absence of structure. This approach essentially implements an extended PCB and should be accessed with similar protocol.

The per-process vectors offer the following advantages:

1. They can be loaded on per-process basis by the system-wide login procedure as desired. If the image is activated, it is eligible to run. Fewer environmental determinations are necessary.
2. Any P1 data area can be mapped at the same time as the rundown image.

The per-process vectors offer the following disadvantages:

1. They use structures under the domain of the image activator.
2. Vector pointers are not copied to subprocess by SYS\$SPAWN implying that the SYS\$SPAWN system service may have to be tampered with to make the facility work for spawned subprocesses.

There is some danger in sharing the same data areas with VMS's image activator, but the new provision for permanent vectors is clearly an intentional feature, though not currently utilized by VMS. Should VMS develop uses for this feature that makes our use of it impossible, we can fall-back on the system-wide vector mechanism which is left entirely to the VAX owner.

Alternately, the vector itself could be revectored to our own code which would be responsible for locating and activating the image activator's tables, leaving the dispatch area solely under the domain of VMS image activation and rundown logic. This is possible since the image activator locates the dispatch area through one pointer, CTL\$A_DISPVEC and the VMS code which calls the vectors locates the particular dispatch table through the per-process vectors. (It remains to be strictly verified that the image activator uses only CTL\$A_DISPVEC, and not any of the specific vector pointers.)

Since we are using the dispatch area in a fashion consistent with the architecture of VMS, it is unlikely that an unresolvable conflict will occur. However, it is important that contingency mechanisms are available.

3.2 Recovery from Process Deletion

When SYS\$DELPRC is called (from whatever process), the PCB of the target process is flagged as delete pending and a kernel mode AST is declared. Process deletion always occurs in the context of the target process. Deletion is accomplished through the AST even if the requesting user is the same as the process targeted for deletion.

Process recovery draws on an analogy between the rundown dispatch mechanism and the search for user-written change mode handlers. If a change mode handler does not recognize a change mode code, it performs an RSB which returns for the search for other handlers. If the handler recognizes the code, it branches to the routine which performs the function. That routine returns with a RET to the last active call-frame which was established by the change mode exception dispatcher.

The environment provided by the kernel mode AST for process deletion coupled with the very early call of the rundown vectors by the SYS\$DELPRC system service, makes recovery of the process straight-forward. The rundown interceptor is called by the SYS\$DELPRC AST with a JSB instruction. The VMS AST delivery mechanism has called the deletion AST code with a CALLG instruction. Consequently, if the rundown interceptor returns via an RSB instruction, it will return to the deletion AST which will continue through process deletion. If, however, the facility returns via a RET instruction, control is passed back through the last active call frame, i.e., to the AST dispatcher which dismisses the AST, causing the process to continue as though nothing had happened.

Before dismissing the AST to abort process deletion, the delete pending bit in the process's PCB must be cleared. This flag is also useful for verifying that process deletion is the reason that rundown has been requested for kernel mode. It is set by the minimal code in SYS\$DELPRC that is executed in the requester's context before queuing the kernel mode AST.

3.3 Forcing System-wide Logout Command File Execution.

Having aborted the deletion of the process, the program must control and force execution of a system-wide logout command file. In other words the interceptor must cause DCL to execute an "@command_file" type command in the normal process context, but beyond control of the process owner.

The usual way an image causes DCL to execute such a command is to call the run-time library procedure LIB\$DO_COMMAND with the desired command as an argument. This procedure is specifically constrained to operate in user mode. The interceptor runs in kernel mode, and must return to its caller in kernel mode.

The goal is to get to user mode, perform the LIB\$DO_COMMAND functions and then restore the environment and return to the caller. The code could force its mode to user, perform its operation, and then return to kernel through a special change-mode-to-kernel dispatcher. An alternate mechanism was selected, however.

The process recovery routine of the interceptor issues a supervisor mode AST just before dismissing its own kernel mode AST. It is the supervisor mode AST that is responsible for altering DCL's course to the logout command procedure.

A supervisor mode AST was selected for the following reasons:

1. By raising mode as soon as possible, any bugchecks incurred are process fatal, rather than system fatal.
2. The convenience of SYS\$DCLCMII in altering change-mode dispatching.
3. It is preferable to run in the least privileged (highest) access mode as is feasible.
4. It is "natural" to deal with CLI data structures in supervisor mode, which is where the CLI customarily runs.

Running in the context of the supervisor AST, the I/O on the CLI's input channel is canceled so that there will be no conflict with outstanding read-with-prompts.

Then the AST establishes a user stack context (saving any that is in place). This is done since there is no guarantee that a user mode stack exists. Before access mode is raised to user, a change mode to supervisor handler is declared so that the mode can later be lowered back to supervisor.

The processor mode is then artificially raised to user by fabricating a PC/PSL pair and performing an REI. This provides the unusual situation of a supervisor mode AST running in user mode! A "@command command is sent to the CLI via the callback routine (SYS\$CLI) while in user mode.

The reason this user mode operation is considered still part of the supervisor AST is that ASTACT level field in the PCB is still set to supervisor. This level is used as a check to avoid spurious AST's. Consequently, the AST dispatcher still sees an active supervisor AST and protects it.

Note that the supported LIB\$DO_COMMAND procedure is not used. Instead the unsupported procedure, SYS\$CLI, which is used by LIB\$DO_COMMAND is used. This is due to the fact that LIB\$DO_COMMAND presumes a "usual" user mode image and performs a call to SYS\$EXIT to rundown the image as a final act. There may not be an image to rundown in our situation. If there is not, the execution of SYS\$EXIT will cause an "exit pending" flag to be set causing the next user image to exit immediately... an intolerable side-effect.

Since the call to SYS\$CLI is modeled after the supported procedure, there is little danger in using the unsupported service. If the CLI callback protocol is changed, it will be changed in LIB\$DO_COMMAND and the interceptor's code can be changed accordingly. Effectively, the procedure is supported "one step removed". Future versions may simply use LIB\$DO_COMMAND followed by the unconditional clearing of the exit pending bit in the PCB.

The supervisor AST cannot be dismissed while in user mode. Before changing to user mode, a change mode to supervisor handler was declared. We therefore perform a CHMS instruction with the appropriate code. If the code is not that expected, it is passed to any previously declared handler. Otherwise, the previous handler is formally reinstated, the stack is cleared of the CHMS arguments, the user stack is reset to its original context, and the supervisor mode AST is dismissed.

As soon as the supervisor mode AST is dismissed, the system-wide command file is run, as setup by the CLI callback routine.

3.4 Deleting the Process after Command File Execution.

Having executed the system-wide command file, there must be a way of deleting the process which does not cause the command file to be executed again. The prototype described in this paper sets a flag that the CLI call-back has been made for this process. (The flag resides in P1 space with the prototype's code.) If the flag has been set, it means that the command file operation has previously been initiated, and the interceptor returns to process deletion with an RSB instruction.

Note that a user could "pump" consecutive delete process requests against his process from another process. In a production version the command-file flag must be augmented by another which can be set

only from another image by someone with OPER or other suitable privilege. This image would be invoked as the last act of the command procedure. The image can then set the appropriate privilege protected flag, based on the history of the process (in kernel mode we can get any privilege we want). Only when this flag has been set will the process resume deletion. If the command-file flag is set, the deletion AST will be dismissed, but no attempt will be made to initiate the logout command procedure.

An operator's utility should also be developed to force deletion of a process, circumventing or aborting the logout procedures.

3.5 Image Residency

It is important that the rundown interceptor be in a permanent segment of memory which will survive across image activation and rundown. This eliminates P0 space since it is destroyed entirely at each image rundown.

The two candidates are P1 space and S0 space.

Image rundown causes all P1 space beyond the address contained at CTL\$GL_CTLBASVA to be deleted. Therefore, if the image is loaded by extending P1 space, then the value at CTL\$GL_CTLBASVA must be changed to point to the new end of P1 space. This makes the code permanently mapped to the process's P1 space.

Loading S0 space is simply a matter of allocating contiguous pages of paged or non-paged pool and loading the image. S0 space MUST be used if the system-wide VMS rundown interception vector is used.

3.6 Image Loading

The code to be executed is placed in P1 space by a P0 based image (/P0IMAGE qualifier used in conjunction with the link of the mapping image).

The create and map section system service, SYS\$CRMPSC can be used to map any file into memory. It can be mapped as a shareable or a private copy. This system service is used repeatedly by the image activator, SYS\$IMGACT to map image sections piecemeal. The easiest way to use this service is to load an image linked "/SYSTEM", i.e., there is no image header.

Since the interceptor must be strictly position-independent, it can not use general-mode addressing, or external run-time images. If any descriptors are used, they must be initialized at run-time.

You may also use SYS\$IMGACT to load the rundown interpreter. The image activator does not perform merged activations directly into P1, so a technique used by VMS to map the CLI into P1 space is employed. The image activator is invoked to merge the image into P0 space for sizing purposes. Having determined the size of the image, the P0 space is deleted and P1 is expanded the appropriate number of pages. The image activator is called again to explicitly map into the expanded address range in P1.

The advantages of SYS\$CRMPSC are:

1. It is a fully supported system service.

The disadvantages of SYS\$CRMPSC are:

1. The rundown interceptor must be strictly position independent.
2. External images can not be activated or accessed at run-time.

The advantages of SYS\$IMGACT are:

1. Non-PIC code is convertible to PIC by SYS\$IMGACT so descriptors and general mode addressing can be used freely. (This conversion is done via a call to a sister system service, SYS\$IMGFIX which is considered part of the image activator, though called separately.)
2. Run-time libraries may be used (cautiously).
3. Page protection is automatically handled.

The disadvantages of SYS\$IMGACT are:

1. It is not a supported system service. However, it is so ingrained in the basic architecture of VMS, that changes in functionality causing incompatibility with this application are highly unlikely. Changes in calling protocol are more likely.

An early version of the prototype used SYS\$CRMPSC system service. The PIC code requirement was unacceptably constraining.

The image activator is so powerful a tool, that its use is warranted in this particular instance, despite its lack of formal support.

To isolate image activator changes, a family of utility subroutines should be developed. Should the image activator protocol change, only this family of routines will have to be changed for maintenance.

!!! WARNING !!!

The prototype code supplied on the DECUS tape and described herein is strictly experimental and is supplied for demonstration purposes only. The code has known deficiencies and is not intended for production purposes. The code is intended to be used by system programmers as a "template" for studying the problem.

3.7 LOGINOUT Inadequacy

VMS's LOGINOUT image will not be usable for logout since it closes the process permanent files (SYS\$OUTPUT, etc.) before calling SYS\$EXIT from kernel mode (which eventually calls SYS\$DELPRC or SYS\$RUNDWN). Under this circumstance, the interceptor still gets control, but has no communication channels for running a logout command file.

Command files typically make use of these files. It is unreasonable to expect logout components to make no use of these standard files. Therefore logouts must be effected by an image which calls SYS\$DELPRC and leaves the files intact. This will provide a wholly normal process context in which the system-wide logout command file may run.

It is natural to ask, "What is given up?". The LOGINOUT image makes calls to security audit software, and does a couple of other minor operations.

The important thing is that these operations are all intrinsically optional. The LOGINOUT image can be bypassed very simply by an UNPRIVILEGED user by:

1. Issuing the DCL command "\$ STOP/ID=0", or
2. Calling SYS\$DELPRC from an image.

On the other hand, logout operations implemented through the proposed rundown interceptor cannot be bypassed by an unprivileged user.

4 Prototype Rundown Interceptor

A prototype rundown interceptor and loader has been successfully implemented using techniques described in the section on problem discussion and solutions.

The per-process vectors were selected for the prototype due to the accessibility for development purposes, and so that the code could be tested with others on the system. (Once a system-wide vector is initialized it is used for every process on the system). The use of the per-process vectors also provided a familiarity with the vector usage protocol which can be used as a model of a protocol for system-wide vector usage, if desired.

P1 space was selected for the prototype for the following reasons:

1. Ease of access.
2. Minimal perturbation to system during test.

The loader uses the image activator to load the interceptor into P1 space and sets up the per-process rundown dispatch vector to activate the rundown interceptor.

The interceptor assumes that logout is performed by a `SYSDDELPRC` system service call, rather than running the `LOGINOUT` image. (The DCL command `"STOP/ID=0"` equivalent). An undeletable process will result if normal logout is attempted.

Note that the interceptor in the prototype is activated from the image activator's dispatch area. When it dismisses the deletion AST, none of the remaining vectors are activated. This is not tolerable for a production version. If P1 dispatching is retained then the remaining vectors can be activated by a JSB indirect on the stack pointer, or they can be run and eliminated by a call to `SYSDRUNDWN` for user mode. If S0 dispatching is employed, the P1 vectors have all been called and completed by the time the interceptor is activated.

The prototype uses an overly simplistic mechanism for resuming process deletion after command file execution. See the section on resumption of deletion for details.

The prototype interceptor outputs messages showing significant events in its execution for demonstration purposes.

The interceptor works for:

1. A top-level process.
2. A spawned subprocess.
3. A top-level process which had deletion requested by any other process.

Please note that the message which marks resumption of supervisor mode will have to be moved to make the third scenario operate.

Thoroughly commented source and build files are supplied on the DECUS tape for this Symposium. Details on the implementation algorithms are included as well as procedures for building a "demo" model.

5 Path to a Production Version.

5.1 Program Residency

A number of factors point to the desirability of using S0 space rather than P1 space for the image.

The primary factor is that the P1 vectors are a part of the image activator's data area. Prudence suggests that we stay away from this area if there is another reasonable approach to accomplish the same thing. The use of P1 residency would require the use of the P1 vectors.

Another factor is that the `SYSDSPAWN` system service does not copy extensions to the parent process's address space into its own. The `SYSDSPAWN` service starts with a new shell, initializes it, and then, through a series of mailbox communications between the parent and spawned process, the logical name tables and CLI symbol tables are optionally copied. If the rundown interceptor were to be mapped in S0 space, no such problem exists.

A routine for mapping an image into S0 paged or non-paged pool must be developed. No tests have been made of loading S0 space with the image activator; however, it is anticipated that no major difficulty will be encountered in its use.

5.2 Vector Mechanism

This issue is related to the image residency problem. If the image is to reside in P1 space then the per-process vector must be used, since uniform mapping of P1 space across processes can NOT be presumed. On the other hand if S0 space is selected, then either vector can be used. However, per-process vectors are not copied to a spawned subprocess.

Use of the S0 vector is currently indicated. It is desirable to avoid sharing the image activator's data area. Per-process capabilities can be provided by keeping a list of per-process interception vectors as in the image activators per-image vectors. A set of utilities can be developed to manage these data areas.

5.3 Environmental Flexibility

Process deletion is invokable in a number of fashions, only a few of which have been tested. For example, the situation of the deletion of a disconnected process which has timed out must be investigated. Tests on the availability of input/output channels must be made and appropriate contingency action developed.

An inventory of process deletion scenarios must be made and appropriate special action (if any) developed for each case.

Some of the scenarios will require special considerations in order to accommodate them. For example, if a process is being deleted because of CPU limit time-out, one or more optional procedures can apply. On first encounter, one may artificially boost the time limit enough to initiate and finish logout procedures. On second encounter, we know that our estimate on the time for logout operations is wrong, or that there is something seriously wrong with the logout code or the machine itself -- an errorlog entry and continuation through deletion is in order.

5.4 Error Mode Analysis

The system must be revisited to implement error-mode detection and contingency operations (particularly in considering the various scenarios of process deletion as discussed above). In general, any error mode must proceed toward normal process deletion.

5.5 Program Structure

If the image activator is used, it will be possible to link at run-time to base functionality, allowing independent maintenance of functional subsystems, e.g., process rundown and image rundown functions.

5.6 System Programming Guidelines

The implementation of the rundown interceptor uses some of VMS's one-of-a-kind hooks. System programmers may need to use the VMS facilities used by this application. In order to do this a comprehensive set of system programming conventions and utilities must be developed to allow system programmers to achieve the effect of the VMS hooks used by this application. In fact, the resulting set of tools could make the use of those VMS facilities easier than without the application.

Protocols and conventions must be developed to coordinate the use of:

1. P1 and S0 Space
2. P1 and S0 Vectors
3. Per-process data conventions and utilities
4. Processor Register Conventions

5.7 Logout driver to replace LOGINOUT

A logout driver must be developed to emulate the user-interface of LOGINOUT. However, the new driver will not close process permanent I/O channels. Extended features can be developed, e.g., clear the screen as a final act leaving no print on the screen whatsoever.

The new driver will terminate the process via a call to SYS\$DELPRC.

Investigations into running LOGINOUT at the end of the command procedure will be made. The image could be protected from direct operation through ACL's specially constructed to allow access during login and post-logout-file operation only.

APPENDIX A

STRUCTURE OF RUNDOWN INTERCEPTOR

The following structure chart outlines the call structure of the rundown interceptor.


```

*****
***           Notes to           ***
***   Rundown Interceptor   ***
***           Structure Chart   ***
*****

```

Module descriptions are found in the code listings. Modules formatted "\$name" are standard VMS system services SYS\$name.

The SYS\$RUNDWN system service is not supported. It is shown here to describe its role.

The VMS_ASTDEL module is actually part of the VMS executive, the Asynchronous System Trap Delivery Mechanism. It is shown here to describe its role.

The SYS\$CLI service is unsupported. Its use is modeled after the LIB\$DO_COMMAND procedure of the VMS run-time library.

The DCL module shows the CLI's role in the command file activation. It represents the command interpreter.

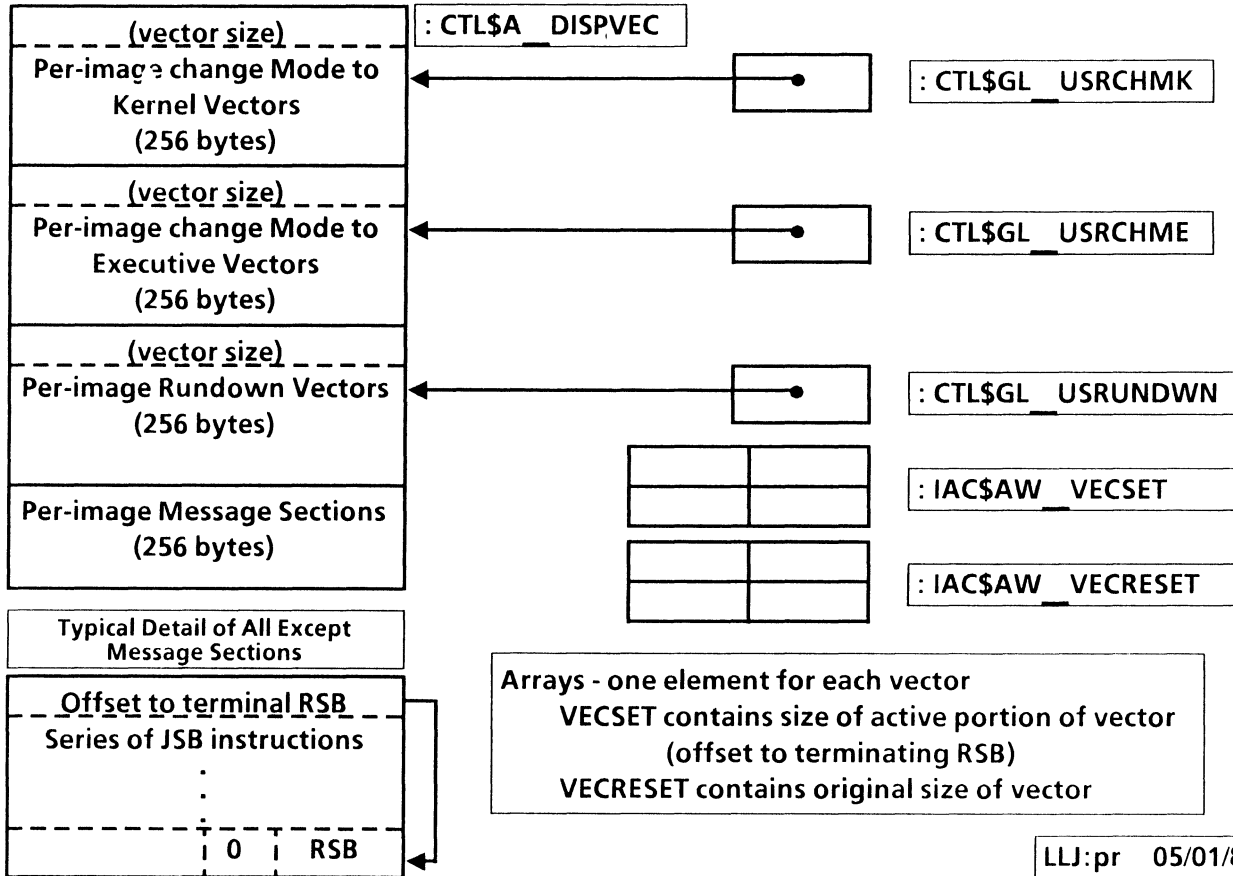
```

*****
*** Structure Chart Legend ***
*****

```

- A. Solid lines indicate activation via VMS Procedure Calling Standard (CALLx/RET), except for the activation of the command procedure by DCL which is the usual "@command-file" activation.
- B. Dashed lines indicate activation by a JSB protocol.
- C. The dash-dot line indicates the issue of the RET to the last active call frame by a procedure which was actually called under the JSB protocol.
- D. Triangular "hats" on modules indicate lexical inclusion within the subordinate, i.e., they are in the same compile module.
- E. \$DELPRC invocation (A) is the last act of system-wide logout command procedure.

VMS Dispatch Vector Data Structures



LLJ:pr 05/01/86

APPENDIX C
LISTINGS OF RUNDOWN INTERCEPTOR

```

0000 1 .TITLE EG_RUNDOWN_INTERCEPTOR
0000 2
0000 3 ; Prototype permanent per-process rundown routine.
0000 4
0000 5 ;++
0000 6 ; Module: EG_RUNDOWN_INTERCEPTOR
0000 7 ;
0000 8 ; Facility: EGRNDN
0000 9 ;
0000 10 ; Function: Intercept rundown of images and processes and
0000 11 ; dispatch for additional processing.
0000 12 ;
0000 13 ; Invocation Protocol: Called via JSB by SYS$SRUNDWN or SYS$DELPRC
0000 14 ; through P1 dispatch vector entry.
0000 15 ;
0000 16 ; Input Parameters:
0000 17 ;
0000 18 ; R4 - Address of current PCB.
0000 19 ; R7 - Access mode parameter to $SRUNDWN maximized with
0000 20 ; previous mode. (This parameter will be USER
0000 21 ; mode for image rundown, and KERNEL mode for
0000 22 ; process rundown.
0000 23 ; AP - Argument pointer existing when the $SRUNDWN system
0000 24 ; service was invoked
0000 25 ;
0000 26 ; Environment:
0000 27 ;
0000 28 ; Kernel Mode, IPL = 0 for image rundown
0000 29 ; Activated by SYS$SRUNDWN with R7 = PSL$C_USER
0000 30 ; or R7 = PSL$C_KERNEL
0000 31 ;
0000 32 ; Kernel Mode AST for process rundown
0000 33 ; Activated by SYS$DELPRC with R7 = PSL$C_KERNEL
0000 34 ;
0000 35 ; Code must reside in a permanent portion of P1 space or
0000 36 ; S0 space (paged or non-paged pool).
0000 37 ; Side Effects:
0000 38 ;
0000 39 ; Can cause execution in context of supervisor mode
0000 40 ; AST in the case of process deletion. Execution
0000 41 ; of the AST will result in system logout command file
0000 42 ; execution in normal user context.
0000 43 ;
0000 44 ; Status Returns:
0000 45 ;
0000 46 ; If an initiating AST is dismissed, then
0000 47 ; $$$_NORMAL is returned in R0.
0000 48 ; Else
0000 49 ; There is no status return.
0000 50 ; Endif
0000 51 ;
0000 52 ; Notes:
0000 53 ;
0000 54 ; Revision: 1.0 20-Sep-1985 Larry L. Johnson
0000 55 ; in collaboration with
0000 56 ; Liz Bellamy of DEC,
0000 57 ; Dallas Regional Office.

```

```
0000 52 ;  
0000 53 ; Language:      VAX/VMS Macro-32 Assembler.  
0000 60 ;  
0000 61 ; Required Macro Libraries:  
0000 62 ;     STARLET      The default VAX System Macro Library.  
0000 63 ;                   (Automatically scanned.)  
0000 64 ;     _IB        The master VAX System Macro Library.  
0000 65 ;  
0000 66 ;  
0000 67 ;     $CLIDEF     ;Definitions for SYSSCLI invocation.  
0000 68 ;     $IPLDEF     ;Definition of interrupt priority levels.  
0000 69 ;     $PHDDEF     ;Process header definitions.  
0000 70 ;     $PRODEF     ;Processor register symbols.  
0000 71 ;     $PCBDEF     ;Process control block definitions.  
0000 72 ;     $IACDEF     ;Image activator data areas.  
0000 73 ;     $IHDEF      ;Image header structure.  
0000 74 ;     $IHDEF     ;  
0000 75 ;     $IHDEF     ;  
0000 76 ;     $PSLDEF     ;Processor status longword constants.  
0000 77 ;  
0000 78 ; External Status References:  
0000 79 ;  
0000 80 ;     Facility Defined Status Values:      None  
0000 81 ;     VAX/VMS Common Run Time Library Status Values:  None  
0000 82 ;     VAX/VMS RMS Status Values:          None  
0000 83 ;     VAX/VMS System Service Status Values:      None  
0000 84 ;  
0000 85 ; External Data References:      None  
0000 86 ;  
0000 87 ; External Routine References:  
0000 88 ;  
0000 89 ;     Facility Defined Routines:          None  
0000 90 ;     VAX/VMS Common Run Time Library Routines:  None  
0000 91 ;     VAX/VMS RMS Routines:              None  
0000 92 ;  
0000 93 ;     VAX/VMS System Services:  
0000 94 ;  
0000 95 ;     $ASSIGN  
0000 96 ;     $CANCEL  
0000 97 ;     $CLI          (Not supported by DEC)  
0000 98 ;     $CMKRNL  
0000 99 ;     $DASSGN  
0000 100 ;     $DCLAST  
0000 101 ;     $DCLCMM  
0000 102 ;     $QIOW  
0000 103 ;  
0000 104 ;--  
0000 105
```

Rundown Macros

```

0000 107 .SUBTITLE Rundown Macros
0000 108
0000 109 ;.....
0000 110 ;... Local Macro Definitions .....
0000 111 ;.....
0000 112
0000 113 ; These macros are used for output of messages for demonstration
0000 114 ; purposes.
0000 115
0000 116 .MACRO PUT_TO_SYSOUTPUT MESSAGE_NAME
0000 117     BSBW     EGRNDN_OPEN_SYSOUTPUT_CHANNEL
0000 118     $QIOW_S -
0000 119             CHAN = SYSOUTPUT_CHANNEL, -
0000 120             FUNC = #IOS_WRITEVBLK, -
0000 121             P1  = "MESSAGE_NAME", -
0000 122             P2  = #"MESSAGE_NAME"_SZ, -
0000 123             P4  = #^X00000020
0000 124     BSBW     EGRNDN_CLOSE_SYSOUTPUT_CHANNEL
0000 125 .ENDM PUT_TO_SYSOUTPUT
0000 126
0000 127 .MACRO DECLARE_MESSAGE MESSAGE_NAME
0000 128     "MESSAGE_NAME":
0000 129 .ENDM DECLARE_MESSAGE
0000 130
0000 131 .MACRO END_MESSAGE MESSAGE_NAME
0000 132     MESSAGE_NAME"_SZ = _-"MESSAGE_NAME
0000 133 .ENDM END_MESSAGE
0000 134
0000 135 .MACRO DEFINE_MESSAGE MESSAGE_NAME, MESSAGE_TEXT
0000 136     DECLARE_MESSAGE MESSAGE_NAME
0000 137     .ASCII ""MESSAGE_TEXT""
0000 138     END_MESSAGE MESSAGE_NAME
0000 139 .ENDM DEFINE_MESSAGE
0000 140

```

Data for Rundown Main Procedure

```

0000 142 .SUBTITLE Data for Rundown Main Procedure
0000 143
0000 144 ;.....
0000 145 ;... Local Read-Only Data Allocations .....
0000 146 ;.....
0000 147
0000 148
00000000 149 .PSECT _EGRNDN_LOCAL_RO_DATA, -
0000 150 QUAD, CON, NOEXE, LCL, NOPIC, NOSHR, REL, NOWRT, NOVEC
0000 151
0000 152 ;Messages used for demonstration purposes.
0000 153
0000 154 DEFINE_MESSAGE RUNDOWN_MARKER_TEXT <Rundown Interceptor Activated.>
001E 155 DEFINE_MESSAGE KERNEL_MARKER_TEXT <Kernel Activation.>
0030 156 DEFINE_MESSAGE MARK_FIRST_PASS_TEXT <First Pass in kernel marked.>
004C 157 DEFINE_MESSAGE PROCESS_MARKED_FOR_DEL_TEXT <Process was marked for delete.>
006A 158 DEFINE_MESSAGE IMAGE_EXIT_FORCED_TEXT <Forcing image exit.>
0079 159 DEFINE_MESSAGE RETURN_FROM_KERNEL_TEXT <Return from kernel rundown.>
0093 160 DEFINE_MESSAGE AST_TEXT <Supervisor AST activated.>
00A1 161 DEFINE_MESSAGE USER_TEXT <Supervisor AST entered USER mode.>
00B2 162 DEFINE_MESSAGE RESUMPTION_OF_SUPER_MODE <Supervisor AST resumes Super mode.>
00F4 163
00F4 164 ;.....
00F4 165 ;... Local Writeable Data Allocations .....
00F4 166 ;.....
00F4 167
00F4 168 .PSECT _EGRNDN_LOCAL_RW_DATA, -
00000000 169 QUAD, CON, NOEXE, LCL, NOPIC, NOSHR, REL, WRT, NOVEC
0000 170
0000 171 ; Local Status Variable Allocations: None
0000 172
0000 173 ; Local Variable Allocations:
0000 174
0000 175 DECLARE_MESSAGE NONKERNEL_MARKER_TEXT
0000 176 .ASCII "Nonkernel Activation, Mode = "
0016 177 FORMATTED_MODE: .BYTE ^A"?
001E 178 END_MESSAGE NONKERNEL_MARKER_TEXT
001E 179
001E 180 ; Process status flags implemented here for now. Eventually
001E 181 ; destined for P1 Pointer Page or an "extended PCB" structure
001E 182 ; in non-paged pool.
001E 183
00000000 001E 184 PROCSTSFIELDS: .LONG 0
00000000 0022 185 EGPRCSTS_V_LOGOUTPND = 0
00000001 0022 186 EGPRCSTS_V_DISMISSAST = 1
0022 187

```

461

Rundown Dispatcher Main Procedure

```

0022 189 .SUBTITLE Rundown Dispatcher Main Procedure
0022 190
0022 191 ;.....
0022 192 ;... Executable .....
0022 193 ;.....
0022 194
0022 195 .PSECT _EGRNDN_CODE, -
00000000 196 QUAD, CON, EXE, LCL, NOPIC, NOSHR, REL, NOWRT, NOVEC
0000 197
0000 198 RUNDOWN_BEGIN:
0000 199
0000 200 ;*****
0000 201 ;***** ENVIRONMENT *****
0000 202 ;*
0000 203 ;* Kernel Mode *
0000 204 ;* (May or may not be running as kernel mode AST) *
0000 205 ;*
0000 206 ;*****
0000 207
0000 208 ;Output a marker message.
0000 209
0000 210 PUT_TO_SYSOUTPUT RUNDOWN_MARKER_TEXT
0020 211
0020 212 ;Determine if rundown requested for user or kernel mode.
0020 213
0020 214 ;If rundown has been requested for user mode then
0020 215 ; Perform rundown services for user mode.
0020 216
03 57 01 0020 217 Cmpl R7, #PSL$C_USER
00 05 12 0030 218 BNEQ 10$
004A 30 0032 219 BSBW EGRNDN_USER_RUNDOWN
00 37 11 0035 220 BRB 30$
0037 221 10$:
0037 222
0037 223 ;Else if rundown has been requested for kernel mode then
0037 224 ; Perform rundown services for kernel mode.
0037 225
00 57 01 0037 226 Cmpl R7, #PSL$C_KERNEL
00 32 12 003A 227 BNEQ 20$
0020 30 003C 228 BSBW EGRNDN_KERNEL_RUNDOWN
0020 00 11 003F 229 PUT_TO_SYSOUTPUT RETURN_FROM_KERNEL_TEXT
0000 00 11 0000 230 BRB 30$
000E 231 20$:
000E 232
000E 233 ;Else
000E 234 ; Do nothing. The meaning of requests to
000E 235 ; rundown for supervisor or executive modes
000E 236 ; is unknown... Allow routine to return to
000E 237 ; caller to let him do as he pleases.
000E 238
000E 239 ;Endif
000E 240 30$:
000E 241
000E 242 ;Return from this routine.
000E 243 ; Check to see if we are to make normal return (RSB) to
000E 244 ; caller, or to issue a RET (normally to dismiss a
000E 245 ; $DELPRC AST).
    
```

```

30-MAY-1986 13:23:34 VAX/VMS Macro V04-00 Page 6
27-APR-1986 22:25:59 PROTOTYPE_RUNDOWN_INTERCEPTOR (1)

Rundown Dispatcher Main Procedure

000E 246
000E 247
000E 248 ;If dismiss AST flag has not been set then
000E 249 ; Return to caller in normal fashion.
01 00000010*EF 01 24 000E 250
05 0076 251 BASC #EGPRCSTS_V_DISMISSAST, PROCSTSFLAGS, DISMISS_AST
0077 252 RSB
0077 253
0077 254 ;Else
0077 255 ; Software has detected that it has been invoked
0077 256 ; in AST context and has determined that the
0077 257 ; AST calling rundown is to be dismissed.
0077 258 DISMISS_AST:
0077 259
0077 260 ;*****
0077 261 ;***** ENVIRONMENT *****
0077 262 ;*
0077 263 ;* Kernel Mode AST *
0077 264 ;*
0077 265 ;*****
0077 266
50 00000000*8F 00 0077 267 MOVL #SS$_NORMAL, R0
04 007E 268 RET
007F 269
007F 270 ;Endif
007F 271

```

463

```

30-MAY-1986 13:23:34 VAX/VMS Macro V04-00 Page 7
27-APR-1986 22:25:59 PROTOTYPE_RUNDOWN_INTERCEPTOR (1)

Rundown for User Mode (Image Termination

007F 273 .SUBTITLE Rundown for User Mode (Image Termination)
007F 274
007F 275 ;User Mode Rundown Functions. Assumes the function is
007F 276 ; image rundown.
007F 277
007F 278 EGRNDN_USER_RUNDOWN:
007F 279
007F 280 ;*****
007F 281 ;***** ENVIRONMENT *****
007F 282 ;*
007F 283 ;* Kernel Mode *
007F 284 ;*
007F 285 ;*****
007F 286
007F 287 ;Calculate ASCII form of access mode for output message,
007F 288 ; and put it to SYS$OUTPUT.
007F 289
00000010*EF 07 30 01 007F 290 ADDB3 #^X30, R7, FORMATTED_MODE
0037 291 PUT_TO_SYSOUTPUT NONKERNEL_MARKER_TEXT
0034 292
0228 30 0034 293 BSBW OUTPUT_IMAGE_NAME
0037 294
0037 295 ;Specify normal return to caller of Rundown Facility
0037 296 ; and leave this routine.
0037 297
00000010*EF 02 04 0037 298 BICL2 #<10EGPRCSTS_V_DISMISSAST>, PROCSTSFLAGS
000E 299
05 003E 300 RSB
000F 301

```

Kernel RunDown (Logout)

```

003F 303 _SUBTITLE Kernel RunDown (Logout)
003F 304
003F 305 EGRNDN_KERNEL_RUNDOWN:
003F 306
003F 307 ;*****
003F 308 ;***** ENVIRONMENT *****
003F 309 ;*
003F 310 ;* Kernel Mode *
003F 311 ;*
003F 312 ;*****
003F 313
003F 314 PUT_TO_SYSOUTPUT KERNEL_MARKER_TEXT
003F 315
003F 316 ;#+
003F 317
003F 318 ;If process deletion is pending, then
003F 319
003F 320 BBS #PC3$V_DELPEN, PCB$L_STS(R4), 10$
003F 321 ERW PROCESS_RUNDOWN_END
003F 322 10$:
003F 323
003F 324 ;*****
003F 325 ;***** ENVIRONMENT *****
003F 326 ;*
003F 327 ;* Kernel Mode AST *
003F 328 ;*
003F 329 ;*****
003F 330
003F 331 ;If logout command procedure has not been run then
003F 332
003F 333 BBSS #EGPRCSTS_V_LOGOUTPND, PROCSTSFLAGS, -
003F 334 ACTIVATE_LOGOUT_FILE_END
003F 335
003F 336 ; Recover from process deletion by:
003F 337
003F 338 ; ...clearing the delete pending bit in
003F 339 ; the PCB at synch IPL, and...
003F 340
003F 341 SETIPL #IPL$_SYNCH
003F 342 BICL2 #PC3$M_DELPEN, PCB$L_STS(R4)
003F 343 SETIPL #0
003F 344
003F 345 ; ...marking the process status flags for dismissal
003F 346 ; of the kernel mode process deletion AST
003F 347 ; (i.e., abort the process deletion).
003F 348
003F 349 BISL2 #<1$EGPRCSTS_V_DISMISSAST>, PROCSTSFLAGS
003F 350
003F 351 ; Declare a supervisor mode AST to force the
003F 352 ; activation of the system-wide command file
003F 353 ; on the dismissal of this kernel mode AST.
003F 354
003F 355 $OCLAST_S -
003F 356 ASTADR = EGRNDN_VECTOR_TO_EGLOGOUT, -
003F 357 ACMODE = #PSL$C_SUPER
003F 358
003F 359 ;Else

```

464

Kernel Rundown (Logout)

```
011E 360
011E 361      ACTIVATE_LOGOUT_FILE_END:
011E 362
011E 363          ;Do nothing, allowing the process to continue
011E 364          ; to deletion since the command file has
011E 365          ; already been run.
011E 366
011E 367      ;Endif
011E 368
011E 369 ;Else
011E 370
011E 371 PROCESS_RUNDOWN_END:
011E 372
011E 373 ;*****
011E 374 ;***** ENVIRONMENT *****
011E 375 ;*
011E 376 ;*          Kernel Mode
011E 377 ;*          (May or may not be in AST context)
011E 378 ;*
011E 379 ;*****
011E 380
011E 381          ;Do nothing, the context of this call is not understood, if
011E 382          ; it is not a process deletion.
011E 383
011E 384 ;Endif
011E 385
011E 386 ;Return to caller
011E 387
05 011E 388      RSB
011E 389
```

```

011F 391 .SUBTITLE Force Logout Command File Execution
011F 392
011F 393 ;*****
011F 394 ; Force Logout command file execution.
011F 395 ;
011F 396 ; This is accomplished by calling the CLI callback routine. This
011F 397 ; routine must be executed in User Mode. This code is first entered
011F 398 ; through a supervisor mode AST. Since the CLI needs to be called
011F 399 ; from User mode, the supervisor mode AST builds a new PSL with
011F 400 ; mode = user and REIs to the user mode code. The user mode code
011F 401 ; calls the CLI with a command form of @command_procedure. Return to
011F 402 ; supervisor mode is accomplished by CHMS which is intercepted by
011F 403 ; a Supervisor change mode handler established by the Supervisor
011F 404 ; mode AST.
011F 405 ;
011F 406
011F 407 ;.....
011F 408 ;... Local Writeable Data Allocations .....
011F 409 ;.....
011F 410
00000022 411 .PSECT _EGRNDN_LOCAL_RW_DATA, -
0022 412 QUAD, CON, NOEXE, LCL, NOPIC, NOSHR, REL, WRT, NOVEC
0022 413
0022 414 ;PSL fabricated for switch to user mode.
0022 415
00000000 0022 416 NEW_PSL: .LONG 0
0026 417
0026 418 ;CLI Callback Request data block.
0026 419
0000007A 0026 420 CLI_REQ_BLOCK: .BLKB CLI$K_SRVDESC
007A 421
007A 422 ;Location of original supervisor change mode handler.
007A 423
00000000 007A 424 OLD_HANDLER: .LONG 0
007E 425
007E 426 ;Original user stack profile.
007E 427
00000000 007E 428 OLD_PR$USP: .LONG 0
00000000 0032 429 OLD_USER_CTL$AL_STACK: .LONG 0
00000000 0036 430 OLD_USER_CTL$AL_STACKLIM: .LONG 0
003A 431
003A 432 ;Local stack for user mode code.
003A 433
0000028A 003A 434 USER_STACK_BOTTOM: .BLKB ^x200
003A 435 USER_STACK:
00000200 020A 436 USER_STACK_SIZE = .-USER_STACK_BOTTOM
0000026A 020A 437 USER_STACK_ADDRESS: .ADDRESS USER_STACK
020E 438
020E 439 ;.....
023E 440 ;... Local Read-Only Data Allocations .....
028E 441 ;.....
028E 442
028E 443 .PSECT _EGRNDN_LOCAL_RO_DATA, -
000000F4 444 QUAD, CON, NOEXE, LCL, NOPIC, NOSHR, REL, NOWRT, NOVEC
00F4 445
00F4 446 ;Command activating system-wide logout command procedure.
00F4 447
    
```

466

```

00F4 448 EG_LOGOUT_COMMAND_STR:
00F4 449 .ASCII "@SYSTEM_WIDE_LOGOUT_COMMAND_FILE"
0100
0100
010C
00000020 0114 450 EG_LOGOUT_COMMAND_SIZE = .-EG_LOGOUT_COMMAND_STR
0114 451
0114 452 ;.....
0114 453 ;... Executable .....
0114 454 ;.....
0114 455
0114 456 .PSECT _EGRNDN_CODE, -
0000011F 457 QUAD, CON, EXE, LCL, NOPIC, NOSHR, REL, NOWRT, NOVEC
011F 458
011F 459 ;*****
011F 460 ;***** ENVIRONMENT *****
011F 461 ;*
011F 462 ;* Supervisor Mode AST *
011F 463 ;*
011F 464 ;*****
011F 465
0000 011F 466 EGRNDN_VECTOR_TO_EGLOGOUT: .WORD ^M<R9,R10>
0121 467
0121 468 ;Register Usage:
0121 469 ;
0121 470 ; R9 = Address of CLI callback request description block.
0121 471 ; R10 = Address of process permanent data region (PPD).
0121 472
0121 473 PUT_TO_SYSOUTPUT AST_TEXT
014E 474
014E 475 ;Cancel outstanding I/O on SYS$INPUT channel. (Can't write to
014E 476 ; SYS$OUTPUT terminal if there is outstanding read with prompt
014E 477 ; on the same device via SYS$INPUT)
014E 478
014E 479 ;Get address of Process Permanent Data region (PPD).
014E 480
5A 00J0J000*9F 9E 014E 481 MOVAB @#CTL$AG_CLIDATA, R10
0155 482
0155 483 ;Cancel I/O on the input channel recorded there.
0155 484
0155 485 $CANCEL_S -
0155 486 CHAN = PPD$W_INPCHAN(R10)
0161 487
0161 488 ;
0161 489 ; Set up user stack.
0161 490 ; (We may or may not have a valid user stack at this point. To
0161 491 ; assure a valid stack we will establish a local one, saving the
0161 492 ; context of any existing user stack to enable it to rundown
0161 493 ; normally).
0161 494 ;
0161 495
0161 496 $CMKRNLS -
0161 497 ROUTIN = SETUP_USER_STACK
0170 498
0170 499 ;
0170 500 ; Declare a supervisor change mode handler so we can get back from user mode
0170 501 ;
0170 502

```

467

```

0170 503 $DCLCMH_S -
0170 504   ADDRES=SUPER_HANDLER, -
0170 505   PRVHND=OLD_HANDLER
0105 506
0105 507 ;
0105 508 ; Force mode to USER in order to make call to CLI callback routine.
0105 509 ;
0105 510   ; Get the current PSL and force its modes to user.
0105 511   ; Push the PSL and the PC of continuation on the stack
0105 512   ; and REI to change the mode.
0105 513
0105 514   MOVPSL NEW_PSL
0105 515   INSV #PSL$C_USER,#PSL$V_CURMOD,#PSL$S_CURMOD,NEW_PSL
0105 516   INSV #PSL$C_USER,#PSL$V_PRVMOD,#PSL$S_PRVMOD,NEW_PSL
0000022*EF 02 10 03 F0 0108 517   PUSHL NEW_PSL
0000022*EF 02 16 03 F0 0109 518   PUSHAL USER_CODE
0000022*EF 02 1A 03 DF 01A3 519   REI
01AA 520
01AA 521 ;*****
01AA 522 ;***** ENVIRONMENT *****
01AA 523 ;*
01AA 524 ;* SUPERVISOR mode AST lowered to USER mode.
01AA 525 ;*
01AA 526 ;*****
01AA 527
01AA 528 ;
01AA 529 ; User mode code - Call the CLI to execute our command procedure then
01AA 530 ; return back to supervisor with a CHM5
01AA 531 ;
01AA 532
01AA 533 USER_CODE:
01AA 534
01AA 535   PUT_TO_SYSOUTPUT USER_TEXT
0107 536
0107 537 ;
0107 538 ; Setup CLI request block to point to the command we want to execute.
0107 539 ;
0107 540 ; This code is modeled after the LIB$DO_COMMAND procedure which
0107 541 ; could not be used since it issues a call to SYS$EXIT as a last
0107 542 ; act. We may not have an outstanding image to rundown which would
0107 543 ; cause a "trap" for the next image activation, forcing it out
0107 544 ; immediately. When DCL executes the first USER image via the
0107 545 ; command file, it will rundown the image if necessary.
0107 546 ;
0107 547
0107 548   MOVAL CLI_REQ_BLOCK, R9
59 0000026*EF 0E 0107 549   MOVB #CLI$K_CLISERV, CLI$B_RQTYPE(R9)
01 09 05 30 01E1 550   MOVW #5, CLI$W_SERVCOD(R9)
01 09 05 30 01E5 551   MOVW #EG_LOGOUT_COMMAND_SIZE, CLI$Q_RQDESC(R9)
0C A9 00000F4*EF 0E 01E9 552   MOVAL EG_LOGOUT_COMMAND_STR, CLI$Q_RQDESC+4(R9)
01F1 553
01F1 554 ;Callback the CLI.
01F1 555
01F1 556   PUSHAL CLI_REQ_BLOCK
0000000*GF 01 FB 01F7 557   CALLS #1, G^SYS$CLI
01FE 558
01FE 559

```

```

01FE 560 ; Go back to supervisor mode via our change mode handler.
01FE 561 ;
FFFF 3F 3E 01FE 562 CHMS #-1
0202 563 ;
0202 564 ;*****
0202 565 ;***** ENVIRONMENT *****
0202 566 ;*
0202 567 ;* SUPERVISOR mode AST *
0202 568 ;*
0202 569 ;*****
0202 570 ;
0202 571 ; Supervisor Change Mode Handler
0202 572 ;
0202 573
0202 574 SUPER_HANDLER:
0202 575
0202 576 ;
0202 577 ; If change mode code is not ours (i.e., not negative) then
0202 578 ; Transfer operation to the original handler.
0202 579 ;
0202 580 TSTL (SP)
0204 581 BLSS 10$
0000007A*FF 17 0206 582 JMP @OLD_HANDLER
020C 583 10$:
020C 584
020C 585 ;
020C 586 ; Else
020C 587 ;
020C 588
020C 589 ;Output a marker message
020C 590
020C 591 PUT_TO_SYSOUTPUT RESUMPTION_OF_SUPER_MODE
0239 592
0239 593 ; Reinststate the original supervisor change mode handler,
0239 594 ; we don't need ours any longer.
0239 595
0239 596 $DCLCHM_S -
0239 597 ADDRES = @OLD_HANDLER
024A 598
024A 599 ;Reset the stack to get rid of the stuff put
024A 600 ; there by our change mode instruction.
024A 601 ; (Three longwords: the change mode code, PC and PSL).
024A 602
024A 603 MOVAL 12(SP),SP
024E 604
024E 605 ; Restore user stack. (Just in case there was an
024E 606 ; active image which requires rundown and therefore
024E 607 ; should be left with a valid user stack context).
024E 608
024E 609 $CMKRNLS -
024E 610 ROUTIN = RESTORE_USER_STACK
0250 611
0250 612 ; Dismiss supervisor mode AST
0250 613
0250 614 RET
0250 615
0250 616 ;*****
    
```


470

```

025E 818 .SUBTITLE      EGRNDN User Stack Utilities.
025E 819 ;*****
025E 820 ; EGRNDN User Stack Routines.
025E 821 ;
025E 822 ; Kernel Mode routines to:
025E 823 ;
025E 824 ; 1. Setup a user stack, saving the original stack context
025E 825 ; 2. Restore original stack context saved in the setup of
025E 826 ; the new stack area.
025E 827 ;
025E 828 ; The user stack context consists of the following three data:
025E 829 ;
025E 830 ; 1. The user stack processor register (PR$USP).
025E 831 ; 2. The stack pointer in the P1 pointer page (@#CTL$AL_STACK+12,
025E 832 ; the last element in a four element array.)
025E 833 ; 3. The stack size in the P1 pointer page (@#CTL$AL_STACKLIM+12,
025E 834 ; the last element in a four element array.)
025E 835 ;-----
025E 836 ; Kernel mode routine to save original user stack context and
025E 837 ; establish a new one.
025E 838 ;
0010 025E 840 SETUP_USER_STACK:      .WORD  ^M<R4>
0260 841
0260 842 ;Register Usage:
0260 843 ;
0260 844 ; R4 = Usual current PCB address of kernel mode routine.
0260 845
0260 846 ;Save original stack context.
0260 847
00000070*EF 03 0260 848 MFPD  #PR$USP, OLD_PR$USP
00000000*9F 00 0267 849 MOVL  @#CTL$AL_STACK+12, OLD_USER_CTL$AL_STACK
00000000*9F 00 0272 850 MOVL  @#CTL$AL_STACKLIM+12, OLD_USER_CTL$AL_STACKLIM
0270 851
0270 852 ; Establish new user stack in local storage by appropriately
0270 853 ; setting the same three items.
0270 854
00000000*9F 03 0270 855 MTPD  USER_STACK_ADDRESS, #PR$USP
00000000*9F 00 0284 856 MOVAL USER_STACK, @#CTL$AL_STACK+12
00000000*9F 00 028F 857 MOVL  #USER_STACK_SIZE, @#CTL$AL_STACKLIM+12
029A 858
004 029A 859 RET
029B 860
029B 861 ;-----
029B 862 ; Kernel Mode routine to restore saved context of user stack
029B 863 ;
0010 029B 864
029B 865 RESTORE_USER_STACK:      .WORD  ^M<R4>
029D 866
029D 867 ;Register Usage:
029D 868 ;
029D 869 ; R4 = Usual current PCB address of kernel mode routine.
029D 870
00000070*EF 03 029D 871 MTPD  OLD_PR$USP, #PR$USP
00000000*9F 00 02A4 872 MOVL  OLD_USER_CTL$AL_STACK, @#CTL$AL_STACK+12
00000000*9F 00 02A8 873 MOVL  OLD_USER_CTL$AL_STACKLIM, @#CTL$AL_STACKLIM+12
02BA 874

```

```

04 02BA 675 RET
02BA 676
02BB 677 ;*****

```

EGRNDN I/O Utilities.

```

0238 679 .SUBTITLE      EGRNDN I/O Utilities.
0238 680 ;*****
0238 681 ; EGRNDN I/O utilities.
0238 682
0238 683 ;.....
0238 684 ;... Local Read-Only Data Allocations .....
0238 685 ;.....
0238 686
00000114 687 .PSECT  _EGRNDN_LOCAL_RO_DATA, -
0114 688          QUAD, CON, NOEXE, LCL, NOPIC, NOSHR, REL, NOWRT, NOVEC
0114 689
0114 690 ; Descriptor for notification device.
0114 691
55 4F 24 53 59 53 0000011C 010E0000 0114 692 SYSOUTPUT_DEVICE:      .ASCID  "SYS$OUTPUT"
54 55 50 54 0122
0126 693
0126 694 ;.....
0126 695 ;... Local Writeable Data Allocations .....
0126 696 ;.....
0126 697
0126 698 .PSECT  _EGRNDN_LOCAL_RW_DATA, -
0000023E 699          QUAD, CON, NOEXE, LCL, NOPIC, NOSHR, REL, WRT, NOVEC
023E 700
023E 701 ;Channel for terminal communication. (This will not work properly
023E 702 ; for a batch job).
023E 703
0000 023E 704 SYSOUTPUT_CHANNEL:      .WORD  0
023E 705
023E 706 ;.....
023E 707 ;... Executable .....
023E 708 ;.....
023E 709
023E 710 .PSECT  _EGRNDN_CODE, -
00000238 711          QUAD, CON, EXE, LCL, NOPIC, NOSHR, REL, NOWRT, NOVEC
0238 712
0238 713 ;-----
0238 714 ; Open a channel for rundown messages for demonstration
0238 715 ; purposes.
0238 716
0238 717 EGRNDN_OPEN_SYSOUTPUT_CHANNEL:
0238 718
0238 719          $ASSIGN_S -
0238 720              DEVNAM = SYSOUTPUT_DEVICE,-
0238 721              CHAN = SYSOUTPUT_CHANNEL,-
0238 722              ACMODE = #PSL$C_KERNEL
05 0202 723          RSB
0203 724
0203 725 ;-----
0203 726 ; Close the rundown message channel.
0203 727
0203 728 EGRNDN_CLOSE_SYSOUTPUT_CHANNEL:
0203 729
0203 730          @DASSGN_S -
0203 731              CHAN = SYSOUTPUT_CHANNEL
05 02E1 732          RSB
02E2 733
02E2 734 ;*****

```

```

02E2 735
02E2 736 ;.....
02E2 737 ;... Local Read-Only Data Allocations .....
02E2 738 ;.....
02E2 739
00000126 740 .PSECT _EGRNDN_LOCAL_RO_DATA, -
0126 741 QUAD, CON, NOEXE, LCL, NOPIC, NOSHR, REL, NOWRT, NOVEC
0126 742
20 00 20 65 67 61 6D 49 0126 743 IMAGE_NAME_TITLE: .ASCII /Image = /
00000003 012E 744 IMAGE_NAME_TITLE_SZ = .-IMAGE_NAME_TITLE
012E 745
012E 746 ;.....
012E 747 ;... Executable .....
012E 748 ;.....
012E 749
012E 750 .PSECT _EGRNDN_CODE, -
000002E2 751 QUAD, CON, EXE, LCL, NOPIC, NOSHR, REL, NOWRT, NOVEC
02E2 752
02E2 753 OUTPUT_IMAGE_NAME:
02E2 754
0000000C 02E2 755 OUTPUT_IMAGE_NAME_MASK = ^MCR2,R3>
0C 8B 02E2 756 PUSHR #OUTPUT_IMAGE_NAME_MASK
02E4 757
52 000000C*9F 00 02E4 758 MOVL @#MMG$IMGHDRBUF, R2 ;Address of image header buffer.
53 06 A2 3C 02E8 759 MOVZWL IH0$W_IMGIDOFF(R2), R3 ;Offset to image id section
52 53 C0 02EF 760 ADDL2 R3, R2 ;Address of image id section
52 62 0E 02F2 761 MOVAL IH1$I_IMGNAM(R2), R2 ;Address of counted string.
53 82 9A 02F5 762 MOVZBL (R2)+, R3 ;Address of count in R3 leaving
02FE 763 ; address of string in R2.
FFC0 30 02F8 764 BSBW EGRNDN_OPEN_SYSOUTPUT_CHANNEL
02FB 765 $QIOW_S -
02FB 766 CHAN = SYSOUTPUT_CHANNEL, -
02FB 767 FUNC = #IO$WRITEVBLK, -
02FB 768 P1 = IMAGE_NAME_TITLE, -
02FB 769 P2 = #IMAGE_NAME_TITLE_SZ, -
02FB 770 P4 = #^X00000024
0322 771
0322 772 $QIOW_S -
0322 773 CHAN = SYSOUTPUT_CHANNEL, -
0322 774 FUNC = #IO$WRITEVBLK, -
0322 775 P1 = (R2), -
0322 776 P2 = R3, -
0322 777 P4 = #^X0000002B
FF8B 30 0345 778 BSBW EGRNDN_CLOSE_SYSOUTPUT_CHANNEL
0348 779
0C 8A 0348 780 POPR #OUTPUT_IMAGE_NAME_MASK
05 034A 781 RSB
034B 782
034B 783 ;*****
034B 784 RUNDOWN_END:
034B 785
034B 786 .END RUNDOWN_BEGIN

```

472

```

0000 1 .TITLE MERGE_SHAREABLE_IMAGE_INT0_P1
0000 2
0000 3 .SUBTITLE Introduction
0000 4 ;++
0000 5 ;
0000 6 ; Module: MERGE_SHAREABLE_IMAGE_INT0_P1
0000 7 ;
0000 8 ; Facility: EGP1
0000 9 ;
0000 10 ; Function: Permanently merge a shareable image into P1
0000 11 ; space, returning the image header buffer and
0000 12 ; the image transfer array to caller.
0000 13 ;
0000 14 ; Invocation Protocol: VAX/VMS Procedure Calling Standard.
0000 15 ;
0000 16 ; CALL MERGE_SHAREABLE_IMAGE_INT0_P1
0000 17 ; (IMAGE_FILE_NAME, IMAGE_TRANSFER_ARRAY, LOWEST_ADDRESS_LOADED,
0000 18 ; HIGHEST_ADDRESS_LOADED, IMAGE_HEADER_BUFFER, STATUS)
0000 19 ;
0000 20 ; IMAGE_FILE_NAME ---> Descriptor.
0000 21 ; Image file name.
0000 22 ; IMAGE_TRANSFER_ARRAY <--- Longword (3)
0000 23 ; Array of transfer
0000 24 ; addresses
0000 25 ; LOWEST_ADDRESS_LOADED <--- Longword.
0000 26 ; Address of beginning
0000 27 ; of loaded image.
0000 28 ; HIGHEST_ADDRESS_LOADED <--- Longword.
0000 29 ; Address of end of
0000 30 ; loaded image.
0000 31 ; IMAGE_HEADER_BUFFER <--- Longword.
0000 32 ; Address of image header
0000 33 ; buffer
0000 34 ; STATUS <--- Longword.
0000 35 ; Status of operation.
0000 36 ;
0000 37 ; Status Returns:
0000 38 ;
0000 39 ; SSS_NORMAL
0000 40 ;
0000 41 ; Any other system service status (to be considered
0000 42 ; a non-recoverable error).
0000 43 ;
0000 44 ; Notes:
0000 45 ;
0000 46 ; 1. Error analysis code is simplistic for the purposes of
0000 47 ; this early revision. Error handling to be
0000 48 ; extensively revisited for production version.
0000 49 ;
0000 50 ; Revision: 1.0 20-Sep-1985 Larry L. Johnson
0000 51 ;
0000 52 ; Language: VAX/VMS Macro-32 Assembler.
0000 53 ;
0000 54 ; Required Macro Libraries:
0000 55 ;
0000 56 ; STARLET The default VAX System Macro Library.
0000 57 ; (Automatically scanned.)

```

Introduction

```

0000 58 ; LIB The master VAX System Macro Library.
0000 59 ;
0000 60
0000 61 $PSLDEF ;Define processor status longword constants
0000 62 $IACDEF ;Define image activator constants
0000 63 $IHDEF ;Define image header details so
0000 64 $IHADEF ; transfer array can
0000 65 ; be returned
0000 66
0000 67 ; External Status References:
0000 68 ;
0000 69 ; Facility Defined Status Values: None
0000 70 ; VAX/VMS Common Run Time Library Status Values: None
0000 71 ; VAX/VMS RMS Status Values: None
0000 72 ;
0000 73 ; VAX/VMS System Service Status Values:
0000 74 ;
0000 75 ; SSS_NORMAL
0000 76 ;
0000 77 ; External Data References:
0000 78 ;
0000 79 ; From SYSSYSTEM:SYS.STB
0000 80 ;
0000 81 ; CTL$GL_CTLBASVA Base of permanent portion of
0000 82 ; P1 space.
0000 83 ;
0000 84 ; External Routine References:
0000 85 ;
0000 86 ; Facility Defined Routines: None
0000 87 ; VAX/VMS Common Run Time Library Routines: None
0000 88 ; VAX/VMS RMS Routines: None
0000 89 ;
0000 90 ; VAX/VMS System Services:
0000 91 ;
0000 92 ; SYS$CMKRNL
0000 93 ; SYS$DELTA
0000 94 ; SYS$EXPREG
0000 95 ; SYS$IMGACT (Not supported by DEC)
0000 96 ; SYS$IMGFIX (Not supported by DEC)
0000 97 ;
0000 98 ;--
0000 99

```

```

0000 101 .SUBTITLE Module Data
0000 102
0000 103 ;.....
0000 104 ;... Argument List Offsets .....
0000 105 ;.....
0000 106
0000 107 ;Argument List Offset Definitions:
0000 108
0000 109
00000004 0000 109 IMAGE_FILE_NAME = 4
00000008 0000 110 IMAGE_TRANSFER_ARRAY = 8
0000000C 0000 111 LOWEST_ADDRESS_LOADED = 12
00000010 0000 112 HIGHEST_ADDRESS_LOADED = 16
00000014 0000 113 IMAGE_HEADER_BUFFER = 20
00000018 0000 114 STATUS = 24
0000 115
0000 116 ;.....
0000 117 ;... Local Read-Only Data Allocations .....
0000 118 ;.....
0000 119
0000 120
00000000 0000 121 .PSECT _EGP1_LOCAL_RO_DATA, -
0000 122 QUAD, CON, NOEXE, LCL, NOPIC, NOSHR, REL, NOWRT, NOVEC
0000 123
0000 124 ;.....
0000 125 ;... Local Writeable Data Allocations .....
0000 126 ;.....
0000 127
0000 128 .PSECT _EGP1_LOCAL_RW_DATA, -
00000000 0000 129 QUAD, CON, NOEXE, LCL, NOPIC, NOSHR, REL, WRT, NOVEC
0000 130
0000 131 ; Local Status Variable Allocations: None
0000 132
0000 133 ; Local Variable Allocations:
0000 134
0000 135 ;Storage for region to map arguments and return of actual
0000 136 ; region mapped by $IMGACT
0000 137
0000 138 ; Used as argument to select PO space expansion by image activator
0000 139 ; for purpose of sizing the mapped image.
0000 140
0000 141 PO_REGION_TO_BE_ADDED:
00000000 0000 142 PO_START_ADDR_TO_BE_ADDED: .LONG 0
00000000 0004 143 PO_END_ADDR_TO_BE_ADDED: .LONG 0
0000 144
0000 145 ; Returned by the image activators merge into PO space and used
0000 146 ; to delete the address range having obtained the image size.
0000 147
0000 148 PO_REGION_ACTUALLY_ADDED:
0000 149 PO_REGION_TO_BE_DELETED:
00000000 0008 150 PO_START_ADDR_ACTUALLY_ADDED: .LONG 0
00000000 000C 151 PO_END_ADDR_ACTUALLY_ADDED: .LONG 0
0000 152
0000 153 ; Results of the deletion of the temporary mapping into PO region.
0000 154
0000 155 PO_REGION_ACTUALLY_DELETED:
00000000 0010 156 PO_START_ADDR_ACTUALLY_DELETED: .LONG 0
00000000 0014 157 PO_END_ADDR_ACTUALLY_DELETED: .LONG 0
    
```

Module Data

```

0018 158
0018 159 ; Results of expansion of P1 program region.
0015 160
0012 161 P1_REGION_ACTUALLY_ADDED:
00000000 0018 162 P1_START_ADDR_ACTUALLY_ADDED: .LONG 0
00000000 001C 163 P1_END_ADDR_ACTUALLY_ADDED: .LONG 0
0020 164
0020 165 ; Mapping range argument for image activator.
0020 166
0020 167 P1_IMAGE_EXTENT_TO_BE_LOADED:
00000000 0020 168 P1_IMAGE_INTENDED_START_ADDR: .LONG 0
00000000 0024 169 P1_IMAGE_INTENDED_END_ADDR: .LONG 0
0028 170
0028 171 ; Results of image activators mapping into P1 space.
0028 172
0028 173 P1_IMAGE_EXTENT_ACTUALLY_LOADED:
00000000 0028 174 P1_IMAGE_ACTUAL_START_ADDR: .LONG 0
00000000 002C 175 P1_IMAGE_ACTUAL_END_ADDR: .LONG 0
0030 176
0030 177 ; A place for the image activator to put the image header information
0030 178 ; during the merge into P0 for sizing purposes.
0030 179
00000230 0030 180 P0_IMAGE_HEADER_BUFFER: .BLKB ^x200
0230 181

```

Main Module

27-APR-1986 11:46:11 MERGE_SHAREABLE_IMAGE_INT0_P1 (1)

```

0230 183 .SUBTITLE Main Module
0230 184
0230 185 ;.....
0230 186 ;... Executable .....
0230 187 ;.....
0230 188
0230 189 .PSECT _EGP1_CODE, -
00000030 190     QUAD, CON, EXE, LCL, NOPIC, NOSHR, REL, NOWRT, NOVEC
0000 191
01e0 0000 192 .ENTRY MERGE_SHAREABLE_IMAGE_INT0_P1, ^M<R5,R6,R7,R8>
0002 193
0002 194 ; Register Usage:
0002 195 ;
0002 196 ;     R5 - Number of pages to be added to P1 for interceptor image.
0002 197 ;     R6 - Address of image header of merged image.
0002 198 ;     R7 - Address of transfer array within image header.
0002 199 ;     R8 - Address of argument image transfer array for
0002 200 ;           auto-incremented transfer of array to caller.
0002 201
0002 202 ;Initialize output arguments.
0002 203
00 5c 04 0002 204     CLRL    @HIGHEST_ADDRESS_LOADED(AP)
0c 5c 04 0005 205     CLRL    @LOWEST_ADDRESS_LOADED(AP)
1e 5c 04 0008 206     CLRL    @STATUS(AP)
0003 207
0006 208
0008 209 ;Do a merged activation into P0 in order to size the address
000e 210 ; space.
000e 211
00000000*EF 7c 000c 212     CLRQ    PO_REGION_TO_BE_ADDED
0011 213
0011 214     $IMGACT_S -
0011 215     NAME = @IMAGE_FILE_NAME(AP), -
0011 216     HDRBUF = PO_IMAGE_HEADER_BUFFER, -
0011 217     IMGCTL = #<IACSM_MERGE!IACSM_EXPREG>, -
0011 218     INADR = PO_REGION_TO_BE_ADDED, -
0011 219     RETADR = PO_REGION_ACTUALLY_ADDED
0035 220
00 50 03 0035 221     BLBS    R0, 10$
0091 31 0038 222     BRW     COMMON_EXIT
0038 223
003f 224
0038 225     ;Perform the image fix to be sure that fixup vector
0038 226     ; work area is cleared before the P1 activation.
0038 227
0038 228     $IMGFIX_S
0042 229
0042 230     ;Unmap the image, it is no longer needed... we have
0042 231     ; the size information we want.
0042 232
0042 233     $DELTVL_S -
0042 234     INADR = PO_REGION_TO_BE_DELETED,-
0042 235     RETADR = PO_REGION_ACTUALLY_DELETED
0057 236
0057 237 ;Calculate the size in pages of the image in P0 space
0057 238
00000000*EF 00000003*EF 03 0057 239     SUBL3   PO_START_ADDR_ACTUALLY_ADDED, -

```


Main Module

```

    55      0062 240      PO_END_ADDR_ACTUALLY_ADDED, -
           0063 241      R5 ;Number of bytes less one.
           0065 242
    55  D6 0063 243      INCL  R5 ;Number of bytes.
           0065 244
    55  00000200 3F  C6 0065 245      DIVL2 #^X200, R5 ;Number of pages
           006C 246
           006C 247 ;Add region at end of P1-space to receive image.
           006C 248
           006C 249      $EXPREG_S -
           006C 250      REGION = #1, -
           006C 251      PAGCNT = R5, -
           006C 252      RETADR = P1_REGION_ACTUALLY_ADDED, -
           006C 253      ACMODE = #PSL$C_USER
           007F 254
           007F 255      ;The image activator requires mapping from low to
           007F 256      ; high addresses. In P1 space the end addresses
           007F 257      ; are higher than the start addresses for regions
           007F 258      ; specified in the order of natural growth.
           007F 259
    00000024*EF  00000018*EF  00 007F 260      MOVL  P1_START_ADDR_ACTUALLY_ADDED, -
           008A 261      P1_IMAGE_INTENDED_END_ADDR
           008A 262
    00000020*EF  0000001C*EF  00 008A 263      MOVL  P1_END_ADDR_ACTUALLY_ADDED, -
           0095 264      P1_IMAGE_INTENDED_START_ADDR
           0095 265
           0095 266 ;Map the image into the previously created
           0095 267 ; address range
           0095 268
           0095 269      ;This mapping originally done in separate routine to
           0095 270      ; facilitate execution in inner access mode when load
           0095 271      ; technique used SYS$CRMPSC rather than SYS$IMGACT.
           0095 272      ;The partitioning is left in anticipation of generalization
           0095 273      ; of this module to perform a selectable merged load
           0095 274      ; into S0 paged and S0 non-paged as well as P1.
           0095 275
    000000E1*EF  6C  FA 0095 276      CALLG  (AP), MAP_IMAGE_INT0_P1
           009C 277
           20 50  E9 009C 278      BLBC  R0, COMMON_EXIT
           009F 279
           009F 280 ;Extract the transfer array from the image header and
           009F 281 ; and return it.
           009F 282
    56  14 AC  00 009F 283      MOVL  IMAGE_HEADER_BUFFER(AP), R6 ;Address of image header
           00A3 284      ; buffer descriptor
    56  04 06  00 00A3 285      MOVL  @4(R6), R6 ;Address of image header
    57  02 A6  3C 00A7 286      MOVZWL IHD$W_ACTIVOFF(R6), R7 ;Offset to transfer
           00AB 287      ; vector array
    57  56  C0 00AB 288      ADDL2  R6, R7 ;Address of transfer vector
           00AE 289
           00AE 290      ;Return the three vectors.
           00AE 291
    58  0P AC  00 00AE 292      MOVL  IMAGE_TRANSFER_ARRAY(AP), R8
           00B2 293
    58  67  00 00B2 294      MOVL  IHA$L_TFRADR1(R7), (R8)+
    58  04 A7  00 00B5 295      MOVL  IHA$L_TFRADR2(R7), (R8)+
    60  08 A7  00 00B9 296      MOVL  IHA$L_TFRADR3(R7), (R8)
    
```

478

```

0030 297
0030 298 ;Make the P1 mapping permanent, so it will not disappear across
0030 299 ; any image activations.
0030 300
0030 301 $CMKRNL_S -
0030 302 ROUTIN = MAKE_P1_IMAGE_PERMANENT
00CC 303
00CC 304 COMMON_EXIT:
00CC 305
00CC 306 ;Return address range mapped for the image.
00CC 307
UC 5C 0000020*EF DD 00CC 308 MOVL P1_IMAGE_ACTUAL_START_ADDR, @LOWEST_ADDRESS_LOADED(AP)
10 5C 000002C*EF DD 0004 309 MOVL P1_IMAGE_ACTUAL_END_ADDR, @HIGHEST_ADDRESS_LOADED(AP)
000C 310
000C 311 ;... and the status.
000C 312
16 5C 50 DD 000C 313 MOVL R0, @STATUS(AP)
00E0 314
04 00E0 315 RET
00E1 316

```

```

00E1 318 .SUBTITLE Module Utilities.
00E1 319
00E1 320 ;*****
00E1 321
00E1 322
00C4 00E1 323 MAP_IMAGE_INTU_P1: .WORD ^M<R2>
00E3 324
00E3 325 ;Get the address of the image buffer descriptor.
00E3 326
52 14 AC DD 00E3 327 MOVL IMAGE_HEADER_BUFFER(AP), R2
00E7 328
00E7 329 ;Map the image into P1.
00E7 330
00E7 331 $IMGACT_S -
00E7 332 NAME = @IMAGE_FILE_NAME(AP), -
00E7 333 HDRBUF = 34(R2), -
00E7 334 INADR = P1_IMAGE_EXTENT_TO_BE_LOADED, -
00E7 335 RETADR = P1_IMAGE_EXTENT_ACTUALLY_LOADED
C103 336
07 50 E9 0103 337 BLBC R0, P1_MAPPING_RETURN
0108 338
0106 339 ;Apply the fixup vectors, simulating PIC.
0105 340
0108 341 $IMGFIX_S
0112 342
0112 343 P1_MAPPING_RETURN:
0112 344
04 0112 345 RET
0113 346
0113 347 ;*****
0113 348
0010 0113 349 MAKE_P1_IMAGE_PERMANENT: .WORD ^M<R4>
0115 350
0115 351 ;In order to prevent all this from being undone on the first
0115 352 ; image rundown, we move the definition of the end of base
0115 353 ; P1 space to point to the end of our work.
0115 354
0115 355 ;This code must be executed in kernel mode to write this
0115 356 ; location.
0115 357
0000000*GF 000000C*EF DD 0115 358 MOVL P1_END_ADDR_ACTUALLY_ADDED, G^CTL$GL_CTLBASVA
0120 359
50 0000000*SF DD 0120 360 MOVL #SS$_NORMAL, R0
0127 361
04 0127 362 RET

```

```

0000 1 .TITLE ADD_PERM_PER_PROCESS_DISPVEC
0000 2 .SUBTITLE Introduction
0000 3
0000 4 ;++
0000 5 ;
0000 6 ; Module:      ADD_PERM_PER_PROCESS_DISPVEC
0000 7 ;
0000 8 ; Facility:    EGP1
0000 9 ;
0000 10 ; Function:   Make a permanent entry in the P1 dispatch vector
0000 11 ;           for:
0000 12 ;           * Change mode to kernel dispatcher
0000 13 ;           * Change mode to executive dispatcher
0000 14 ;           * Rundown interceptor
0000 15 ;
0000 16 ; Invocation Protocol: VAX/VMS Procedure Calling Standard.
0000 17 ;
0000 18 ;           CALL ADD_PERM_PER_PROCESS_DISPVEC
0000 19 ;           (DISPATCH_ROUTINE_TYPE, DISPATCH_ROUTINE_ADDRESS, STATUS)
0000 20 ;
0000 21 ;           DISPATCH_ROUTINE_TYPE --->
0000 22 ;           Longword integer.
0000 23 ;           Type of dispatch vector routine.
0000 24 ;           EGP1VEC_C_USRCHKM = Kernel Dispatcher
0000 25 ;           EGP1VEC_C_USRCHME = Executive Dispatcher
0000 26 ;           EGP1VEC_C_USRUNDWN = Rundown Routine
0000 27 ;
0000 28 ;           DISPATCH_ROUTINE_ADDRESS ---> Longword.
0000 29 ;           Address of routine.
0000 30 ;
0000 31 ;           STATUS <--- Longword.
0000 32 ;           Status of Operation.
0000 33 ;
0000 34 ; Status Returns:
0000 35 ;
0000 36 ;           $$$_NORMAL Normal Successful Operation.
0000 37 ;           $$$_INSFARG Bad Argument Count.
0000 38 ;           EGP1VEC__BADVECCOD Invalid dispatch routine type.
0000 39 ;           EGP1VEC__SADADDR Routine is not above base of P1
0000 40 ;           space and will not survive
0000 41 ;           rundown.
0000 42 ;           EGP1VEC__PRVVEC There is already a per-image vector
0000 43 ;           in place. To proceed would cause
0000 44 ;           temporary vectors to become permanent
0000 45 ;           pointing to non-existent code.
0000 46 ;           (Can happen if the image using this
0000 47 ;           routine is linked with the debugger)
0000 48 ;
0000 49 ; Notes:
0000 50 ;
0000 51 ;           1. These entries are usually made as per-image entries
0000 52 ;           by linking to a privileged shareable image (see
0000 53 ;           example in SYS$EXAMPLES:USSDISP.MAR). The entries
0000 54 ;           are removed by image rundown. This routine will
0000 55 ;           cause a permanent entry to the dispatch table which
0000 56 ;           is unaffected by rundown.
0000 57 ;           2. Routine added by this procedure is activated by JSB style

```

Introduction

```

0000 53 ; call, and should therefore have no entry mask.
0000 59 ;
0000 60 ; Revision: 1.0 19-Sep-1985 Larry L. Johnson
0000 61 ;
0000 62 ; Language: VAX/VMS Macro-32 Assembler.
0000 63 ;
0000 64 ; Required Macro Libraries:
0000 65 ; STARLET The default VAX System Macro Library.
0000 66 ; (Automatically scanned.)
0000 67 ;
0000 68 ; External Status References:
0000 69 ;
0000 70 ; Facility Defined Status Values:
0000 71 ; EGP1VEC__BADVECCOD
0000 72 ; EGP1VEC__BADADDR
0000 73 ; EGP1VEC__PRVVEC
0000 74 ;
0000 75 ; VAX/VMS Common Run Time Library Status Values: None
0000 76 ;
0000 77 ; VAX/VMS RMS Status Values: None
0000 78 ;
0000 79 ; VAX/VMS System Service Status Values: None
0000 80 ;
0000 81 ; SSS_NORMAL Normal Successful Operation.
0000 82 ; SSS_INSFARG Bad Argument Count.
0000 83 ;
0000 84 ; External Data References:
0000 85 ;
0000 86 ; From SYS$SYSTEM:SYS.STB
0000 87 ;
0000 88 ; IAC$AW_VECSET Address of the vector size
0000 89 ; array in P1 space.
0000 90 ; IAC$AW_VECRESET Address of the original vector
0000 91 ; size array in P1 space.
0000 92 ; CTL$GL_CTLBASVA Pointer to address of end of
0000 93 ; base of P1 space.
0000 94 ; CTL$A_DISPVEC Pointer to address of first
0000 95 ; half-page vector in P1.
0000 96 ;
0000 97 ; External Routine References:
0000 98 ;
0000 99 ; Facility Defined Routines: None
0000 100 ;
0000 101 ; VAX/VMS Common Run Time Library Routines: None
0000 102 ;
0000 103 ; VAX/VMS RMS Routines: None
0000 104 ;
0000 105 ; VAX/VMS System Services:
0000 106 ;
0000 107 ; SYS$CMKRNL
0000 108 ;
0000 109 ;--
0000 110

```

Procedure Data

```

0000 112 .SUBTITLE Procedure Data
0000 113
0000 114 ;.....
0000 115 ;... Argument List Offsets .....
0000 116 ;.....
0000 117
0000 118 ;Argument List Offset Definitions:
0000 119
00000004 0000 120 DISPATCH_ROUTINE_TYPE = 4
00000002 0000 121 DISPATCH_ROUTINE_ADDRESS = 8
0000000C 0000 122 STATUS = 12
0000 123
0000 124
0000 125 ;.....
0000 126 ;... Local Read-Only Data Allocations .....
0000 127 ;.....
0000 128
0000 129
00000000 130 .PSECT _EGP1_LOCAL_RO_DATA, -
0000 131 QUAD, CON, NOEXE, LCL, NOPIC, NOSHR, REL, NOWRT, NOVEC
0000 132
0000 133 ;External Parameters:
0000 134
0000 135 $OPDEF ;Define symbolic opcodes.
0000 136 $$SDEF ;Define system service status codes.
0000 137
0000 138 ;Module Parameters:
0000 139
0000 140 ;The code indicating absolute (PC relative deferred)
0000 141 ; in an instruction argument.
0000 142
0000007F 0000 143 ABSOLUTE_ADDRESSING_MODE = ^X9F
0000 144
0000 145 ;The size of each vector in the dispatch area.
0000 146 ; The dispatch area is currently four vectors
0000 147 ; of one-half page each for Kernel Dispatch,
0000 148 ; Executive Dispatch, Rundown Dispatch, and
0000 149 ; per-image message vectors respectively.
0000 150
00000100 0000 151 MAX_DISPATCH_VECTOR_SIZE = ^X100
0000 152
0000 153
0000 154 ;.....
0000 155 ;... Local Writeable Data Allocations .....
0000 156 ;.....
0000 157
0000 158 .PSECT _EGP1_LOCAL_RW_DATA, -
00000000 159 QUAD, CON, NOEXE, LCL, NOPIC, NOSHR, REL, WRT, NOVEC
0000 160
0000 161 ; Local Status Variable Allocations: None
0000 162
0000 163 ; Local Variable Allocations: None
0000 164

```

Main Procedure

```

0000 166 .SUBTITLE      Main Procedure
0000 167 ;.....
0000 168 ;... Executable .....
0000 169 ;.....
0000 170
0000 171 .PSECT  _EGP1_CODE, -
00000000 172     QUAD, CON, EXE, LCL, NOPIC, NOSHR, REL, NOWRT, NOVEC
0000 173
0FE6 0000 174 .ENTRY ADD_PERM_PER_PROCESS_DISPVEC, ^M<R3,R5,R6,R7,R8,R9,R10,R11>
0002 175
0002 176 ;Register Usage:
0002 177 ;
0002 178 ; R3 = Dispatch routine address.
0002 179 ; R4 - Not used.
0002 180 ;     Cannot be used to pass info to kernel routine.
0002 181 ; R5 = Dispatch vector index for rundown vector.
0002 182 ; R6 = Address of VECSET array.
0002 183 ; R7 = Address of VECRESET array.
0002 184 ; R8 = Instruction load pointer.
0002 185 ; R9 = Address of original terminating RSB instruction.
0002 186 ; R10 = Address of size field of rundown vector.
0002 187 ; R11 = Dispatch vector offset to base of dispatch area.
0002 188 ;
0002 189
0002 190 ;Initialize registers with primary data.
0002 191
0002 192     ;Move arguments to registers
0002 193
    53 08 BC 00 0002 194     MOVL  @DISPATCH_ROUTINE_ADDRESS(AP), R3
    55 04 BC 00 0006 195     MOVL  @DISPATCH_ROUTINE_TYPE(AP), R5
000A 196
000A 197     ;Get the addresses of the VECSET and VECRESET arrays
000A 198
    56 00000000*BF 00 000A 199     MOVL  #IAC$AW_VECSET, R6
    57 00000000*BF 00 0011 200     MOVL  #IAC$AW_VECRESET, R7
0018 201
0018 202 ;Validate arguments.
0018 203
0018 204
0018 205     ;Validate argument count.
    6C 03 01 0018 206     CMPL  #3, (AP)
0018 207     BEQL  10$,
    50 00000114 3F 00 0010 208     MOVL  #$$$_INSFARG, R0
    67 11 0024 209     BRB   COMMON_EXIT
0026 210
0026 211
0026 212     ;Validate dispatch selection code.
0026 213
    55 05 0026 214     TSTL  R5
    09 13 0028 215     BGEQ  20$,
    50 00000000*BF 00 002A 216     MOVL  #EGP1VEC__BADVECCOD, R0
    5A 11 0031 217     BRB   COMMON_EXIT
0033 218
0033 219
0033 220     CMPL  R5, #EGP1VEC_C_USRUNDWN
    09 15 003A 221     BLEQ  30$,
    50 00000000*BF 00 003C 222     MOVL  #EGP1VEC__BADVECCOD, R0
    
```

483

```

Main Procedure
      46 11 0043 223          BRB      COMMON_EXIT
          0045 224          30$:
          0045 225
          0045 226          ;Validate that routine address is at least
          0045 227          ; above the base of P1 space. Anything
          0045 228          ; lower is not going to exist, or will
          0045 229          ; disappear during rundown.
          0045 230
00000000*9F 53 01 0045 231          CMPL   R3, @#CTL$GL_CTLBASVA
          09 1E 004C 232          BGEQU  40$.
50 00000000*8F 00 004E 233          MOVL  #EGP1VEC__BADADDR, R0
          30 11 0055 234          BRB      COMMON_EXIT
          0057 235          40$:
          0057 236
          0057 237
          0057 238 ;Validate environment
          0057 239
          0057 240          ;Validate that no privileged shareable image vectors
          0057 241          ; exist for this image. Return error if there are
          0057 242          ; since this routine would make them permanent rundown
          0057 243          ; vectors to code that would no longer exist after the
          0057 244          ; first rundown.
          0057 245
          0745 6645 01 0057 246          CMPW   (R6)[R5], (R7)[R5]
          09 13 005C 247          BEQL  50$
50 00000000*8F 00 005E 248          MOVL  #EGP1VEC__PRVVEC, R0
          26 11 0063 249          BRB      COMMON_EXIT
          0067 250          50$:
          0067 251
          0067 252 ;Calculate addresses of data of interest in target dispatch vector.
          0067 253
          0067 254          ;Get address of size byte (first longword of the half-page
          0067 255          ; vector) -- the base of the vector.
          0067 256
          0067 257          ;Calculate offset to vector of interest off
          0067 258          ; base of dispatch vector area.
          0067 259
          55 05 0067 260          MULL3  R5,-
          0069 261          #MAX_DISPATCH_VECTOR_SIZE,-
5A 00000100 8F 0069 262          R10
          006F 263
          006F 264          ;Add to base of dispatch vector area to get address
          006F 265          ; of desired vector.
          006F 266
5A 00000000*8F 00 006F 267          ADDL2  #CTLSA_DISPVEC, R10
          0076 268
          0076 269          ;Calculate address of the original terminating RSB instruction
          0076 270          ; within vector (the first longword of the vector contains
          0076 271          ; the offset to the terminating RSB off the base of the
          0076 272          ; particular vector).
          0076 273
          59 5A 6A C1 0076 274          ADDL3  (R10), R10, R9
          007A 275
          007A 276          ;Calculate the address of first byte to be loaded.
          007A 277          ; (the RSB is going to be left in place until everything
          007A 278          ; else is finished).
          007A 279
          58 59 01 C1 007A 280          ADDL3  #1, R9, R8
          007E 281
          007E 282          ;Write the vector instruction entry in kernel mode.
          007E 283
          007E 284          $CMKRNLS =
          007E 285          ROUTIN = WRITE_VECTOR_INSTRUCTION
          0080 286
          0080 287          COMMON_EXIT:
          0080 288
          0C 8C 5D 00 0080 289          MOVL  R0, @STATUS(AP)
          0091 290
          04 0091 291          RET
          0092 292

```

Write Vector Instruction

```

0072 294 .SUBTITLE Write Vector Instruction
0072 295
0072 296 ;*****
0072 297 ; Write vector instruction in kernel mode.
0072 298
0072 299
0010 0092 299 WRITE_VECTOR_INSTRUCTION:      .WORD  ^M<R4>
0094 300
0094 301 ;Register Usage:
0074 302 ;
0074 303 ; The following registers are assumed initialized with the following
0074 304 ; data, of which only R8, the instruction load pointer, is changed.
0074 305 ;
0074 306 ; R3 = Dispatch routine address.
0074 307 ; R4 = The usual PCB address provided by kernel dispatcher.
0074 308 ;      (Restored by mask to avoid being lost by kernel dispatcher.)
0074 309 ; R5 = Dispatch vector index for rundown vector.
0074 310 ; R6 = Address of VECSET array.
0074 311 ; R7 = Address of VECRESET array.
0074 312 ; R8 = Instruction load pointer.
0074 313 ; R9 = Address of original terminating RSB instruction.
0074 314 ; R10 = Address of size field of rundown vector.
0074 315 ;
0074 316
0074 317 ;Write the address mode (absolute). (Later the RSB will
0074 318 ; be converted to a JSB when everything is complete. This way
0074 319 ; if exit is forced, the vector will functionally appear
0074 320 ; to be as it was... no undefined states.)
0074 321
38 9F 8F 90 0094 322          MOVB  #ABSOLUTE_ADDRESSING_MODE, (R8)+
0098 323
0098 324          ;Write the address of the routine.
0098 325
38 53 00 0098 326          MOVL  R3, (R8)+
0098 327
0098 328          ;Write the new terminal RSB instruction. (Note that
0098 329          ; we don't auto-increment here. This will facilitate
0098 330          ; calculation of the offset to the terminal RSB to
0098 331          ; be placed in the vector and the vector offset arrays.)
0098 332
63 05 90 0098 333          MOVB  #OP$_RSB, (R8)
009E 334
009E 335          ;Update the size field of the vector, the vector array,
009E 336          ; and the vector reset array.
009E 337
50 58 5A 03 009E 338          SUBL3  R10, R8, R0      ;Calculate the new size
00A2 339          MOVL  R0, (R10)      ;Store in vector.
6645 50 6C 00A5 340          MOVW  R0, (R6)[R5]      ; and the vector array
6745 50 80 00A9 341          MOVW  R0, (R7)[R5]      ; and the reset array
00AD 342          ; (setting the reset
00AD 343          ; array is what makes it
00AD 344          ; permanent)
00AD 345
00AD 346          ;Change the old terminal RSB to a JSB to activate the
00AD 347          ; rundown vector.
00AD 348
67 16 90 00AD 349          MOVB  #OP$_JSB, (R9)
00B0 350
00B0 351 ;Return success to the caller.
00B0 352
50 01 00 00B0 353          MOVL  #SS$_NORMAL, R0
00B3 354          RET
00B4 355
00B4 356 ;*****
00B4 357
00B4 358 .END

```

485

APPENDIX D
LISTINGS OF RUNDOWN INTERCEPTOR LOADER

```
0001 PROGRAM PROTOTYPE_INTERCEPTOR_LOADER
0002
0003 c++
0004 c
0005 c Module: PROTOTYPE_INTERCEPTOR_LOADER
0006 c
0007 c Facility: EGRNDN
0008 c
0009 c Function: Load prototype rundown interceptor into P1 permanently,
0010 c adding permanent rundown vector to image activator's
0011 c rundown dispatch vector.
0012 c
0013 c Invocation Protocol: VMS Image Activation.
0014 c
0015 c Notes:
0016 c
0017 c Revision: 1.0 21-Sep-1985 Larry L. Johnson
0018 c
0019 c Language: VAX/VMS Fortran, Enhanced 1977 ANSI Standard.
0020 c
0021 c External Status References:
0022 c
0023 c IMPLICIT NONE
0024 c
0025 c Facility Defined Status Values: None
0026 c VAX/VMS Common Run Time Library Status Values: None
0027 c VAX/VMS RMS Status Values: None
0028 c VAX/VMS System Service Status Values: None
0029 c
0030 c External Data References:
0031 c
0032 c EXTERNAL EGP1VEC_C_USRUNDWN
0033 c INTEGER*4 EGP1VEC_C_USRUNDWN
0034 c
0035 c
0036 c External Routine References:
0037 c (Routines which include type specification are
0038 c function type routines which typically return
0039 c status in RD, the Fortran function value.
0040 c
0041 c Facility Defined Routines:
0042 c
0043 c MERGE_SHAREABLE_IMAGE_INT0_P1
0044 c ADD_PERM_PER_PROCESS_DISPVEC
0045 c
0046 c VAX/VMS Common Run Time Library Routines: None
0047 c VAX/VMS RMS Routines: None
0048 c VAX/VMS System Services: None
0049 c
0050 c--
0051
0052 C.....
0053 c... Local Specifications .....
0054 C.....
0055
0056 c... Local Status Variable Specifications:
0057
```

```

0060      INTEGER*4      STATUS      ! General module status.
0061      INTEGER*4      P1_MERGE_STATUS ! Load of image.
0062      INTEGER*4      VECTOR_STATUS ! Addition of per-process vector.
0063
0064      c... Local Variable Specifications:
0065
0066      ! Logical name for rundown interceptor to be loaded.
0067
0068      CHARACTER*11    IMAGE_FILE_NAME /*'INTERCEPTOR'*/
0069
0070      ! Address range into which the interceptor was loaded.
0071
0072      INTEGER*4      HIGHEST_ADDRESS_LOADED
0073      INTEGER*4      LOWEST_ADDRESS_LOADED
0074
0075      ! Image header of the interceptor.
0076
0077      CHARACTER*512    IMAGE_HEADER_BUFFER
0078
0079      ! Transfer array, extracted from the interceptor image header.
0080
0081      INTEGER*4      IMAGE_TRANSFER_ARRAY(3)
0082
0083      c.....
0084      c... Executable .....
0085      c.....
0086
0087      ! Merge the rundown interceptor into P1 space permanently.
0088
0089      CALL MERGE_SHAREABLE_IMAGE_INTO_P1
0090      &      (IMAGE_FILE_NAME
0091      &      >IMAGE_TRANSFER_ARRAY
0092      &      >LOWEST_ADDRESS_LOADED
0093      &      >HIGHEST_ADDRESS_LOADED
0094      &      >IMAGE_HEADER_BUFFER
0095      &      >P1_MERGE_STATUS
0096      &      )
0097
0098      STATUS = P1_MERGE_STATUS
0099
0100      ! Summarize the operation for demonstration purposes.
0101
0102      WRITE (6,1)
0103      &      P1_MERGE_STATUS
0104      &      >HIGHEST_ADDRESS_LOADED
0105      &      >LOWEST_ADDRESS_LOADED
0106      &      >IMAGE_TRANSFER_ARRAY
0107
0108      1  FORMAT (
0109      &      X,'Interceptor Load Detail:/'
0110      &      X,' Merge Status: ',Z3/
0111      &      X,' Highest Address: ',Z8/
0112      &      X,' Lowest Address: ',Z8/
0113      &      X,' Transfer Array: ',Z8,', ',Z8,', ',Z8/
0114      &      )

```

488

```

0115 ! If the interceptor made it to its destination then
0116 !     Add a permanent rundown vector to the image activator's
0117 !     dispatch vector array, causing the interceptor to be
0118 !     invoked on each image rundown, including process deletion.
0119 ! Endif
0120
0121     IF (P1_MERGE_STATUS) THEN
0122         CALL ADD_PERH_PER_PROCESS_DISPVEC
0123             (%LOGC(EGP1VEC_C_USRUNDWN)
0124             ,IMAGE_TRANSFER_ARRAY(1)
0125             ,VECTOR_STATUS
0126             )
0127         STATUS = VECTOR_STATUS
0128     ENDIF
0129
0130 ! Exit with general status.
0131
0132     CALL EXIT (STATUS)
0133
0134     END
    
```

PROGRAM SECTIONS

Name	Bytes	Attributes
0 500B0	120	PIC CON REL LCL SHR EXE RD NOWRT LONG
1 500B4	142	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2 500B8	640	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
Total Space Allocated	902	

ENTRY POINTS

Address	Type	Name
0-00000000		PROTOTYPE_INTERCEPTOR_LOADER

VARIABLES

Address	Type	Name	Address	Type	Name
2-00000204	I*4	HIGHEST_ADDRESS_LOADED	2-0000000C	CHAR	IMAGE_FILE_NAME
2-00000017	CHAR	IMAGE_HEADER_BUFFER	2-00000228	I*4	LOWEST_ADDRESS_LOADED
2-00000210	I*4	P1_MERGE_STATUS	2-00000218	I*4	STATUS
2-00000220	I*4	VECTOR_STATUS			

489

PROTOTYPE_INTERCEPTOR_LOADER

30-May-1986 13:22:26
27-Apr-1986 10:48:55

VAX FORTRAN V4.3-145 Page 4
PROTOTYPE_RUNDOWN_INTERCEPTOR_LOADER.FOR:3

ARRAYS

Address	Type	Name	Bytes	Dimensions
2-00000000	I*4	IMAGE_TRANSFER_ARRAY	12	(3)

LABELS

Address	Label
1-00000000	1

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name
	ALD_PERM_PER_PROCESS_DISPVEC		EGP1VEC_C_USRUNDWN		FOR\$EXIT
	MERGE_SHAREABLE_IMAGE_INTC_P1				

COMMAND QUALIFIERS

```

FORTRAN/LIST PROTOTYPE_RUNDOWN_INTERCEPTOR_LOADER.FOR
/CHKCHK=(NOBOUNDS,OVERFLOW,NOUNDERFLOW)
/DEBUG=(NOSYMBOLS,TRACEBACK)
/STANDARD=(NOSYNTAX,NO$OURCE_FORM)
/SHOW=(NOPREPROCESSOR,NOINCLUDE,MAP,NODICTIONARY,SINGLE)
/WARNINGS=(GENERAL,NODECLARATIONS)
/CONTINUATIONS=19 /NOCROSS_REFERENCE /NOD_LINES /NOEXTEND_SOURCE /F77
/NO$_FLOATING /I4 /NONACHINE_CODE /OPTIMIZE

```

COMPILATION STATISTICS

```

Run Time:          1.20 seconds
Elapsed Time:     3.62 seconds
Page Faults:      226
Dynamic Memory:   335 pages

```

APPENDIX E
LISTINGS OF MESSAGE FILES

```
1 .TITLE P1_DISPVEC_MSGS_AND_CONSTANTS
2
00000301 3 .FACILITY      EGP1VEC,1 /PREFIX=EGP1VEC__
4
5 .SEVERITY SUCCESS
6
04010009 7 NORMAL        <Normal successful completion>
8
9 .SEVERITY INFORMATIONAL
10
11 .SEVERITY WARNING
12
13 .SEVERITY ERROR
14
03010012 15 PRVVEC       <Privileged sharable image vector is in the way.>
0801201A 16 BADVECCOD    <Invalid vector selection code.>
0E018022 17 BADADDR      <Dispatch routine is in volatile address space.>
18
19 .SEVERITY SEVERE
20
21 ! Define facility constants.
22
23 .LITERAL      EGP1VEC_C_USRCHKM = 0,-
24               EGP1VEC_C_USRCHME,-
25               EGP1VEC_C_USRUNDWN
26
27 .END
```

There were 0 errors, 0 warnings, and 0 informational messages issued.
MESSAGE/LIST EGP1VEC

APPENDIX F

PREPARATION PROCEDURES

```
$ MACRO = "MACRO/LIST/CROSS=(REGISTERS,SYMBOLS,MACROS)
$ !*****
$ !
$ !-----
$ !
$ !             !!!  NOTE  !!!
$ !
$ ! If any of the images are linked with the debugger,
$ ! rundown vectors belonging to the debugger will get in the way of
$ ! adding the permanent rundown vector, causing the load to fail
$ ! with an appropriate error message.
$ !-----
$ !
$ FORTRAN/LIST PROTOTYPE_RUNDOWN_INTERCEPTOR_LOADER.FOR
$ MACRO MERGE_SHAREABLE_IMAGE_INTO_P1+SYS$LIBRARY:LIB/LIB
$ MACRO ADD_PERM_PER_PROCESS_DISPVEC+SYS$LIBRARY:LIB/LIB
$ MESSAGE/LIST EGPIVEC
$ !
$ LINK/POIMAGE/NODEBUG
PROTOTYPE_RUNDOWN_INTERCEPTOR_LOADER, SYS$INPUT/OPTION
MERGE_SHAREABLE_IMAGE_INTO_P1
ADD_PERM_PER_PROCESS_DISPVEC
EGPIVEC
SYS$SYSTEM:SYS.STB/SEL
$ !
$ MACRO PROTOTYPE_RUNDOWN_INTERCEPTOR+SYS$LIBRARY:LIB/LIB
$ LINK/NODEBUG/SHARE/MAP/FULL
PROTOTYPE_RUNDOWN_INTERCEPTOR, SYS$INPUT/OPTION
SYS$SYSTEM:DCLDEF.STB/SEL
SYS$SYSTEM:SYS.STB/SEL
$ !
$ EXIT
```


The Time Warp Simulator

or

An Object Oriented Simulated Concurrent Processing Shell for the VAX

J. Steven Hughes
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA

ABSTRACT

This paper describes the Time Warp Simulator, an application which runs on a sequential processor and simulates the Time Warp distributed simulation mechanism. Implemented in parallel with the Time Warp/Hypercube project, this tool allowed the early development and testing of application code and the testing of the Time Warp interface. In addition the simulator has been found to promote an object oriented application development methodology while also allowing simulated concurrent processing.

The Time Warp Simulator is a tool that was developed to support the implementation of an application to run under the Time Warp distributed simulation mechanism on a 32 node version of the CalTech Hypercube multiprocessor. The primary purpose of the overall project was to demonstrate the speedup that could be realized through concurrent execution of a discrete event simulation under the control of Time Warp. This project brought together leading edge work in hardware architecture, operating systems, and application development methodology.

1.1 Hardware Architecture

The CalTech Hypercube was chosen as a suitable hardware architecture for the implementation of the Time Warp mechanism

because of its symmetry[3]. The hypercube architecture is a multiprocessor architecture with 2^d identical nodes. Each of these nodes is connected individually to d other nodes over a bidirectional link. Each node has d nearest neighbors with a maximum distance d to any other node. The symmetry of the architecture allows topologically identical views of the system from any single node.

The implementation of the hypercube architecture[1] is a result of a CalTech collaboration between Charles Seitz and Geoffrey Fox, to build a 64 node hypercube and an initial operating system and to run physics problems. A subsequent combined CalTech/JPL project[4] was started in October 1983, with Geoffrey Fox as principal investigator, to continue research and development of the hardware, system software, and applications.

The Mark II machine currently being used for Time Warp implementation, is a 32 processor configuration. Each node consists of a 8086/8087 processor pair with 256k RAM. An Intel 310 functions as the intermediate host handling all I/O to the cube and is connected to a VAX by ethernet.

1.2 Operating System

The Time Warp mechanism[2] is essentially a distributed operating system specialized for simulation. It was invented at RAND Corporation by David Jefferson and Henry Sowizral and first implemented in a version of Lisp on a network of microprocessor workstations. Time Warp is based on a discrete event simulation paradigm and allows the dynamic creation and destruction of object instances and subsequent object communication using time stamped messages. The concept of simulation time plays a key role and when associated with messages, signals when an event should be processed. The Time Warp mechanism speeds up simulations in real time by executing objects concurrently in a multiprocessor environment and by allowing some objects to proceed ahead in simulation time while others lag behind. The situation that can arise where an object receives a message with a time stamp in its past, is handled using a rollback mechanism that returns the object to a prior state. This synchronization is invisible to the application programmer except that some concept of the mechanism should be understood in order to produce efficient code.

1.3 Application Development Methodology

The application development methodology which results from programming an application to run under the Time Warp mechanism is based on an object oriented methodology. By definition, all code to be run under Time Warp must be partitioned into objects which have no shared memory. An object may communicate with any other object but may do so only through the use of time stamped messages sent to be received at the present or some future time.

2 TIME WARP MECHANISM

Intuitively the Time Warp mechanism uses the virtual time stamped event messages sent between objects to schedule the execution of the objects. In addition the object is executed with its local virtual time set to the virtual time of the event message. Virtual time is similar to the concept of simulation time in the discrete event driven simulation paradigm in that it specifies the time for processing an event.

As mentioned before, on a multiprocessor hardware architecture, Time Warp achieves speed up by allowing some parts of the simulation to proceed ahead in virtual time while others lag behind. In process synchronization, two situations arise. If object A schedules an event in object B's future, the message from A is inserted in B's input queue in time stamp order, for subsequent processing. However if the event is scheduled for B's present or past, the processing could proceed incorrectly depending on its affect on B's state. Time Warp handles this latter situation by being prepared to roll object B back to an earlier virtual time and to start it processing again with the new message in its input queue. Rollback is accomplished using state snapshots, antimessages, and global virtual time. Global virtual time is a computed time before which there exists no unprocessed messages in the system. It is used for management of processed messages and prior states and for committing output messages outside the system. An antimessage is a copy of a message which causes annihilation of both messages when they encounter one another. When object B is rolled back, all messages that had been sent in what is now the object's future are subject to having an antimessage created and sent. If an antimessage and the original unprocessed message meet in the input queue of an object, the annihilation is simple, and does not effect the object. However, if the original message has been processed, in which case the antimessage has been received in the objects past, the antimessage itself will cause a rollback.

2.1 Applications Development Under Time Warp

An application developed to be run under Time Warp consists of a set of dynamically created objects which communicate with each other using event and query messages. An object performs event message queuing, querying, object creation and destruction, and other utility functions through the use of Time Warp system calls or entry points. Under the current implementation each logical object type is implemented using three code segments and one data segment. The code segments are the Init section, Event section, and Query section.

2.1.1 Init Code Section -

The Init section is that section of code that is executed once whenever the object instance is created. It allows for the creation of the object's state but does not allow the processing of any messages. Assignment of state variables in this section are one time assignments.

2.1.2 Event Code Section -

The event section handles event messages. When two objects communicate using event messages, the message sent is inserted into the receiving object's input queue and execution of the sending object continues. An object is scheduled for execution when it has at least one event message in its input queue. The set of event messages in the queue with the lowest receive virtual time are collected and at object execution the object's local virtual time is set to this virtual time. The object may then use time warp entry points to access the text, sender object's name, and send receive time of any event message in this set. The code in the event section may also perform other time warp calls including those for the creation and destruction of other objects.

2.1.3 Query Code Section -

The query section handles query messages. The query is a special type of object communication that allows one object to query the state of other with the restriction that the query occurs with the local virtual times of the objects being equal. Object A may query object B in either its event section or its query section. Object B handles the query in its query section and returns the reply using the query reply entry point. Object B has the restriction that it may not modify its own state.

2.1.4 State -

Each object must have a data structure called the state. This state must include all global and static variables. The state is created dynamically at object create time when the init section of the project is executed.

2.2 Time Warp Entry Points[5]

Following are brief functional descriptions of the Time Warp entry points.

Obcreate - Obcreate is the object create time warp entry point. It allows the creation of a uniquely named object instance at a specified virtual time. Arguments also exist for specifying the logical object type of the object and the processor node number that the object will initially reside on.

Obdestroy - Obdestroy is the object destroy Time Warp entry point. It allows the destruction of a named object instance at a specified virtual time.

Evtmsg - Evtmsg queues an event message in the receiving object's input queue. The receiving time, length of the message, and the message text are supplied as arguments. The message is guaranteed to be queued.

Qrymsg - Qrymsg queues a query event message in the receiving object's input queue and causes the sending object to wait for the reply. The query text and its length, and the reply buffer and its length are supplied as arguments. A return status is also available specifying whether or not the reply message had to be truncated. The query of one object by another is considered to occur with the local virtual time of each object being equal.

Qreply - Qreply is the Time Warp entry point used in the query section of an object being queried to return a query response. The receiving object's name, the reply and its length are supplied as arguments.

Getmsg - Getmsg may be used in either an object's event or query section to get the text of all messages available at the current local virtual time. The message number, a buffer and the buffer size are supplied as arguments. The actual length of the data copied into the buffer and a status are returned. The status may be success, message truncation, or message does not exist.

Getmbytes - Getmbytes is functionally similar to Getmsg except that a byte pointer and length are specified for returning a substring of the message text.

Me - Me returns the name of the calling object.

Sendtime - Sendtime returns the virtual time that a message was sent. A message number is supplied as an argument and the virtual time and a status is returned. The status may be set to success or message does not exist.

Sender - Sender returns the sending object's name. A message number is supplied as an argument and the name of the sending object and a status is returned. The status may be success or message does not exist.

Simtime - Simtime returns the current local virtual time of the calling object.

Mcount - Mcount returns the number of messages in an object's input queue that have a receive virtual time equal to the calling object's current local virtual time.

3 THE TIME WARP SIMULATOR[6] - HISTORY

The need for the simulator is evident when one considers the complexity of the Time Warp implementation project. An operating system was to be developed for a new hardware architecture and a methodology and skills for the design and implementation of a concurrent event application had to be developed. In addition, the Operating System based on Time Warp was to be machine independent. Therefore, a machine interface level had to be implemented to handle message communication, routing, memory management, error handling and I/O. The Time Warp layer would then handle typical operating system functions such as object creation and destruction, scheduling, and synchronization.

In considering the need for the application programmers to be able to start early development it seemed reasonable to develop a simulator of the Time Warp mechanism on a sequential processor. Given that Time Warp mechanism allow messages to be sent only in the present or at a future time, a single system wide queue of event messages,

ordered by receive virtual time, would give the simulator the capability to determine what object is executing or should be executed next in any instance of real time. This precludes the need for antimessages since objects would not be executed out of order and only pseudo rollback is required in a few specific cases. Global virtual time, difficult to calculate in a multiprocessor environment, is immediately available and the number of snapshots of the state of an object is limited to two.

3.1 Time Warp Simulator - Justification

Due to the reduction in complexity of the system, the simulator was available in a relatively short amount of time and was not dependent on other work units of the project. In comparison, the layered aspect of the system implementation on the hypercube caused many critical dependencies between work units. The availability of the simulator would therefore allow the early development of the application software and the early testing of the proposed Time Warp interfaces.

3.2 Time Warp Simulator System Structure

An application to be run under the Time Warp Simulator is linked to run as a single job on a sequential processor. This single job consists of a number of logical modules that handle the required functional areas, where each module may be implemented using several procedures or functions. The three main functional groupings of modules are the Time Warp entry points, the Time Warp simulator driver and utility modules, and the application modules. All coding is currently done in the C programming language and the system was originally developed for the VAX family of processors. Since its implementation, the simulator has been ported to the IBM/PC and the Macintosh.

An application developer simply codes and compiles his application and links it with the precompiled Time Warp entry point, driver and utility modules and executes the resulting image as a single job under the host's operating system. This allows the

developer to use the host's interactive debugging system or to do application debugging using standard output.

3.3 Adherence To Time Warp

In implementing a simulator of the Time Warp mechanism, the intent was to implement a mirror image of the application interface in form and function. Simulators by definition do not perfectly imitate the original system and three exceptions are known to exist. The first exception is based on a Time Warp characteristic that in a multiprocessor environment there would be no guarantee of the order of arrival of messages. This is evident in the situation where several messages have been sent to be received by the same object at the same virtual time. The simulator however, in the above case, queues messages in the order sent and an application programmer could make use of this information. The resulting application could then possibly execute differently in a multiprocessor environment.

The second exception is a hidden message situation involving 3 objects sending messages to be received at the same virtual time. If object A sends a message to both objects B and C, and object B sends a message to object C, then the order in which object A sends its messages is critical. If the message from A to C is sent first, then object C is scheduled for execution before object B and therefore will not have the message from B available. The solution to this problem is the proper ordering of the message sending. In addition, even though actual Time Warp could handle this situation, it would be advantageous to code in a similar manner.

The third exception is that the simulator requires the use of an application object entry point array. This array simply allows the simulator to link the application object's init, event and query code sections. The array is coded as a separately compiled module and need not be recompiled as long as no new object type names are added to the simulation.

3.4 Development Aids

Three application development aids are available to the application programmer who is developing code using the simulator. Breakpoints may be set at either object execution (when the object receives a message) or when a Time Warp entry point is called. At a breakpoint the capabilities then exist to either display the event message queue or the object's state.

4 A TIME WARP APPLICATION

An application that has been used as a test case for the Time Warp system is the Game of Life. This game is played on an N by N board. The N**2 cells have two states, either live or dead, and a first generation or initial configuration of live and dead cells is given by the player. Succeeding generations have their configurations determined by a set of rules. The set chosen for this implementation are 1) a live cell remains alive in the next generation if it has two or three live neighbors in the current generation, 2) a dead cell becomes alive in the next generation if it has exactly three live neighbors in the current generation, and 3) otherwise a cell is dead in the next generation. For this implementation the board is wrapped around so that all cells have exactly 8 neighbors.

The design of this game for Time Warp includes the implementation of four logical object types. These are the board initializer object, cell object, board display object, and a standard output object.

4.1 Board Initializer Object

The board initializer object executes once and creates all instances of the cell objects, the board display object and the standard output object. It also sends an event message to each of the created objects containing initialization information such as the board size.

4.2 Board Cell Objects

Given an N by N board, N**2 board cell objects are created by the initializer object. The one initialization event message received by the cell object after creation is used to determine its position on the board, its neighbors, and its initial state. Subsequent execution of a cell object consists of 1) sending its current state via an event message to the board display object, 2) querying its eight neighbors for their states and setting its own state accordingly, and 3) sending an event message to itself with a future time stamp signifying the next generation.

4.3 Board Display Object

The Life board display object receives N**2 event messages from the cell objects at each generation. It then formats the game board and sends one row of the board at a time to the standard output object for display on the terminal.

4.4 Life Game Results

A typical implementation of the life game used as a test case consists of an 8 by 8 board with some initial configuration such as the blinker. A blinker is a straight line configuration of three live cell which returns to its initial configuration every other generation. Generally 100 generations are allowed to complete. Run time statistics available at job completion show that over 400 queries per second are being executed on a lightly loaded VAX 11/780.

5 CURRENT APPLICATIONS USING THE SIMULATOR

The following is a list of some current applications using the simulator.

Life Game - Query Version - This application, detailed above, is typically used as a test case for Time Warp because of its use of both event and query messages. It executes in a lock step manner which helps in early development and debugging.

Life Game - Event Message Version - This implementation of the Life game has a live cell send its state to its neighbors via an event message. No querying is done and it is therefore much more efficient than the query version.

Army Command and Control Simulation Model of Message Processing and Handling by Various Staff Elements. This application, a work unit of the original Time Warp/Hypercube project funded by the sponsor, has been implemented using the simulator and will be the first non-trivial application running on the hypercube under the Time Warp mechanism. This military application creates about 60 object instances.

Data Flow modeling of Military Intelligence Processing. This simple model has been used to support the development of a prototype military intelligence workstation.

Hopfield Associative Memory Model.[8] This application is a software implementation used to study the properties and structure of the Hopfield Associative memory and its applications including Expert Systems.

Knowledge Network Model. A distributed architecture containing expert system, data base management, user interface and conventional algorithmic processing objects is being modeled to study its characteristics and performance properties prior to prototyping.

References

1. "The Cosmic Cube", C.L. Seitz, CACM Special issue on architecture. January 1985
2. "Fast Concurrent Simulation Using the Time Warp Mechanism", D. Jefferson (UCLA), H. Sowizral (Rand), SCG conferences on Distributed Simulation, San Diego, January 1985.
3. "Implementation of Time Warp on the CalTech Hypercube", D. Jefferson (UCLA) B. Beckman and group from JPL, SCG conference on Distributed Simulation, San Diego, January 1985
4. "Caltech/JPL Concurrent Computation Project - Annual Report 1984", G. Fox, February 1985.
5. "Design Specifications for Time Warp Entry Points", B. Beckman, JPL Internal Documents, 1984 -1985.
6. "Time Warp Simulators User's Manual", S. Hughes, V. Warren, K. Sturdevant, JPL Internal Document, September 1985.
7. "Object-Oriented Programming: Themes and Variations", M. Stefik and D. Bobrow, The AI Magazine, Winter 1986.
8. "Automated Deduction/Neural Net Accomplishments", J. Spagnuolo, JPL Internal Document, May 1985.

POSTER PAPERS

Distributed Batch Queues On MicroVAX IIs

M. A. Oothoudt, J. F. Amann, and M. V. Hoehn
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

ABSTRACT

The price/performance ratio of the MicroVAX II makes its use as a batch job "compute engine" very attractive. We have developed software to allow a user to submit batch jobs to a single queue on a host VAX and have those jobs run on any available VAX attached to the host with DECNET. The host VAX acts as the central queue manager for the remote nodes and provides mass storage for the jobs running on them. The remote microVAX IIs access data files on host disks via DECNET and have sufficient local disk space for VMS, user executables, and limited amounts of user run-time storage. For jobs typical of our environment, a single VAX-11/780 host should be able to service at least 20 MicroVAX IIs.

INTRODUCTION

The Clinton P. Anderson Meson Physics Facility will soon be using three VAX-11/750s and six MicroVAX IIs for medium energy physics data acquisition. These machines will be used for data acquisition for six months of the year but will be idle much of the remainder of the year. Since these nine machines have a combined processing power equivalent to the two VAX 8600s in our Data Analysis Center (DAC), we wished to find a simple way to add their processing power to the DAC during the idle periods.

The chosen solution is to off-load batch jobs from the 8600s onto the "farm" of remote VAXs, leaving the 8600s free to handle interactive jobs. An 8600 "host" serves as a "queue manager" for the remote machines and as a "disk server" so that massive data files need not be located on the relatively small (71 MB) MicroVAX local disks. Each batch job runs on a single remote VAX, accessing any necessary data files over DECNET. Aside from usage of idle CPU power, this solution has the added advantage of getting significant throughput for batch jobs in the daytime when the 8600s are almost entirely consumed by interactive compute-bound jobs.

COMPONENTS

To run this VAX farm system, VMS V4.2 (or later) and DECNET are required on all nodes. Ethernet is used as the physical connection for DECNET at our facility, but any communications link supported by DECNET could be used.

The farm software running at our

facility consists of

1. DCL command procedures for the system manager to initialize the farm system and for the user to run jobs in it.
2. A small disk "data base" on the host machine. This file contains a list of remote nodes and the status of any jobs running on those nodes. The file is owned by the SYSTEM; the WORLD has only READ access to it.
3. Batch queues. In the host a batch queue SYS\$FARM: contains a job for each user job running on a remote node. This host job controls and monitors the user job on the remote node; it uses very little CPU time (e.g. 5 minutes of host time to control a 15 hour user job on the remote CPU). Because our DAC is a Cluster we also have a generic queue so that jobs may be submitted to the execution queue from any cluster node. On the remote node a batch queue runs the user's job.
4. Four images. One of these images must be installed with SYS\$PRV privilege so that it may update the farm data base. The remaining images require no special privileges beyond that associated with a normal account (TMPMBX and NETMBX).

(An early version of the software was implemented as a user-written server symbiont. However, server symbionts are only possible for printer or terminal queues and in such queues there can only be one active job [ie. one active remote node per queue] at a time. The inconveniences of a server symbiont were not compensated by any advantages, so the symbiont implementation was dropped.)

USER INTERFACE

A user who wishes to submit jobs to the host farm queue must execute a farm command file (usually from his login command file) to initialize his process. The primary effect of this command procedure is to define a symbol S*UBMIT which points to farm command file SUBMIT_FARM.COM.

When the user submits a job, SUBMIT_FARM checks to see if the job is intended for queue SYS\$FARM; if not, it does a normal DCL SUBMIT with the user's command line. Farm jobs have farm command procedure FARMER_BATCH.COM prefixed to the list of user batch command files, and the resulting job is submitted to SYS\$FARM. All of this is done transparently to the user.

Once the user's job has started running on a remote node, the user can run image FARM_VIEW to list all nodes in the farm system and the status of executing jobs (e.g. amount of CPU time used). If the user needs more detailed knowledge about his job, he can login at the remote node. Otherwise all of his operations are done on the host.

EXECUTION LOGIC OF A FARM JOB

When a farm job starts executing in a host queue, command file FARMER_BATCH is run and it, in turn, executes image FARMER. FARMER gets the user's SUBMIT command information from the Job Controller, allocates a remote node from the farm data base, and passes the job information to the remote node. FARMER then hibernates; when the remote node sends status information for the remote job, FARMER wakes and updates information in the data base. When the remote job terminates, FARMER deallocates the node in the data base and terminates the host batch job.

In the remote node the user's job executes in a normal batch queue. Data files on the host machine disks may be accessed via standard DECNET remote file access; e.g. from FORTRAN

```
OPEN (UNIT=1,NAME='host_node::  
1 DUAL:[USER.DAT]RUN972.DAT',...)
```

Because of problems under VMS V4.2 (see below), the user's batch command file must transfer all user executable images to the remote nodes's local disk.

In the remote node a farm system network task monitors the progress of the user's job and periodically reports back to the host.

PERFORMANCE

The performance of the farm system can be measured in terms of its effects on the host machine, the Ethernet DECNET link, and the remote nodes. We assume that jobs run on this system are "CPU bound". By this we mean that a job running on an otherwise idle system has essentially zero NULL time. (The farm system does, however, support

significant data transfer capability; a typical job processes 30 MB of data per hour of microVAX II CPU time.) Since we do not yet have a full complement of farm nodes, we estimated these effects as follows.

The test system consisted of a VAX-11/780 host with disks on an HSC-50. A VAX-11/750 and a VAX 11/780 were the remote nodes with an Ethernet link used for DECNET communication. The effects on the host are primarily due to reading of data files by the remote job from the host disk. We ran an essentially IO-bound job on the remote nodes which read 3 KB records from a sequential unformatted file on the host disk. We then measured the fraction of CPU time consumed on the host to service these read requests. We found that approximately 34 milliseconds of host CPU time were required per record transferred. This time was essentially independent of the number of remote nodes doing READs or the rate at which the READs occurred. Approximately 55-60% of that time was accounted for by the "FAL" process created on the host to service the network requests. The remainder of the time was spent on the interrupt stack and in KERNEL mode. Since previous measurements have shown that a MicroVAX II typically requires 700 milliseconds to process that same 3 KB buffer we conclude that a dedicated VAX-11/780 host could support more than 20 MicroVAX II's in such an environment.

To measure the overhead on the remote node we compared the time on a node required to read and process data read from a local disk as compared to data read over the network. We found that both wall clock time and CPU time increased by roughly 5% when data was read over the network. Even when data was read over the network the job on the remote node remained essentially compute bound (<1% NULL time).

Even at the highest data rates we do not expect that the load on the Ethernet will be important. At the highest rates measured, data was being transferred over the network at 77 KB/sec using approximately 85% of the host 11/780 CPU time in doing so. Extrapolating this to 100% usage gives an upper limit of roughly 90 KB/sec or 7% of the Ethernet bandwidth at 10 Mbit/sec. If run as a dedicated host, a faster CPU such as the 8600 could conceivably handle more remote nodes and put a heavier burden on the network. It is unlikely however that we will ever operate in such a mode due to heavy demand for time on our 8600's.

PROBLEMS

The farm system was implemented under VMS V4.2 where commands like

```
$ RUN host_node::file_specification
```

on a remote node cause access violations if the image was linked using the LINKER CLUSTER option. Since a majority of our applications use this option, we were forced to require the user to copy all executable images to the remote node's local disk. As

a side effect, accounts needed to be created on the remote disks for every user of the farm system.

Under VMS V4.3 the above problem does not exist and we are investigating whether we can leave the user's executable images on the host disks. This is a very important consideration for the MicroVAX systems, which have limited disk space.

CONCLUSIONS

For the last three months batch jobs have been running almost continuously on a farm consisting of two VAX-11/750s. When our six new microVAXs arrive they will be added into our farm.

For year-round batch support, several minimally configured MicroVAX II systems will be purchased for installation in the DAC. The minimal configuration will consist of a MicroVAX II in a rackmount BA23 box with 4 MB of addon memory, a 30 MB disk, and an Ethernet link to the host 8600.

ACKNOWLEDGEMENTS

This work was supported by the U.S. Department of Energy under Contract W-7405-ENG-36.

**1986 NORTHEAST DECUS
REGIONAL CONFERENCE**

ADDING DEVICES TO RSX WITHOUT A SYSGEN

Dennis P. Costello
National Research and Resource Facility
for Submicron Structures
Cornell University
Ithaca, NY

ABSTRACT

Performing an RSX Sysgen can be a painful and lengthy procedure. This is generally due not to the Sysgen process itself, but to the work which surrounds Sysgen. It may be necessary to regenerate DECnet or other privileged software, the RSX sources may not be present on the target machine, or some other difficulty may arise.

Generally, adding a new device to an RSX system requires that Sysgen be performed. This paper describes a technique for building a loadable device driver using sources provided by DEC. A command file which substantially automates this process is listed, and has been submitted to the spring 1986 RSX SIG tape.

No particular knowledge of device drivers is assumed, but the reader should be familiar with the Sysgen process and the function of the various components of RSX.

Introduction

There are two types of device drivers in RSX; resident and loadable. Early versions of RSX allowed only the simpler of these, the resident driver. The resident driver is a set of Executive subroutines--the driver code and device database are included within the Executive address space. RSX version 3.2 introduced loadable drivers, in which the driver code is isolated into a special type of privileged task, and therefore removed from the Executive address space. This greatly facilitated the debugging and patching of device drivers, whether user written or supplied by DEC, because a change in driver code no longer necessitated a new Sysgen. All device drivers refer to device databases, which are collections of control blocks that describe each I/O device known to the system. Device databases may be either resident or loadable. Resident drivers always have resident databases, but loadable drivers may have either variety of database. A resident database is created during Sysgen as part of the Executive (located in Executive module SYSTB, *not* in pool), and

therefore cannot be changed without relinking the Executive. By contrast, a loadable database is created when its associated driver is built, and is copied into pool when the driver is loaded. Thus, a loadable driver with a loadable database can be added to an RSX system after Sysgen has been completed. In either case, the database is within the Executive address space, which allows easy access from privileged tasks and the Executive itself. When Sysgen builds a loadable driver, the driver always has a resident database. The driver built by the procedure described here will have a database which is identical to what Sysgen would have created, except that it will be loadable. Figure 1 illustrates the relationships between resident and loadable drivers and databases, and Figures 2-4 illustrate the three legal combinations of driver and database types. The only prerequisites for this procedure are that loadable driver support, ANSI magtape support (if desired), and the desired terminal driver features (if the terminal driver is being rebuilt) must have been selected during the original Sysgen. Device databases are comprised of three different types of

control blocks: the Device Control Block (DCB), the Unit Control Block (UCB), and the Status Control Block (SCB). The DCB contains information common to all devices of a given type, that is, all devices connected to the same type of controller. For example, there is one DCB for all DUnn disks in the system, one for all MSnn tape drives, but there may

be several for TTnn terminal lines, since there is a separate DCB for each type of terminal controller (DL11, DZ11, DH11). The UCB contains all the information necessary to control an individual device, so there is one per device. The SCB contains information on an individual I/O operation. There is usually one SCB per controller. Figures 5, 6, and 7 illustrate the relationships among these control blocks. The DCBs form a linked list, pointed to by \$DEVIID. Each DCB also contains a pointer to the UCB for the first device of that type. UCBs of a given type are always contiguous, therefore there is no need for pointers from one UCB to the next. However, each UCB does have a pointer to the SCB associated with that unit. Figure 5 shows a device database for an RL211 controller with four RL01 or RL02 disk drives attached. There is a single DCB for these drives, four UCBs, and a single SCB. The RL211 controller does not support more than one I/O operation at a time, regardless of the number of attached drives. In this case, one SCB is sufficient. Figure 6 shows a database where a second RL211 controller is added to the situation of Figure 5. There is still just one DCB, since both controllers are of the same type. The number of drives has not changed, so neither has the number of UCBs. But there is an additional SCB, since there can now be one I/O operation active simultaneously on each of the two controllers. Figure 7 shows a database for four terminal lines. There is a single DCB, which indicates that all the lines are attached through the same controller type. There are four UCBs, each of which points to a separate SCB. This could be either separate DL11 terminal controllers, or a terminal multiplexer such as the DZ11. Separate SCBs are provided for terminal lines because multiplexers such as the DZ11 can support simultaneous I/O operations on all attached lines.

Procedure to Build Loadable Device Driver - General Case

The general procedure for building a loadable device driver with a loadable database consists of the following steps:

- o Write the driver code;
- o Write the code for the device database;
- o Assemble these source files, and link them together;
- o Remove the old device database, if it exists;
- o Load the new device driver and database, using the MCR LOAD command;
- o Test the driver thoroughly; and
- o Load the driver using the VMR LOAD command, so it is available at every system boot.

The most difficult of these steps is writing the driver and database code. The procedure described here substantially automates these steps. The obvious replacement for the first step is to copy the existing DEC driver code, which can be found in [11,10]xxDRV.MAC.

NOTE

xx is the device mnemonic of the driver being built

Sysgen calls the command file [200,200]SGNPER.CMD to produce resident device database code. To generate loadable device database code, I have several times "executed" this command file manually. That is, I calculated the value of each variable and deduced what output SGNPER would have generated, had it been called by Sysgen in the normal way. While this is preferable to writing the database code from scratch, it is still a fairly long and error-prone procedure. To automate the process further, I wrote a command file, TABBUILD.CMD, which calls SGNPER.CM for me. TABBUILD impersonates the portion of SYSGEN.CMD which calls SGNPER.CMD. It first reads the files SYSSAVED.CMD and SGNPARM.CMD, which were produced by the original SYSGEN, and uses them to set a number of global symbols appropriately. It then opens the files which SGNPER will act upon, and invokes SGNPER. SGNPER then writes a new copy of the device database, which is identical to Executive module SYSTB with the exception of the added devices. The database code for the individual driver is then extracted from this file using your favorite editor and used to build a loadable driver with loadable database, which can then be added into the system. If SGNPER were coerced into producing only the xxDRV database without all the other code, it would not be able to take into account the rest of the devices on the system when calculating floating vector and CSR addresses, and is more likely to get them wrong. That is why it is better to let SGNPER produce the entire database and delete the unwanted portions.

**Procedure to Build Loadable Device Driver
- using TABBUILD**

Please note in the following procedure that if the loadable database which is being produced will replace an existing database, the existing database must be removed before the loadable driver is loaded. Otherwise, the existing database will be used instead of the new one, and nothing will have been accomplished. In order to clarify the discussion of the procedure, this case is examined in a later section. Similarly, the extra steps involved when the database being added to the system is for a DU or MU device, or for a DY device on a 22-bit Qbus system, are also explained in separate sections.

NOTE

In the following, [ggg,mmm] is a scratch directory, and xx is the device-type mnemonic (for example, DL for RI.01/RL02 disks).

1. Back up the system disk before proceeding, then copy all files needed into a scratch directory and set that as your default directory. Since SGNPER modifies certain system files, reasonable caution dictates that these changes be made to copies of these files. The files needed are:

```
[ 11, 10 ] RSXMC.MAC
[ 11, 10 ] xxDRV.MAC
[ 200, 200 ] SYSSAVED.CMD
[ 200, 200 ] SGNPARM.CMD
[ 200, 200 ] SGNPER.CMD
TABBUILD.CMD (in Appendix A;
              also on SIG tape)
```

NOTE

If an existing database will be replaced, it may be necessary to edit SYSSAVED.CMD now. See the section "REMOVING AN EXISTING DATABASE THROUGH POOL SURGERY" for more information.

2. Invoke TABBUILD.CMD. It will ask for the mnemonic of the device controller to be added to the system, and the number of such controllers. SGNPER will write some comments to the terminal, and may ask some questions regarding the controller(s) and devices to be added. Answer these as you would during a real Sysgen.

3. Inspect the files SYSSAVED.CMD, SGNPARM.CMD, and RSXMC.MAC. SGNPER will have appended lines to the end of these files, describing the entire device configuration of the system. These lines should be duplicates of the definitions originally written by the Sysgen. This should not be a problem, since the variables will simply be redefined. Examine closely any new lines added to these files. Pay particular attention to lines in RSXMC.MAC which add or change variable definitions. These variables generally control the generation of conditional Executive code (to support driver features such as ECC). Since a Sysgen will not be performed, this code will not be generated, and the driver may not work correctly. In general, try to understand the purpose of all the lines which are added to these files, and verify that nothing has gone wrong so far.

4. Edit the file xxTAB.MAC. This is what SGNPER.CMD would have written as SYSTB.MAC during a real SYSGEN, and contains the databases for all devices in the system, including pseudo devices, such as SY: and NL:. Remove all extraneous material from this file and change the first field of the xx DCB, leaving the file in the following format:

```

                                .TITLE xxTAB
;
; comments: Digital copyright
; notice, etc.
...
; DEVICE TABLES
;
$xxDAT: :
;
                                .WORD 0
                                { other DCB fields }

;
                                { UCB's }

;
                                { SCB's }

$xxEND: :
                                .END
```

- Assemble the driver and databases source files and taskbuild the driver:

```
MAC xxDRV=[ 1, 1 ] EXEMC/ML, [ ggg, mmm ] RSXMC/PA: 1, xxDRV
MAC xxTAB=[ 1, 1 ] EXEMC/ML, [ ggg, mmm ] RSXMC/PA: 1, xxTAB
TKB @xxDRVBLD.CMD
```

Where xxDRVBLD.CMD contains:

```
[ 1, 54 ] xxDRV, , [ 1, 54 ] xxDRV=[ ggg, mmm ] xxDRV, xxTAB
[ 1, 54 ] RSX11M.STB/SS, LB:[ 1, 1 ] EXELIB/LB, LB:[ 1, 1 ] SYSLIB/DL
/
STACK=0
PAR=DRVPAR: 120000: length
/
```

In the above, length is taken from the following table:

Device	Length
DU, MU	20000
XE, TT	40000
Others	14000

NOTE

If the device driver being built is DU or MU, or DY on a 22-bit Qbus system, see the appropriate section for further information.

- LOAD the new driver and database, using the MCR LOAD command. If space does not exist in DRVPAR, you may need to use the /PAR and /HIGH switches on the LOAD command.

NOTE

If an existing database is to be replaced, it must be removed before the new driver is LOAded. See the section "REMOVING AN EXISTING DATABASE THROUGH POOL SURGERY". If the device driver being built is DU or MU, or DY on a 22-bit Qbus system, see the appropriate section for further information.

- Test the new driver. If it is a disk driver, for each drive initialize a volume, create some directories and files, and read and write some files on the disk. Note that INITIALize is not in itself an adequate test of a disk driver, since it is possible to build a database which allows INI to work normally, but does not allow the disk to be mounted or files to be created on it. I have in fact made that mistake myself. If is a tape driver, for each drive initialize a tape, mount it, and write some files with COPY or PIP. Read the tape on a system which is known to support ANSI tapes (preferably another RSX or VMS system). If it is a terminal driver, plug a terminal into each port in turn, log in through that port, and execute several commands. Only when you are convinced that everything works should you proceed to the next step.

8. Install the new driver permanently, using the VMR LOAD command. You may wish to modify SYSVMR.COM to include the LOAD command, create a new virgin RSX11M.SYS, and run VMR @SYSVMR. This way, the partition layout can be changed so the new driver will fit into DRVPAR. Up to this point, the bootable system image has not been changed, so backing up to recover from an error has required nothing more than booting the system, and repeating the appropriate steps. But once the system image is modified, any errors might make the image unbootable, forcing you to restore the system disk from backup tapes and start all over again. A conservative approach would be to perform a software boot using the BOOt command and test the modified system, before using SAV/WB to make the new image hardware bootable. This is particularly true if an existing database was removed.

In step 6, PUCOM must be installed into a common region. If there is a hole in the memory partition layout of at least 75 (octal) memory blocks, the following two commands can be issued from MCR (with the * replaced with the starting block number of the hole). Otherwise, they must be added to SYSVMR.COM, and a new system image built.

```
SET /MAIN=PUCOM:*:75:COM
INS PUCOM
```

Special Considerations for DU and MU Drivers

These drivers refer to extra support code which is collected into a common region called PUCOM. This must be assembled, taskbuilt, and loaded. In step 5, issue the following extra commands:

```
MAC PUCOM=[ 1, 1 ] EXEMC/ML, [ ggg, mmm ] RSXMC/PA: 1, [ 11, 10 ] DSAPRE/PA: 1, PUCOM
TKB @PUCOMBLD
```

Where PUCOMBLD.COM contains:

```
[ 1, 54 ] PUCOM, , [ 1, 54 ] PUCOM=[ ggg, mmm ] PUCOM, [ 1, 54 ] RSX11M. STB/SS
/
STACK=0
UNITS=0
PAR=PUCOM: 140000: 20000
/
```

Special Considerations for DY Drivers on 22-bit QBUS Systems

This driver requires a common region, DYCOM, located within the first 124k words of memory, for buffering I/O to and from the controller (which can only generate 18 bit addresses). This must be assembled, taskbuilt, and loaded. In step 5, issue the following extra commands:

```
MAC DYCOM=[ 1, 1 ] EXEMC/ML, [ ggg, mmm ] RSXMC/PA: 1, DYCOM
TKB @DYCOMBLD
```

Where DYCOMBLD.CMD contains:

```
[ 1, 54 ] DYCOM, , [ 1, 54 ] DYCOM= [ ggg, mmm ] DYCOM
/
STACK=0
PAR=DYCOM: 0: 0
/
```

In step 6, DYCOM must be installed into a common region. If there is a hole in the memory partition layout which can be used, the following two lines can be issued from MCR (with the * replaced with the starting block number of the hole). Otherwise, they must be added to SYSVMR.CMD, and a new system image built.

```
SET /MAIN=DYCOM: * : length: COM
INS DYCOM
```

Where length is the number of DY controllers times 20 (octal).

Removing an Existing Database Through Pool Surgery

If an existing database is being replaced, the old device configuration answers in SYSSAVED.CMD may need to be changed or deleted. Generally, if a controller is being added, and the devices attached to the existing controller(s) have not changed, nothing need be done at this stage. There will be no predefined answers to the questions for the new controller, and SGNPER will simply ask the appropriate questions as it is being run. However, if the configuration of devices attached to the existing controller(s) changes, the data in SYSSAVED.CMD will be incorrect. The appropriate lines must be either corrected or deleted. SGNPER will then either use the corrected answers, or ask the appropriate questions for all controllers of that type. The symbols to be altered are located near the end of SYSSAVED.CMD, in the section labeled "Peripheral Configuration". String symbols are defined for each controller, and generally for each device. These are named \$\$xxn (for controllers) or \$nxxm (for devices). Note that n is the controller

number, and m is the device number. Thus, 2 RL02s on a single controller would be defined by the symbols \$\$DL0, \$0DL0, and \$0DL1. These string symbols each contain a set of fields separated by commas. Generally, the last field of the controller symbol \$\$xxn is the number of devices attached to the corresponding controller. You should either delete all these symbol definitions, or edit them appropriately. Before the newly-built driver is LOAded, the old device database must be removed from the system. Since the UNLoad command removes only the driver, not the database, extra steps are required at this point. Loadable databases may be easily removed by either booting the system (if the driver was loaded via MCR) or by building a new system image with VMR (if the driver was loaded via VMR). Resident databases (such as are produced by Sysgen) must be removed using a technique I call pool surgery. (Purists please note that I am operating on objects in SYSTB and not in pool, so this could as easily be known as SYSTB surgery. 'Pool surgery' has a nicer ring to it.) The technique is as follows:

- o Make sure no one else is using the system. Any conflicts while performing pool surgery can corrupt the system in unpredictable ways.
- o Make sure that all devices whose database will be removed are idle, have no open files, have no checkpoint space allocated, are not mounted (marked for dismount is not sufficient), have no tasks installed from them, and that the associated driver has been UNLoaded. Note that it is not possible to remove the database for the system disk or terminal drivers, since either all the terminals will go dead, or the system will lose communication with its system disk. If this happens, you'll have to reboot and start the pool surgery over. These cases require some extra steps, which are described at the appropriate time.
- o Look at [1,34]RSX11M.MAP to find the location of Executive symbol \$DEVHD. If the MAP file is not available, use DMP to examine [1,54]RSX11M.STB. Each record of the .STB file has several symbol definitions, whose format is shown in Figure 8. Dump the file, twice, once looking for the name of the symbol, and once looking for its location, using the following commands:

```
DMP TI :=RSX11M.STB/R5/RC
DMP TI :=RSX11M.STB/RC
```

- o Use the MCR OPEN command for all subsequent steps, except as noted.
- o OPEN the address of \$DEVHD to find its value. This is the pointer to the first DCB. Use the @ operator of the OPEN command to follow the DCB linked list to its end (which is denoted by a link value of 0).
- o Look at the ASCII value of the third word of each DCB. This will contain the device mnemonic for the DCB. Find the DCB(s) which must be removed (for the terminal driver, you may need to remove all TT DCBs).

- o Remove the DCB from the linked list by resetting the link pointer on the preceding DCB to point to the following DCB. If the DCB is not for the system disk or terminal drivers, OPEN may be used to make this change on the running system. Otherwise, ZAP must be used to make these changes on a copy of the system image file, RSX11M.SYS.
- o From MCR, confirm that the device database has been removed, using the DEV command. Then LOAD the driver, and test it as in the normal procedure.
- o If ZAP was used above, these steps must be done on the copy of the system image using VMR commands. The modified system image is then booted using the MCR BOOT command, at which time the new driver(s) will be loaded and available for testing. Once you are satisfied, use SAV /WB to make the modified system image the hardware bootable system. If you do this before the driver is adequately tested, or use ZAP on the original system image (rather than a copy), any error will most likely render the system disk unusable.

NOTE

Do *not* attempt to reclaim the space occupied by the old database. If you add this space to pool, the pool allocation routine will force a system crash when it first encounters this block, since the old database is located outside pool's address limits.

Appropriate Applications

This technique described in this paper is especially appropriate for adding a new type of controller to a system. It has been successfully used several times. In one case, an RSX Sysgen was performed on a VAX for a newly acquired PDP-11/23. This system had dual RX02 floppies and a Winchester disk which emulated several RK06s. A bootable floppy was required to move RSX to the target system. The only RSX system available that had an RX02 drive (and thus could make a bootable floppy) did not have a DM driver for the RK06/Winchester. A DM driver was added to the RX02-based RSX system, and this was then used to copy RSX onto the target system's RK06 system disk. In another case, a newly arrived PDP-11/44 had an RL02 drive, an RA80 drive, and an Ethernet connection, but no tape drive. Existing RSX systems on the same Ethernet also had RL02's but no tape drives or RA disks. The only RSX distribution kit was on magtape. The only tape drive was on a VAX. The problem was how to move the distribution kit onto the RA80 so a Sysgen could be done. To do so, I started by copying the RL02 system disk from one of the other RSX systems, and booting it on the 11/44. Then, the network node name and address were changed to avoid conflicts, and the network was brought up. A DU disk driver was added to the system, and the RA80 was initialized and mounted. The distribution kit was read onto a disk on the VAX. The RSX distribution kit was copied over the network onto the 11/44's RA80, VMR was used to make the RA80 bootable with the baseline system, and a normal Sysgen was performed. A TU80 tape drive was added to the same system months later. A MS driver was added to the system, which allowed the tape drive to be used for BRU. The common thread of these three examples is that no new executive features were required, so I was able to get around some rather sticky problems. In particular, the systems which I was modifying had all had loadable driver support selected when they were Generated. Also, a new controller type was added to the system in each case, and therefore it was not necessary to remove a device database. Other situations in which the technique might be particularly helpful involve turnkey systems. Performing a Sysgen on such a system might not be possible, since the sources or objects necessary to re-build privileged user tasks might not be available. The user could use this technique to avoid a Sysgen when new hardware is added. Or the vendor could use it to make loadable drivers in advance. This would be preferable to common practice among such vendors of generating "one of everything" into their systems, since it frees up space for pool that is otherwise taken up by unused device databases.

Inappropriate Applications

This technique will not be of any use at all in two situations:

- o Adding a tape drive to a system without ANSI Magtape support. Extra code is needed in the Executive to support the use of MOUnT, PIP, COPY, DIR, etc. on tape drives. If ANSI Magtape support was not selected during the original Sysgen, this code will not be present.
- o Changing from half-duplex to full-duplex terminal driver, since the full-duplex driver requires executive code which the half-duplex driver does not. Changing the set of terminal driver features supported will probably not work either, since many of these features also involve conditional support code in the Executive.

Conclusion

Loadable device drivers with loadable databases provide a comparatively easy, if seemingly obscure, way to add support for a newly acquired I/O device to an RSX system. The entire procedure described in this paper can be accomplished in a few hours, as opposed to the several days that might be required to perform a full Sysgen and Netgen, and to re-build privileged user tasks.

Acknowledgements

This work was supported by the National Science Foundation under Grant ECS-8200312 to the National Research and Resource Facility for Submicron Structures. The author also wishes to acknowledge the assistance of Lindy Costello in the preparation of this paper, and of John Koumjian in the preparation of the slides which accompany it.

Device Driver Types

	DATABASE	
DRIVER	RESIDENT	LOADABLE
RESIDENT	DEC STANDARD RESIDENT	NOT POSSIBLE
LOADABLE	DEC STANDARD LOADABLE	TOPIC OF THIS PAPER

Figure 1

Resident Driver, Resident Database

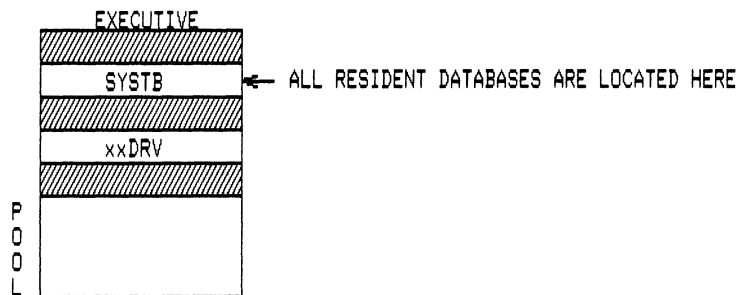


Figure 2

Loadable Driver, Resident Database

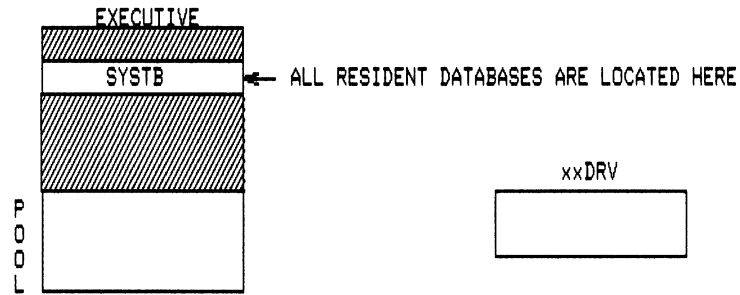


Figure 3

Loadable Driver, Loadable Database

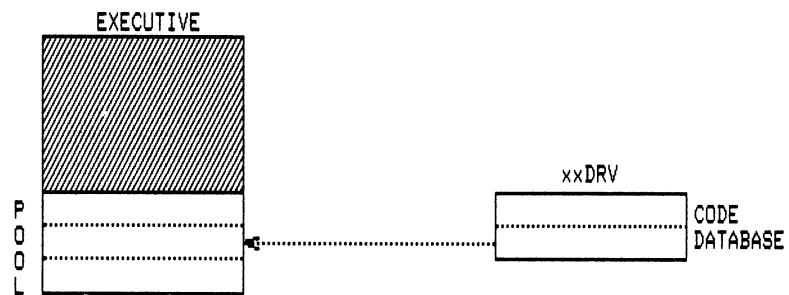


Figure 4

Device Database Structure

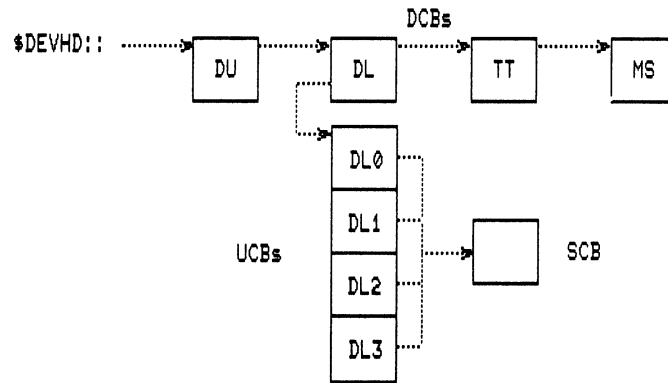


Figure 5

Device Database Structure

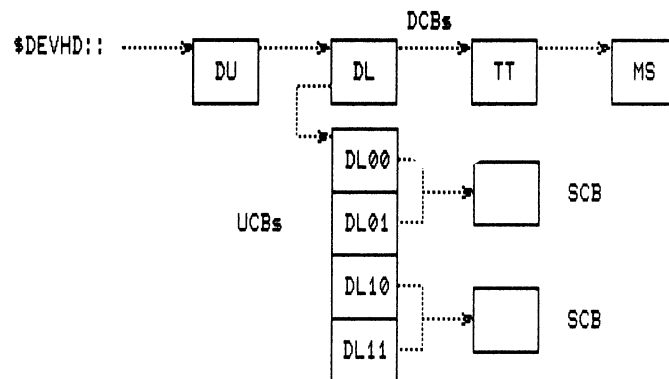


Figure 6

Device Database Structure

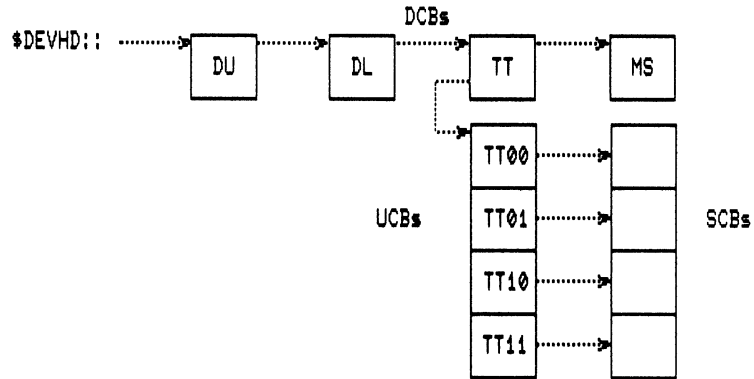


Figure 7

Finding Address of \$DEVHD

FORMAT OF RECORDS IN RSX11M.STB

S.Y.M.	SYMBOL NAME IS 2 WORDS IN RAD50 FORMAT
B.O.L.	
ADDRESS	ADDRESS IS 1 WORD
S.Y.M.	
B.O.L.	
ADDRESS	
...	

Figure 8

APPENDIX A

TABBUILD.CMD

This command file can also be found on the Spring 86
RSX SIG Tape.

```
; TABBUILD.CMD
;
; Author: Dennis P. Costello
;       National Research and Resource Facility for Submicron Structures
;       G02 Knight Lab
;       Cornell University
;       (607) 255-2329
;
; Copyright (c) 1986. This command file may be reproduced without charge
; provided this copyright notice remains intact.
;
; This command file at the attendant procedure for its use are believed to be
; correct, however, neither the author, the Submicron Facility, nor Cornell
; University acknowledge any liability for any damages arising from its use.
; In particular, the user is strongly encouraged to read and understand the
; paper "Adding Devices to RSX without a Sysgen", published in the Spring 86
; DECUS Proceedings, before proceeding. A copy is included here as PAPER.RNO.
.;
.; Prefix file for calling SGNPER.CMD. Sets global symbols appropriately, opens
.; files that are expected to be open, etc., and then calls SGNPER.
.;
.; Assumes that the files SYSSAVED.CMD and SGNPARM.CMD are in the current
.; directory, and that they are copies of the file produced by SYSGEN.
.;
.; Also assumes that SGNPER.CMD is in the current directory. This should be run
.; in a scratch directory, to reduce the chance of corrupting any system files.
.;
.; Will append to RSXMC.MAC, so make a backup copy first.
.;
.; If you have the RL02 kit, $EXC should be defined as the disk on which EXCPRV
.; is mounted.
.;
.; For RSX-11M only, NOT RSX-11S, MICRO/RSX, P/OS, or RSX-11M+
.;
.; Get name of device to be added
.;
    .DISABLE LOWERCASE
    .ASKS XX [2:2] Enter name of device to be added (e.g., DL, MS)
    .ASKN [1:9.:1] XXN Enter number of controllers of type 'XX'
.;
.; Read saved answer file
.;
    .SETF $SGN1
    .SETF $SGN2
    @SYSSAVED
    @SGNPARM
.;
.; Set other global variables
.;
    .SETN $SYGRP 1
    .SETN $UIC 11
    .SETN $GRP 0
    .IFT $MAP .SETN $GRP 4
    .TEST <UIC>
    .SETS $DFUIC <UIC>[2:<STRLEN>-1]
    .SETS $SGNUC <UIC>[2:<STRLEN>-1]
    .SETS $LST ""
    .IF $ALD NE "NL:" .SETS $LST "'$ALD'[,3'$GRP']"
    .SETS $EXC "SY:"
```

```

.SETS $DRV $EXC
.SETS $DIR ["`$UIC`,2`$GRP`"]
.SETS $PREFIX "LB:[`$SYGRP`,1]EXEMC/ML,SY:[`$UIC`,10]RSXMC"
.SETS $XEPRE "LB:[`$SYGRP`,1]EXEMC/ML,DEUNA/ML,SY:[`$UIC`,10]RSXMC"
.SETS $UICMS ["`$SYGRP`,5`$GRP`"]
.SETF $ACDFD
.SETF $ACL
.IFT $DCL .OR .IF $NUC GT 0 .SETT $ACL
.SETS $SYS "M"
.SETF $FUD
.IF $TTY EQ "C" .SETT $FUD
.SETT $PERIP
.SETS $SF1 ""
.SETT $SAVE
.SETT $SAVED
.SETS $SAO "SYSSAVED.CMD"
.SETF $QBUS
.IF $TPR EQ "11/03" OR $TPR EQ "11/23" OR $TPR EQ "11/73" .SETT $QBUS
.IF $TPR EQ "11/83" OR $TPR EQ "LSI-11/73" OR $TPR EQ "KXJ11" .SETT $QBUS
.SETT $XPRES
.SETF $PTW
.;
.; Change number of controllers in $DEV, $DV2, $DV3 to show new device
.;
    .SETS STRING "$DEV"
    .GOSUB ADDDEV
    .SETS STRING "$DV2"
    .GOSUB ADDDEV
    .SETS STRING "$DV3"
    .GOSUB ADDDEV
.;
.; Open the various files that SGNPER expects to write to
.;
    .OPENA #0 SYSSAVED.CMD
    .OPENA #1 RSXMC.MAC
    .OPEN #2 `XX`TAB.MAC
    .OPEN #3 `XX`DRVASM.CMD
.;
.; Write header records in SGNPER's output files
.;
.DATA #2 .TITLE `XX`TAB
.ENABLE DATA #2
    .IDENT /M4.2/
;
; COPYRIGHT (c) 1983, 1985 BY
; DIGITAL EQUIPMENT CORPORATION, MAYNARD
; MASSACHUSETTS. ALL RIGHTS RESERVED.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY
; BE USED AND COPIED ONLY IN ACCORDANCE WITH THE
; TERMS OF SUCH LICENSE AND WITH THE INCLUSION OF THE
; ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
; COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON. NO TITLE TO
; AND OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERRED.
;
; THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO
; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
;

```

```

; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR
; RELIABILITY OF ITS SOFTWARE ON EQUIPMENT THAT IS
; NOT SUPPLIED BY DIGITAL.
;
;
; VERSION M4.2 BASELEVEL 38
;
; CREATED BY SYSGEN VERSION 3.00
;
; SYSTEM TABLES
;
; MACRO LIBRARY CALLS
;
; .MCALL HWDDEF$,SCBDEF$,UCBDEF$,CLKDEF$
; HWDDEF$ ;DEFINE HARDWARE REGISTERS
; SCBDEF$ , ,SYSDEF ;DEFINE SCB OFFSETS
; UCBDEF$ , ,TTDEF ;DEFINE UCB OFFSETS
; CLKDEF$ ;DEFINE CLOCK QUEUE OFFSETS
;
; LOCAL ASSIGNMENTS
;
; UMD=0
; .IIF DF D$$IAG,UMD=400 ;DIAGNOSTIC FUNCTIONS BECOME LEGAL
; ERL=0
; .IIF DF E$$DVC,ERL=4
;
; DEVICE TABLES
;
; .DISABLE DATA #2
; .DATA #2 '$XX'DAT::
;
; ; Call SGNPER to setup the device tables
;
; @SGNPER
; SET /UIC=['$DFUIC']
;
; ; Write trailer records in SGNPER's output files
;
; .DATA #2 '$XX'END::
; .DATA #2 .END
;
; ; Close the files written by SGNPER
;
; .CLOSE #0
; .CLOSE #1
; .CLOSE #2
; .CLOSE #3
;
; ; All done!!!
;
; .EXIT 0
;
; ; Change the number of controllers for device XX to XXN, if XX is in STRING
;
; .ADDDEV:
; .TEST 'STRING'
; .SETN POS 1
;
; .LOOP:
; .IF XX EQ 'STRING'[POS:POS+1] .GOTO FOUND
;
; .SETN POS POS+4
; .IF POS GT <STRLEN> .RETURN
; .GOTO LOOP
;
; .FOUND:
; .SETS XXNS "'XXN'"
; .SETS 'STRING' 'STRING'[1:POS+2]+XXNS+'STRING'[POS+4:<STRLEN>]
; .RETURN

```




Remote Bridge Management

John Heffernan
Donna Ritter

Abstract

This paper describes the general network management problem and the models we have come up with to solve it. We then describe the general capabilities of network management. Next, we discuss the LAN Bridge 100, a store and forward repeater and its network management software RBMS. RBMS is a implementation of some of the concepts described in the first section.

1 THE NETWORK MANAGEMENT PROBLEM

As the LAN (Local Area Network) becomes a system bus and Wide Area Networks begin interconnecting LAN's to form large and complex distributed systems, the network becomes the system. The problem network management faces today is one of size, complexity, and diversity. As networks are becoming quite large, network management can no longer be thought of as an afterthought or as a part time job. Instead of multiple independent system managers managing their part of the network, we tend to see network managers who have overall network responsibility. Networks

today are complex because of the sheer number of devices and many ways to configure these devices. Networks today are diverse because of the wide variety of devices and communications hardware and software on the network. The architectural models for network management that we are working on today can form the basis for products that address the management of today's and tomorrow's networks in an integrated fashion. RBMS (Remote Bridge Management) is a product based on many of the principles described in the first half of this paper.

We see below the LAN multi-access medium as the LAN system bus.

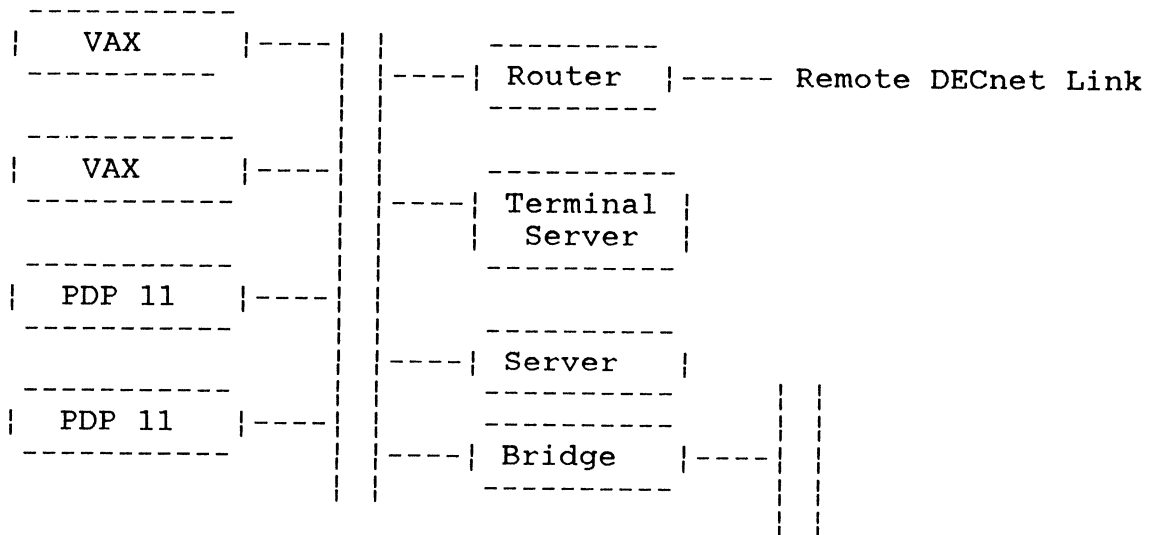


Figure 1

Ideally, the network manager logs into a host node dedicated to network management and can manage any network component such as bridges, terminal servers, and DECnet nodes. For example, the network manager should be able to do a "set line cost 10" operation for all bridges in the network with one command. Today, we are working on models that may help us construct integrated network management solutions. Some of the concepts of such a model are presented in sections one and two.

1.1 Definitions

Next, we present some definitions that we use later.

Entity- An individually addressable component in the network.

Agent- Software in the entity that carries out the remote management functions.

Attribute - A perceived property of an entity, an attribute has a value that can be examined and possibly modified.

Database- In the context of network management, we mean a management database. That is, a place where management attributes are stored.

Datagram Transport Service - Management messages are transferred via datagrams. There is one datagram per management message. The caller can not be sure that messages have been received by the other side. Also called connectionless service.

Director- Software that does the managing. This software is usually resident on the management station and has remote access to the agent via a protocol.

Directive - A directive is a management request, or action, defined upon an entity which may be applied to that entity.

Event sinks- A node where event messages are received and processed.

Multicast- On the Ethernet, this refers to sending a packet to a known group address so that the packet can be received by more than one node per transmission. Normally, packets are sent to only one node.

Protocol- The rules and regulations governing communication between the agent and the director.

Reboot- To cause the server to be started again. Generally, this means that the server is stopped and then operating software and characteristics are reloaded.

Service Element - That part of the entity that does the work that the entity is chartered to do (aside from management, which is done by the agent part of the element).

Server- General term for a product that sits on a LAN and serves multiple clients on the network.

Virtual Circuit Transport Service - Management messages may be transferred over a virtual circuit. This provided for guaranteed message delivery; the transport user can be assured that messages are either received by the other end or else the user is notified. Also called connection oriented service.

Bridge- A device that interconnects networks at the data link level.

LAN Bridge 100- An intelligent store and forward repeater that extends Local Area Networks.

RBMS- Remote Bridge Management Software- Software that manages the LAN Bridge 100. RBMS runs under the VAX/VMS operating system.

2 THE NETWORK MANAGEMENT MODEL

The basic model of network management is shown below.

On the director side, we see applications built on top of more basic management functions. The functions are the primitives needed to access the management data in the agent. The application provides the added value and user interface needed to make sense of and ease the management process. In the general case, the director provides configuration, fault, accounting, performance, and security management. Note that these functions are those prescribed by the ISO model and they are not necessarily implemented by Digital.

Software on both sides carries out the functions by building and parsing messages that convey the desired operation between the director and the agent. Messages are sent back and forth between director and agent. In a typical operation, the user types a

"set" command. The intention of the set command is to modify an attribute in the entity being managed. For example, the network manager may want to "SET LINE 1 COST 10" in a bridge. In this case, the application in the director, which does configuration management, calls a function that builds the appropriate message to be sent to the agent. The message is then sent over the network using an agreed upon management protocol. Depending on the entity being managed, a datagram or virtual circuit is used. The agent receives the request and validates it. It then carries out the operation and sends status back to the director. The director reads the status and reports any errors to the user.

In the typical operation above, the context of the command is a single bridge. However, in our architectural models we allow for the concept of user defined groups of entities which may be managed essentially at the same time. RBMS, for example, allows the context of a "set" command to be all the bridges "known" to RBMS.

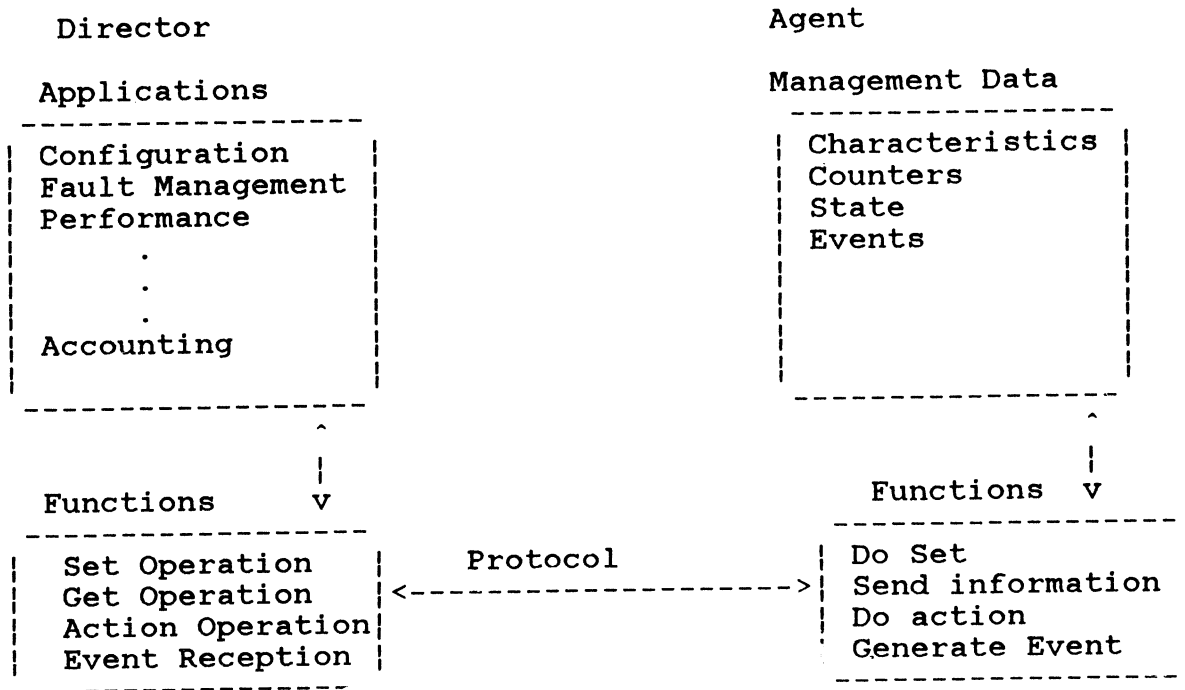


Figure 2

2.1 Attributes, Directives, And Events

Another way of looking at an entity is shown below.

We see that the agent and the service element "share" directives, attributes, and events. The service element is that part of the entity not dedicated to management (that is, does whatever work the entity is chartered to do).

Attributes are values associated with the entity. These may be characteristics, counter., or status. Characteristics are operational parameters used to control the operation of the entity. Examples are baud rate on a terminal server or maximum addresses on a DECnet node. Counters are those attributes that count events. Examples are number of lost packets for a DECnet node or framing errors in a data link. Finally, status are attributes that reflect the current state of the service element.

Directives are a command to the agent to do something. We can classify them as either examine, modify, or action directives. Examine directives return the value of counters, status, or characteristics. Modify directives modify the value of attributes. Finally, actions perform some management action such as rebooting a server.

Some common actions are listed below.

- o *Initialize entity*- Reset the attributes to those stored in a database for the server or the factory defaults and reboot the server.
- o *Create entity*- Create a new instance of an entity. For example, there may be a create entity directive to create a new service on a terminal server.
- o *Create link*- Create a communications link between two entities. For example, there may be a directive to create a logical link between two nodes.
- o *Delete link*- Destroy a communication link between two entities. For example, there may be a directive to break a logical link between two nodes.

Directives may change attributes of the entity.

Events are the occurrence of conditions within the entity that are of significance to management. For example, a server may generate an event when it reboots. Events are sent over the LAN and are received by event sinks and usually put in a log file at the management station.

Higher level software in the director calls the primitive functions in order to provide an easy to use interface that controls and monitors the agent. This is illustrated graphically below.

We see the higher level applications layered on top of the primitive functions. This is because the primitive functions can be shared by different higher level applications.

Configuration management includes the typical management in currently available products such as the maintenance of attributes and seeing what is on the network. Fault management addresses the diagnosis of failing network components and includes things like loopback testing and invoking diagnostics on the communications hardware. Performance management addresses how the network works under various loads. This might include monitoring counters or providing histograms of daily traffic.

Accounting addresses keeping track of who is using what. This might include generating usage reports so that network components can be charged back to the user.

Security management addresses keeping the network secure. This includes password maintenance and perhaps security event logging. Authentication also comes under security management. Authentication means verifying the identify of the person performing the requested management function.

2.2 Typical Operations

Some typical operations provided by network management are discussed next. The most common operation is to access attributes. This includes setting characteristics, reading counters, and reading status information. Network management software also includes a summary display which shows only the most important information about an entity. Counters are especially important when things go wrong. They can indicate which component is in trouble.

For fault management, the network manager may also choose to invoke diagnostics from the network management software. Most products allow at least a reboot of the product, which normally invokes self test on power up. The results of the self test may or may not be available to the network management software.

The network manager may also perform security management. He or she may set passwords on the communications product.

Another common network management operation is down line loading. The network management station

sends the operational software, diagnostic software, or operational parameters to the server when the server boots up. Usually, the server multicasts a load me" message to the network and the host responsible (it could be many) takes responsibility for down loading the server. Some products, like the LAN Bridge 100, have their operational software and characteristics stored on board since they must come up operational despite potential host problems. A similar operation is the upline dump of software and attributes when a server is in trouble. This allows support personnel to diagnose the problem with the server.

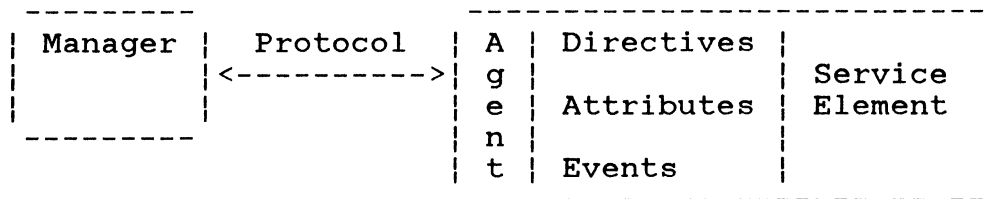
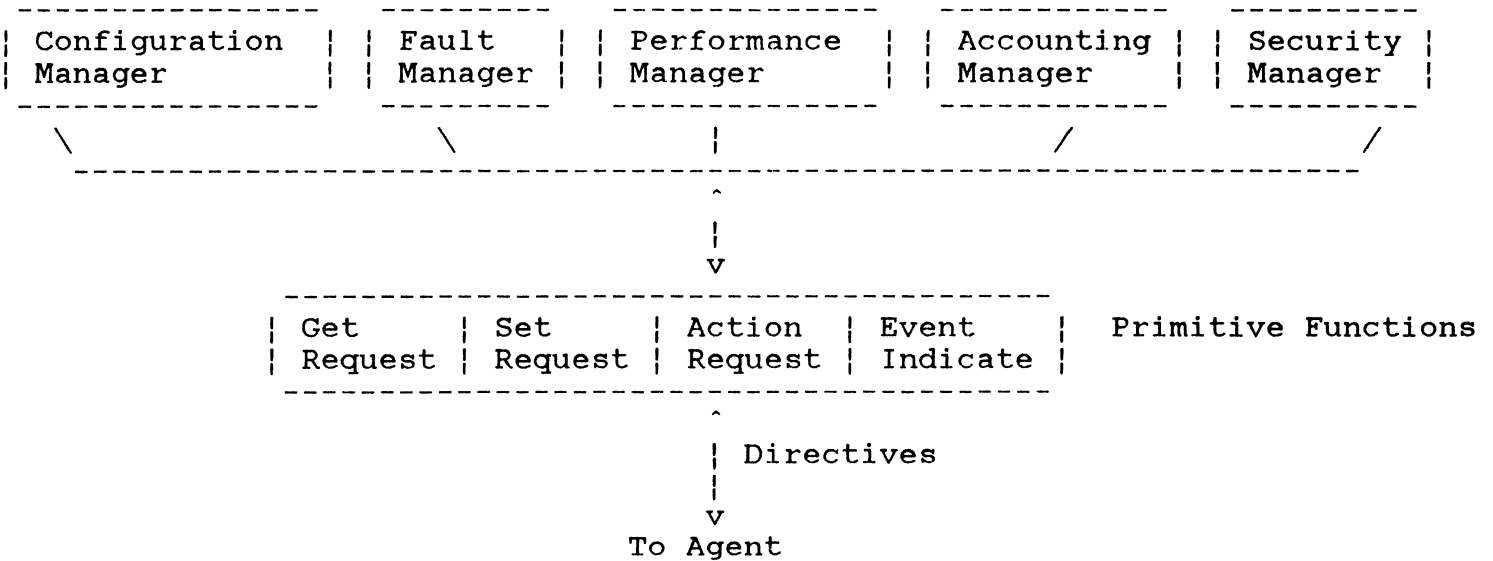


Figure 3



Layered Director Approach

Figure 4

2.3 System Management Protocols

There are many different system management protocols around now and we are moving towards standardizing on a single protocol (most likely, whatever becomes the ISO standard). MOP and NICE are two examples of protocols that have been used for DECnet. MOP is oriented towards low level datagram operations. NICE runs over a virtual circuit and is intended for use with a running node. RBMS uses a version of the IEEE 802.1 protocol over a datagram service. In the future, we hope to use a standard protocol and a standard transport mechanism.

In general, these protocols are all request/response protocols. That is, a request is made and the initiator of the request waits for the response. The typical format for a set request packet is shown below (based on IEEE 802.1).

The request ID is copied into the response message to allow the director to uniquely identify the message. The request ID field can also be used to hold information such as user identification, retry count, and sequence number in the case of a connectionless datagram transport mechanism.

2.4 Management Information Databases

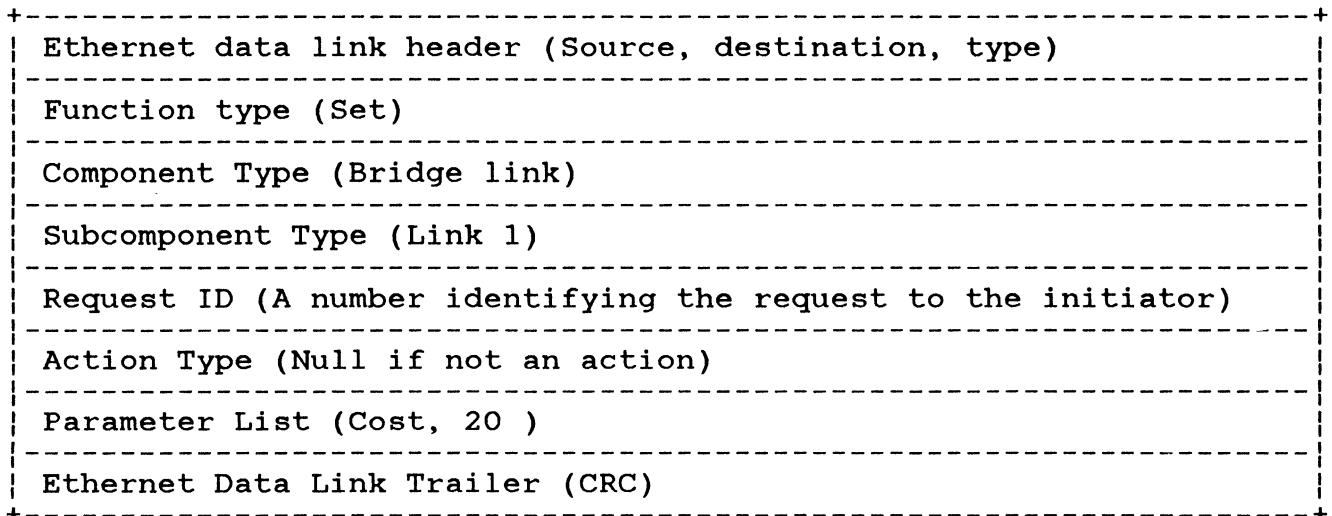
In general, the following databases may be present for any particular product.

Volatile- This is the local copy of operational parameters. Counters are always here along with the local copy of attributes. These are the ones used by the currently running service element software.

Local permanent- This database may or may not exist depending on the product. It may reside on local mass storage or in non-volatile RAM (NVRAM). The local permanent database is saved across a power down or reboot. The LAN 100 bridge has local permanent storage in the form of NVRAM. Usually, if local permanent storage is present, there is no need to have a host permanent database.

Host permanent- This is a copy of the attributes stored on the host management station. These are sent to the server when the server reboots. Usually, these are present if the product does not have local permanent storage.

Directory- This is a management station database that maps names to addresses and may contain other data only of interest to management software. Basically, this is a list of servers that can be managed. The management software may have a command to add all responding servers to this list; this builds the directory automatically.



Typical Ethernet Management Message

2.5 User Interfaces

The user interface is software that parses the user's command and displays the output. This is another area we hope to standardize and make consistent across all our network management products. The old style NCP style interface will probably give way to a more screen oriented display such as seen in the NMCC/DECnet Monitor. One new concept that can be seen in RBMS is the USE command. The USE command selects the entity or entities to be managed in subsequent commands. This becomes necessary as we move towards managing more than one server or more than one product by the same network management software.

2.6 Implementation Model

Next, we show a generic model of a single product network management package.

The display, command parser, and command executor make up the user interface. The user interface parses commands, forms messages if needed (some commands may not access the network), relays the message to the listener process, and then waits for the response from the listener process. It then decodes the status of its request and formats the result if need be.

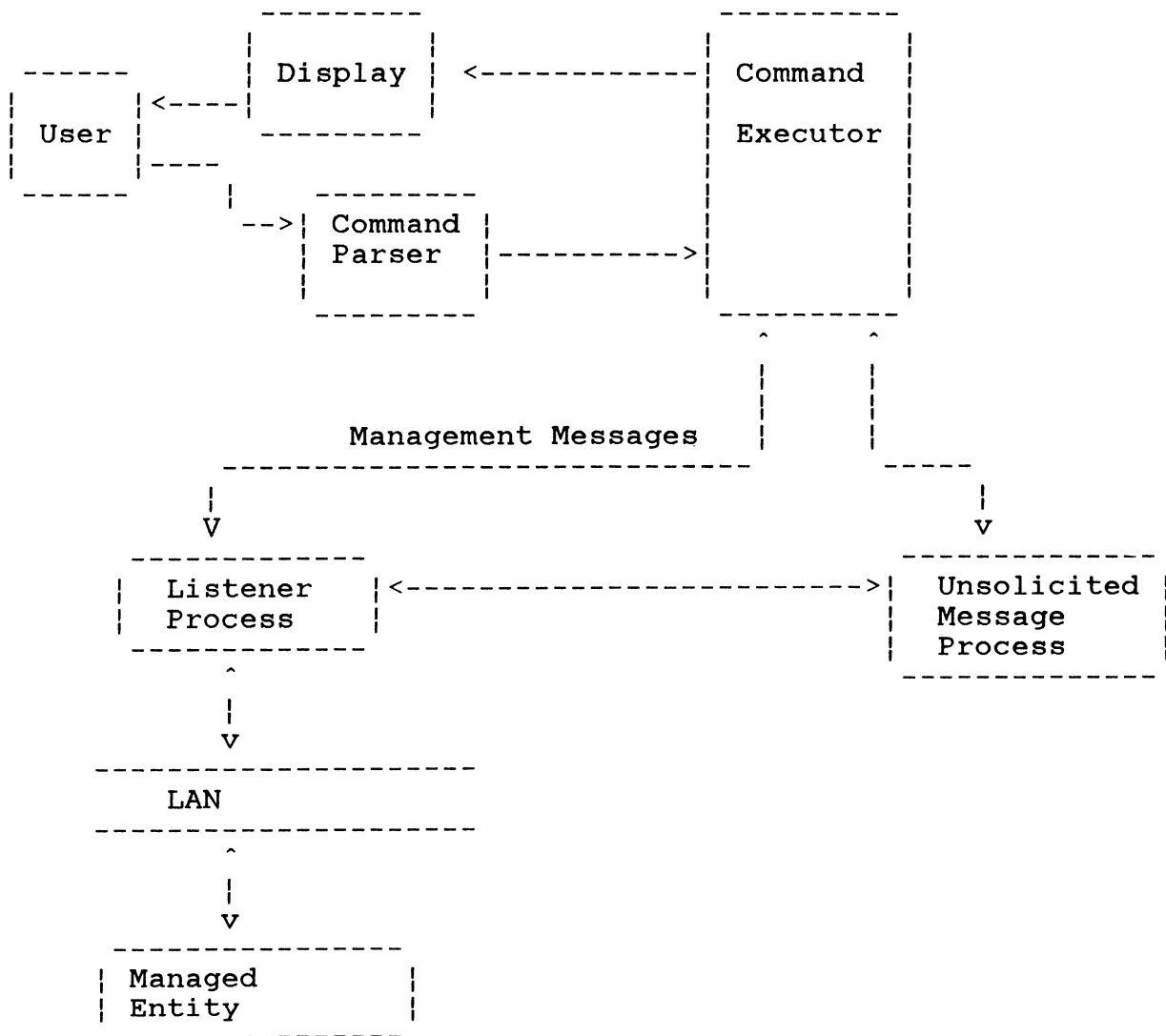


Figure 5

The listener process handles communications with the network and the the managed entities. It is responsible for demultiplexing any incoming traffic between the user interfaces (there may be more than one) and the unsolicited message process. If the product uses virtual circuits, it handles these. If a datagram service is used, the listener handles retries, timeouts, and sequencing.

The unsolicited message process handles any messages that arrive unsolicited. These could be events or requests for down loads.

Together, these three components show a general implementation model for a single network management product. Next, we look at RBMS, which is an actual implementation of some of the ideas we have presented. RBMS is a VMS layered product that will allow users at a VAX/VMS host to control and monitor any LAN Bridge 100 in an Extended LAN.

3 LAN BRIDGE 100 FEATURES

The bridging function performed by the LAN Bridge 100 operates at the data link layer, therefore it is an intelligent forwarding/filtering device which is protocol independent. It bridges two Ethernets (802.3 CSMA/CD LAN's) together. It uses the 802.1 standard request and response connectionless protocol for system management access. Bridges do not require RBMS, the management software running on VMS, to operate; however the Network manager gains a tremendous advantage with the use of RBMS. RBMS not only allows for the control of bridges, but also gives the Network Manager a tool to troubleshoot the network.

As Local Area Networks (LAN) grow they may exceed the physical limitations of signal propagation on the cable. The extension of the LAN using bridges solves this problem. This provides transparent data link connectivity. It also provides the ability to build redundancy in the network by placing bridges in parallel so that if one of them fails, the other will take over immediately. Finally, remote management capabilities provide the added feature of being able to control the filtering and forwarding of the traffic and to assist in isolating faults.

3.1 Dynamic Address Learning

Digital's LAN Bridge 100 receives all frames (from both Ethernets) since it operates in promiscuous mode. It dynamically learns where nodes are in the network by observing the source address of these frames. The bridge has a cache of table entries for these addresses that includes the Ethernet on which it was received and an "aging" timer. This information is used to forward and filter frames. Periodically, the table is checked for "aged" entries that have not been heard from; these are deleted from the table.

3.2 Forwarding Decision

When a packet is received, the bridge must decide whether it should be forwarded. To make this forwarding decision, the bridge looks up the destination address in its forwarding database.

Figure 6 shows an Extended LAN example where Station C is on the right side of Bridge_1. Assuming that a packet comes in Line 1 (Ethernet 1) of Bridge_1 with a destination address of C, the bridge will look up C in its forwarding database. If C is found, the bridge will have an entry that marks it as last being seen on its right side, so it would be forwarded on Line 2. If destination address B was received by Bridge_1, it would be discarded, because the packet is already on the correct side. If an address is not in the database, the bridge will flood the packet on all Ethernets except the Ethernet on which it was received. It will then learn this new address by putting it in the forwarding database.

3.3 Remote Management Module In The Bridge

The LAN Bridge 100 has a network management module (agent) that responds to management requests sent by an RBMS station. RBMS can be used to query the bridge for information, to control the bridge's state, to cause it to run self-test code and to set addresses in the forwarding database. This control over the forwarding database enables the Network Manager to filter traffic selectively on the Extended LAN.

3.4 Loop Detection

If bridges are to function in an environment free from management intervention, they must be able to break any loops that might occur in the topology either intentionally (for redundancy) or unintentionally. The LAN Bridge 100 uses a Loop Detection algorithm called the Spanning Tree Algorithm. (See reference 3 for a complete description of the Spanning Tree Algorithm.) The Spanning Tree Algorithm allows bridges in the network to automatically configure themselves into a loop free topology. This algorithm requires no management intervention, but can be tuned by Remote Bridge Management.

In very simple terms, this algorithm elects a Root Bridge by selecting the bridge with the lowest priority. The Root Bridge is the root of a loop free spanning tree. The Root's priority field is settable by management. Setting this priority field can force a choice of which bridge becomes the Root. In the case of a tie, the lowest physical address is selected.

Next, for each LAN, a Designated bridge is elected based on the lowest cost path to the Root. The Designated bridge is the bridge which forwards frames for the LAN segment. The other bridges on that LAN segment will be in a backup mode. The cost parameter is settable by management and can force the selection of a certain bridge to be the designated bridge. All bridges will turn off all of their lines, except for the single line that is the shortest path to the Root. The designated bridges will also turn on any lines attached to LANs for which they are designated.

Figure 7 is an example of some physical topology. It contains a backbone connected to 3 floors, each by a bridge. The third floor also contains a second LAN with 2 bridges in parallel.

For simplicity, we will assume that all bridge priorities and line costs are the same. These parameters could be changed by RBMS to force the topology to configure differently.

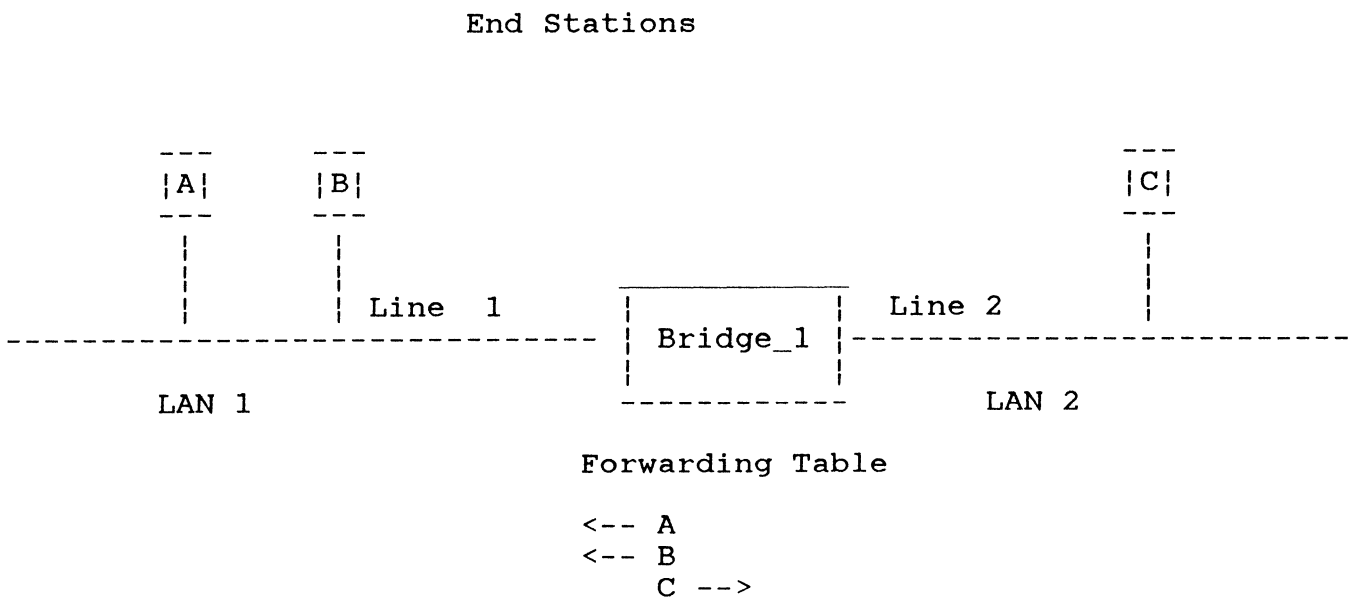


Figure 6

Network Physical Configuration

Mesh Topology

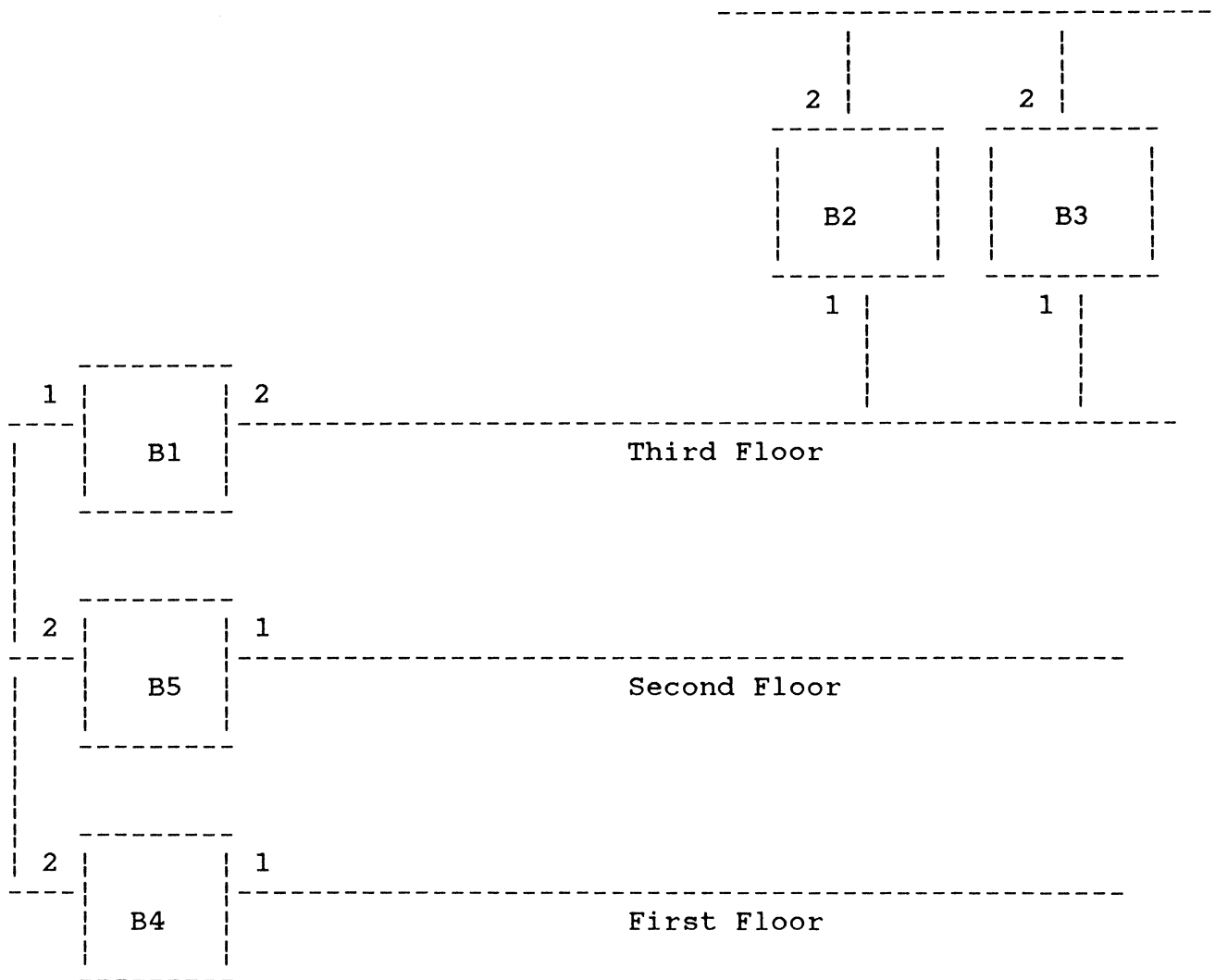


Figure 7

Figure 8 shows the results of the spanning tree algorithm. Bridge B1 with the lowest id (address/priority) is the Root. It is designated on both the Backbone LAN and LAN 3. Both of its lines are in the forwarding state.

A similar situation exists for Bridges B4 and B2. Bridge B3 is not a designated bridge; therefore it placed its Line 2 in the backup state to prevent loops. It left Line 1 in forwarding state since it is the least cost path to the Root.

Bridge B5 is designated on LAN 2. Line 2 is in the forwarding state because it is the least cost path to the Root. Line 1 is in forwarding state because Bridge B5 is designated on LAN 2.

Result of the Spanning Tree Algorithm

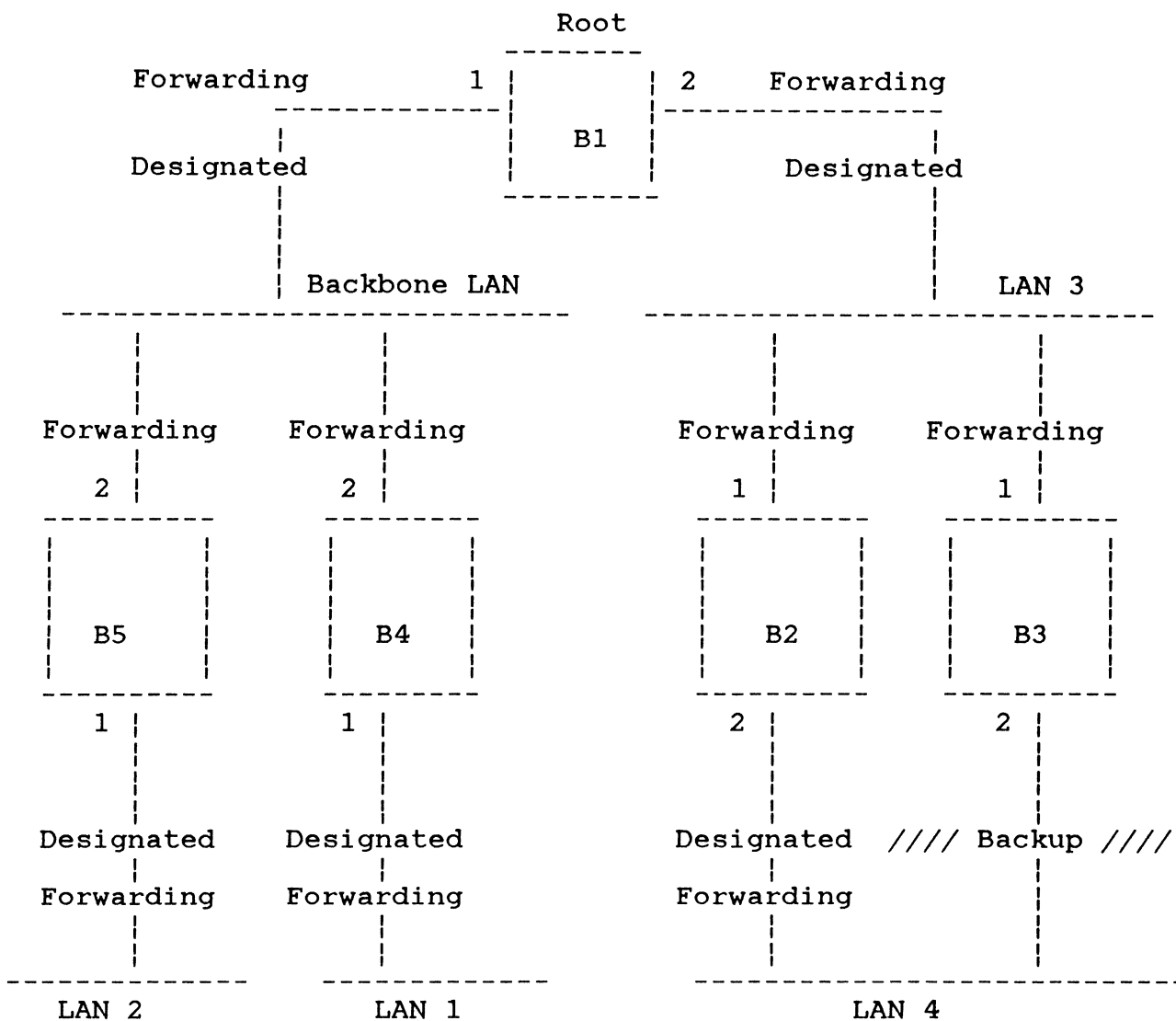


Figure 8

4 REMOTE BRIDGE MANAGEMENT SOFTWARE

RBMS allows the Network Manager to observe and control bridges. It runs on any VAX/VMS host with a DEC Ethernet controller. RBMS gives the Network Manager a tool to troubleshoot the network. This troubleshooting can help selectively filter traffic in the network as well as determine where potential faults have occurred. For example, if the Network Manager wanted to isolate a babbling transceiver to one network, the address could be placed in the bridge's forwarding database to be filtered.

4.1 Overview Of Remote Bridge Management Software (RBMS)

RBMS is made up of 2 processes. One is the Bridge Control Process (BCP) and the other is the Bridge Management Listener (BML). Multiple users can access RBMS on a VAX/VMS system. Each user will have a BCP process.

BCP is the user interface to RBMS. It provides the translation of a user's command to a bridge protocol message. It checks to see that the user has the privileges required to execute the requested command. It sends the protocol messages to BML for transmission. When it receives a reply from a bridge through BML, it parses the answer, and displays the resulting data on the user's screen.

BML handles all communication over the Ethernet. It is responsible for multiplexing and demultiplexing the requests from BCPs. It is also responsible for command retries and timeouts.

RBMS manages several databases. It keeps a host directory to associate ASCII names to bridge addresses. This feature enables the Network Manager to give meaningful names to the bridges rather than typing in their physical addresses. The directory also contains descriptive information about the bridges. RBMS allows access to the databases kept on the bridge. The bridge keeps a database with counter and status information. It keeps a cache for all of the forwarding entries it knows about. These entries include those dynamically learned and those set by RBMS.

4.2 Using RBMS In A Trouble Shooting Environment

RBMS allows the Network Manager to examine bridge or line status, characteristics and counters. It also allows examination of the forwarding database in the bridge. The forwarding database can be examined one address at a time or examined in categories (i.e. those addresses set by management, those addresses learned by the bridge, permanent entries or all entries). RBMS can be used to set certain bridge/line parameters, such as the spanning tree parameters, and to set addresses in the forwarding database.

Groups of bridges or lines can be addressed with a single command by using the KNOWN BRIDGES/KNOWN LINES command. If the Network Manager is interested only in those bridges that are in the OPERATE state, the ACTIVE BRIDGES option may be used. (The OPERATE state is the normal state of the bridge.) A USE command is also available that allows the Network Manager to establish a target for subsequent commands.

This command set gives the Network Manager a means to determine the condition of an Extended LAN and to selectively filter traffic. These capabilities allow RBMS to be used as a troubleshooting tool on an Extended LAN.

4.2.1 Example 1 -

In Figure 9, there are two LANs connected together by a bridge. The fault is that node ABC and node XYZ cannot communicate.

First the state of the bridge connecting the two LANs is examined. The "USE" command is executed to establish bridge Lan1_to_Lan2 as the current target for subsequent commands. Next a SHOW STATUS command is executed to determine if the bridge is in operate state (Figure 10).

Extended LAN Trouble Shooting Example #1

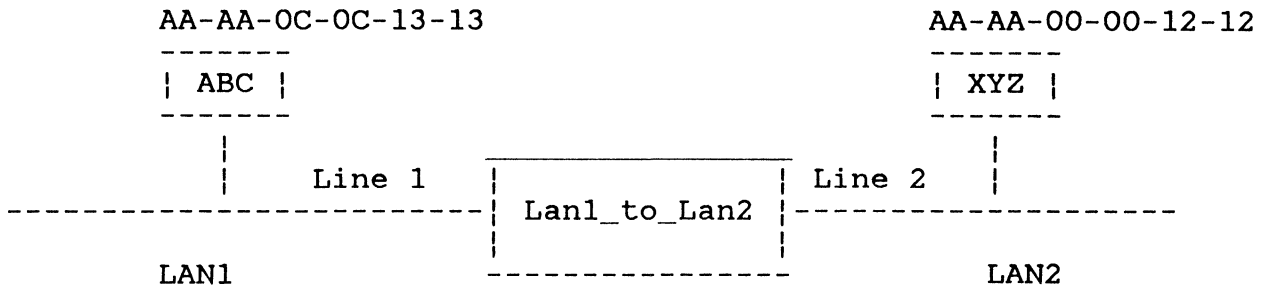


Figure 9

```
RBMS> USE BRIDGE LAN1_TO_LAN2

RBMS> SHOW BRIDGE LAN1_TO_LAN2 STATUS

Bridge Status as of 10-APR-1986 09:53:22
Bridge LAN1_TO_LAN2, Address 08-00-2B-02-99-55

Bridge state:
Forwarding entries - current volatile: 257
Forwarding entries - current non-volatile: 2
Management heard link: 1
```

Figure 10

The bridge is in the OPERATE state, so next the line states are examined. Given this topology, both lines should be in the FORWARDING state (see Figure 11).

The reason the two nodes cannot communicate is that the address is in the forwarding database as being filtered (destination is set to NONE). This can be rectified by issuing a management command to set the address to be forwarded on the appropriate line as is shown in Figure 13.

Since both lines are in the forwarding state, the address of the unreachable node is examined in the Bridge's forwarding database (see Figure 12).

```
RBMS> SHOW KNOWN LINES STATUS
```

```
Line characteristics for Line 1 as of 10-APR-1986 09:54:09
Bridge LAN1_TO_LAN2, Address 08-00-2B-02-99-55
```

```
Port id:          A
State:            FORWARDING
Line type:        Ethernet CSMA/CD
Remote management SETs: Enabled
Collision Presence: Disabled
```

```
Line characteristics for Line 2 as of 10-APR-1986 09:54:09
Bridge LAN1_TO_LAN2, Address 08-00-2B-02-99-55
```

```
Port id:          B
State:            FORWARDING
Line type:        Ethernet CSMA/CD
Remote management SETs: Enabled
Collision Presence: Disabled
```

Figure 11

```
RBMS> SHOW BRIDGE LAN1_TO_LAN2 PHYSICAL ADDRESS AA-AA-00-00-12-12
```

```
Forwarding Entry for Address AA-AA-00-00-12-12 as of 10-APR-1986
Bridge LAN1_TO_LAN2, Address 08-00-2B-02-99-55
```

```
Set by:           MANAGEMENT
Outbound Line:    Line NONE
Last seen on:     Line NONE
Destination:      NONE
Auto-delete:      NO
```

Figure 12

```
RBMS> SET BRIDGE LAN1_TO_LAN2 PHYSICAL ADDRESS AA-AA-00-00-12-12 LINE 2
```

Figure 13

4.2.2 Example 2 -

In Figure 14 there are 3 LANs interconnected by bridges. This example shows how to locate a node in the network given its address.

The address AA-00-04-00-1E-10 is the one that needs to be located. This is done by examining the forwarding databases of the bridges. Figure 15 shows the output of examining this entry in the database on Lan1_to_Lan2. The display shows that the bridge knows that the address is to be forwarded on Line 1.

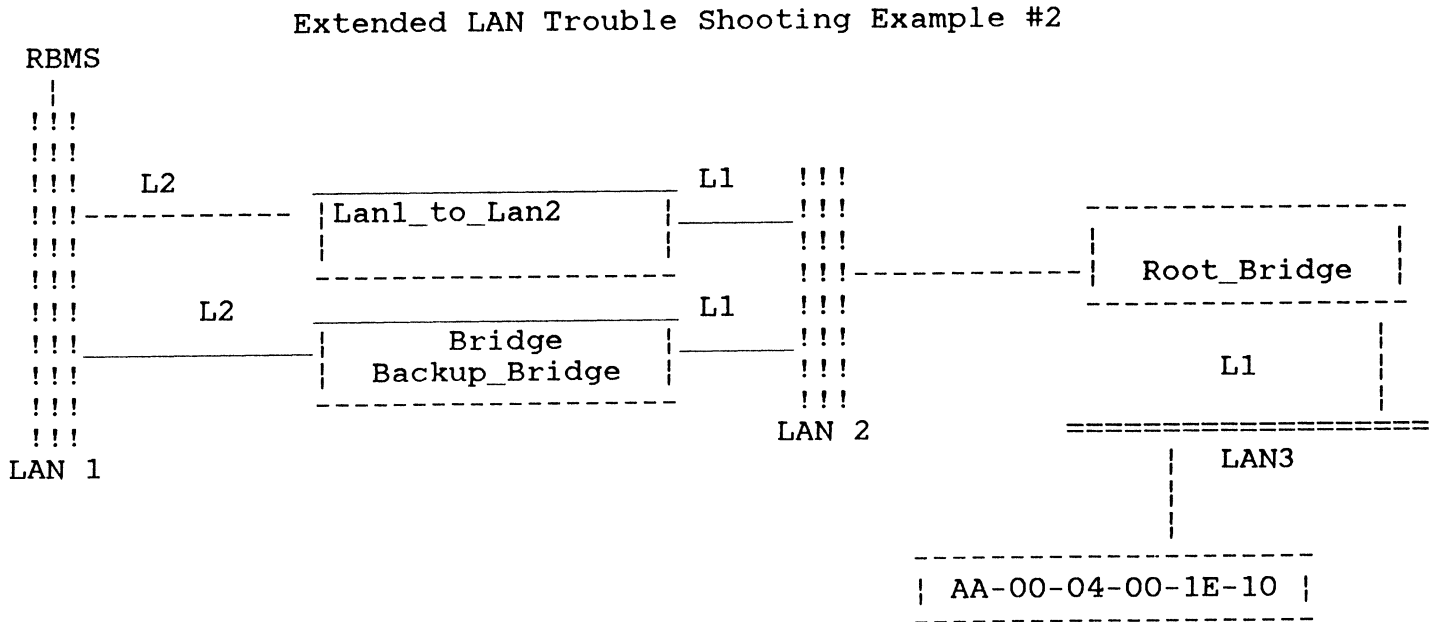


Figure 14

```
RBMS> SHOW BRIDGE LAN1_TO_LAN2 FORWARDING ADDRESS AA-00-04-00-1E-10
```

```
Forwarding Entry for Address AA-00-04-00-1E-10 as of 10-APR-1986
Bridge LAN1_TO_LAN2, Address 08-00-2B-02-99-55
```

```
Set by: LEARNING
Outbound Line: Line 1
Last seen on: Line N/A
Destination: NORMAL
Auto-delete: YES
```

Figure 15

Figure 16 shows the forwarding entry for the address to be located. The display indicates that the bridge ROOT_BRIDGE has learned that the address is on its Line 1. This is indicated by the field "OutBound Line". This narrows the location of the node to be somewhere on LAN3.

```
RBMS> SHOW BRIDGE ROOT_BRIDGE PHYSICAL ADDRESS AA-00-04-00-1E-10
```

```
Forwarding Entry for Address AA-00-04-00-1E-10 as of 10-APR-1986
Bridge  ROOT_BRIDGE,   Address  08-00-2B-02-0B-1A
```

```
Set by:                LEARNING
Outbound Line:         Line    1
Last seen on:          Line N/A
Destination:           NORMAL
Auto-delete:           YES
```

Figure 16

The field "Last Seen on" only applies to addresses that are set by management. If this were an address set by management, the "Last Seen on" field would indicate which side the bridge "saw" the address on; that is which side it would have learned it on. This is because when an address is set by management the "Outbound Line" is set, even though it may not be the line the address was last seen on.

5 SUMMARY

We have seen that current Networks require a distributed system of management. To facilitate this distributed system, Remote Server Management is needed to control the various network components on the Extended LAN. We discussed a general model and defined some of the desired features for Remote Server Management.

We discussed Remote Bridge Management Software (RBMS) as an example of Remote Server Management and noted that it not only gave us the ability to control and observe bridges; but also is an invaluable tool in trouble shooting an Extended LAN.

6 ACKNOWLEDGMENTS

Much of the general network management discussion is based on the work of Stan Goldfarb and Nancy LaPelle and various task forces they chaired. Thanks to Nancy LaPelle and Colin Strutt for a careful review of this paper.

7 REFERENCES

- 1) Alan Kirby, Tony Lauck, "An Architecture for Transparently Interconnecting IEEE 802 Local Area Networks", Presented at IEEE 802 Meeting, San Diego, CA, 1984.
- 2) Bill Hawe, George Varghese, "Extended Local Area Network Management Principles", Presented at IEEE 802 Meeting, San Diego, CA, 1984.
- 3) Radia Perlman, "An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN", *Ninth Data Communications Symposium*, September, 1985.
- 4) Nancy LaPelle, Ken Chapman, "Building Blocks For Remote LAN System Management", *FOC/LAN85 Proceedings*, Information Gatekeepers, Boston, MA.
- 5) Bill Hawe, Alan Kirby, Bob Stewart, "Transparent Interconnection of Local Area Networks With Bridges", *Journal Of Telecommunications Networks*, Summer 1984.

- 6) "DNA Ethernet Data Link Functional Specification, Version 1.0.0", Order No. AA-Y298A-TK.
- 7) "DNA Ethernet Node Product Architecture Specification, Version 1.0.0", Order No. AA-X440A-TK.
- 8) "DNA Maintenance Operations Functional Specification, Version 3.0.0", Order No. AA-X436A-TK.
- 9) "DNA Network Management Functional Specification, Version 4.0.0", AA-X437A-TK.
- 10) "The Ethernet - A Local Area Network - Data Link Layer and Physical Layer Specifications, Version 2.0, (Digital, Intel, and Xerox)", Order No. AA-K759B-TK.

Productivity Increases with the CORTEX Application Factory: Empirical Survey Results

Anthony C. Picardi
Cortex Corp.
128 Roberts Road
Waltham, Mass. 02154

Abstract

The purpose of this paper is to present empirical evidence of productivity increases resulting from use of Cortex's Application Factory. A high-level architectural overview of the Factory is given first, including a sketch of how the modules work together and where the Factory fits in the application development life cycle. A list of some recent Factory applications is given next for the purpose of acquainting the reader with the types of applications the Factory is used for.

The productivity estimates given in this paper were derived by comparing the actual time needed to produce applications with the Factory against production time using standard COBOL. Since none of the applications could be done twice, the COBOL efforts were estimated from the actual delivered functionality using function point estimating techniques developed by Albrecht in the late 1970's. The results of the comparison indicate an average productivity increase of a factor of 15.

Further survey results are given which serve to quantify the size and complexity of the applications in terms of function point inputs such as the number of screens and reports and the type of VAX target computer. Finally, some relationships between the size and complexity of the application and development efficiency are investigated using some of the metrics developed.

Beyond the average estimated productivity gain of 15, the survey showed that Factory applications were not concentrated in any one business sector or application. Average Factory applications included 45 screens 15 reports and 33 datasets. There is some indication from graphical data that large applications are less efficient in terms of required hours per function point, which is also true of the estimated COBOL effort for large projects. However, it seems that the relative advantage of the Factory relative to standard COBOL increases slightly as the application size and complexity increase. These conclusions are made tentatively, however, due to the small sample size and variance of the sample data.

1 APPLICATION FACTORY OVERVIEW

1.1 What Is The Application Factory?

The Cortex Application Factory evolved quite literally from the same roots as all other factories -- from the need to automate repetitious and tedious production processes. Cortex had long been involved in the development of business applications and had developed its own high-level language (Builder) for that purpose in the late 1970's. The fact that modern on-line multi-user business applications are composed of generic modules -- menus, screens, databases, reports, and procedural logic -- prompted Cortex developers to seek ways of capturing their expertise in building business applications to automate the process. If this could be achieved, then each consultant could be freed to concentrate on the creative part of the application development process, leaving the tedious coding to the computer.

This was the genesis of the Application Factory. The core of the Factory is the generator, which is capable of reading specifications about an application -- the application "meta data" -- and generating compilable code from them. The process of collecting specifications has also been automated by producing a Factory application for this task! Since specifications are nothing but data about generic parts of applications, it was possible to generate an application to collect and organize information about applications. Figure 1 shows a schematic of the Application Factory concept. The visible part of the Factory is the developer interface (a Factory application) which collects meta data information and stores it as a collection of RMS files. The input to the generator is this set of RMS files while the output is a set of programs which are the highly modularized components of an application.

Since it is so easy to create and change modules with the Factory, it is best used in a "prototyping" type of development process in which users are involved in the development process to test the various modules as they are developed. Experience has shown that much richer applications and more satisfied users result from this prototyping method rather than the more formalized specification method, especially when the applications are highly user-interactive.

1.2 What Is The Application Factory Used For?

The Application Factory is used to convert a business application data model and operational specification into an implementation on the full range of DEC VAX computers (and clustered computers) running the VMS operating system. An application is an integrated set of source modules and data definitions, not just a heap of screen and report programs.

Applications that have been developed with the Factory range from the "bread and butter" sales tracking, order entry, accounting, inventory and payroll applications to esoteric applications to track samples in quality control labs or to track raw materials through manufacturing, inspection, labelling, packing and shipping stages of automated manufacturing plants. Businesses using the Factory range from heavy manufacturing and energy production to travel agencies and health clubs. A number of consulting enterprises have sprung up to use the Factory to deliver customized business applications. Figure 2 shows a list of some of the representative applications included in the survey.

2 FUNCTION POINT SOFTWARE ESTIMATING TECHNIQUES

2.1 Major Inputs

The need for a new measure of software productivity beyond the traditional "lines of code" (LOC) evolved with the advent of higher level languages with which developers were producing fewer lines of code per unit time at a greater cost per line, yet still claiming productivity increases. The relevant question became: regardless of the number of lines of code, how do languages compare in terms of functionality delivered? Late in the 1970s Allen Albrecht of IBM began to develop a method of measuring an application's size and complexity in terms of function points. Function points measured the delivered functionality of a program instead of the developer's prior estimate of how many lines of code it would take (Albrecht, 1979). Since then his work has been extended to include numerous measures of complexity and environmental influences on

developer performance (Drummond, 1985; Zwanzig, 1984) as well as coefficients linking function points to many programming languages.

Function point estimating is done by first counting up the number of function-providing elements in each of five classes: inputs such as data screens; outputs such as reports; files; inquiries such as key screens or queries; and interfaces to other hardware or software systems. For each type of function, the number is weighted by a judgemental difficulty factor, and all the functional components are then added up to yield the "unadjusted function point total". This unadjusted total is then further modified up or down based on the importance of a set of "project influencing factors". These factors measure the importance of features such as communications, performance, hardware utilization or documentation and thus the extent to which developers will spend time devising elegant solutions to these problems. For example, the more critical performance is to an application, the more time will be spent on ways to deliver performance even though the total number of function points is not increased. The result of this activity is a total function point index which is independent of programming language and which embodies measures of both the size of the application, via extensive measures such as a number of screens and the complexity of the application, via intensive measures such as influencing factors pertaining to procedural complexity.

The heart of the function point technique is to relate the resulting function point index derived from the above activity to some empirical sample of software projects to establish a quantitative relation between function points and the expected time the project will take. Indeed much of the work since Albrecht's original paper has been to establish coefficients relating development effort in a particular language to total estimated function points. Even though the function points for a particular application may be fixed, technological advances in programmer development tools or languages can now be reflected in this coefficient. Thus a more efficient programming language will deliver more function points per unit time than an older language. This applies equally to non-procedural languages and application generators for which the old measures of lines of code are irrelevant.

2.2 Use To Estimate COBOL-Equivalent Effort

In order to be useful, a measure of productivity increase must be relative to some familiar standard. For this reason, the estimates of productivity increases in this paper are based upon the effort it would take to produce the same application functionality using the Application Factory relative to COBOL. The empirical coefficient relating COBOL to total function points was obtained from a 1983 GUIDE publication. The coefficient relates the total function point measure for an application to the total time it would take to produce the application entirely coded in COBOL, with the level of technology (editors, report printers and generators) that was typical at IBM in the early 1980s. Since the most common language for business applications has traditionally been COBOL, this was thought to be the least common denominator for the Factory comparison.

3 THE APPLICATION FACTORY SURVEY

3.1 The Sample

The survey sample consisted of applications which have been completed by Cortex's Application Factory customers and internal Cortex Factory users in the period between late summer of 1985 and early spring of 1986. About half of the data was collected via a four-page written questionnaire and half via a telephone survey. The target population was censused rather than sampled since the numbers were small enough to allow complete inclusion in this time frame. Even so, probably a full 20 percent of the applications were not included due to non-reporting or omission from the census list. It is not felt that this introduced any significant bias in the results, however, since the omissions were not concentrated in a particular sector or application type. In all the sample consisted of 29 applications.

3.2 Descriptive Results

The following figures describe the quantitative aspects of the applications that were used as inputs to the function point estimate. Figure 3 shows the distribution of the number of data screens in the applications among the sample. The average number of screens was 45 with a definite trend toward more data screens in the later telephone survey sample.

Figure 4 shows the distribution of the number of reports among the survey sample, with the average being 15. This is a clear underestimate of the number of reports in a "typical" application since a number of the sites reported that they had delivered the first application with only a fraction of the required reports due to aggressive delivery schedules. Nevertheless, the function points were calculated on the basis of the actually delivered number of reports.

Figure 5 shows the distribution of the number of datasets included in the survey sample. Datasets are logical file data structures and are implemented in the Factory as RMS files. The average number of datasets was 33, again with the trend being to larger applications over the time span of the survey.

The survey included fourteen "influencing factors" which could potentially add complexity to the delivered application. Among the factors which were found to be important were: communications between the application and sources of data via VAXMAIL or other means over DECNET in 14 percent of the cases; distributed processing via CPU sharing of data files in 10 percent of the cases; performance-related FDL tuning and other modifications in 66 percent of the cases; customizations related to a high number of concurrent users or transaction volume relative to the target CPU in 48 percent of the cases; attention paid to application generality for multiple installations in 24 percent of the cases; customizations needed to add application features not obtainable via non-procedural Factory specification in 38 percent of the cases; and hooks added to the application for the purpose of easing future modification in 59 percent of the cases.

There are a number of adjustment factors that normally add function points to an application which the Factory provides by default with no extra effort on the part of the developer. These features

include: on-line entry of data; complex multi-screen functions; on-line update of the database; full documentation of the application meta data; and utilities which automate database updating for data structure changes and moving the application to various target operating machines.

The resulting distribution of total function points for the sample applications is shown in Figure 6 with the average being 657 total function points per application. This average number of function points corresponds to a generic application with an average of 45 screens, 15 reports and 33 datasets with an upward adjustment factor of 7 percent due to performance and logical complexities of the type mentioned above. This average number of function points corresponds to an equivalent of 68,960 lines of COBOL code.

Information on the relation between function points and the CPU size that the applications were targeted for was only available for about half the sample and is most meaningfully given in terms of the ranges below. The surprising lack of correlation between the size of machine and the function points in the application results from the facts that machines are shared among other applications most of the time and the wide variety of configurations achievable with any one basic CPU via additions of memory or disk drives.

CPU	Function Point
Microvax	237 to 1325
750	111 to 1437
780	458 to 1640
785	291 to 1195
8600	763 to 2418
8650	594

3.3 Productivity Ratio Of COBOL To Factory Effort

Function points were used to estimate the person-weeks of effort that each of the applications in the survey would take if coded from scratch in COBOL. This estimate was then divided by the actual person-weeks that the application took to develop.

This ratio is termed the productivity increase of the Factory relative to COBOL and was found to average 15 over the entire survey sample. The distribution of productivity increases is shown in Figure 7. The smallest productivity increase was found to be 3.4 and the largest was found to be 32. A ratio of 32, for example, can be interpreted as meaning that the programmer on this particular application was 3100 percent more efficient using the Factory than an average COBOL programmer would be expected to be using COBOL!

3.4 Relationships Among Size, Efficiency And Productivity

Having data on both function points and actual resources, it is appropriate to ask whether the larger applications take longer than the shorter ones. Figure 8 shows an exploratory plot relating the delivered function points to actual person-week resources. One would expect the effort to increase as functionality increases, which is the case in general, however, there does seem to be the slight "decreasing returns to scale" phenomenon common in all production systems. This behavior means that you get proportionately fewer and fewer function points per person-week expended as the project increases in size (function points).

The decreasing returns to scale relation between effort and application size can be seen more clearly in Figure 9 in which actual hours per function point is related to total function points in the applications. This scatter plot suggests, again, that function points are harder to come by as the application size and complexity increases.

But decreasing returns to scale for production factors is nothing new -- indeed the relative efficiency of small-developer teams is well-known in the software industry. The important question is whether the Factory performs better on large projects than COBOL could be expected to perform. Figure 10 shows the relation between the productivity increase ratio (shown in Figure 7) and the size of the application as measured in function points. The scatter plot indicates a slight trend to increased advantage relative to COBOL as application size and

complexity increases. These last inferences drawn from Figures 8, 9 and 10 are made only tentatively since the small sample and its large variance do not admit to strong statistical inferences.

4 CONCLUSIONS

This survey of applications produced with the Cortex application Factory has shown them to be diverse, covering a wide range of industries and application types. The average application consisted of 45 screens, 15 reports and 33 datasets and was produced 15 times more efficiently than it could have been expected to have been produced using COBOL (in a 1980s IBM COBOL coding environment). Although the small sample and its variance do not admit to quantitative inferences, there is an indication that Factory-produced development efforts may not be as subject to the large project decreasing returns to scale as could be expected using COBOL. This may be due in part to the Factory's application STATUS and developer guidance system, and automatic meta data documentation which significantly reduces the confusion and interference among multi-developer teams.

Future investigation of Factory productivity will endeavor to establish the nature of the learning curve for developers as they accumulate experience using the Factory in successive applications.

5 REFERENCES

1. Albrecht, Allan J., "Measuring Application Development Productivity", *Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium*, October, 1979, Pages 83-92.
2. Drummond, Steve, "Measuring Applications Development Performance", *DATAMATION*, February, 1984.
3. Zwanzig, Ken, ed. *Handbook for estimating Using Function Points*, for GUIDE Project DP-1234, November, 1984.

Figure 1: The Application Factory Concept

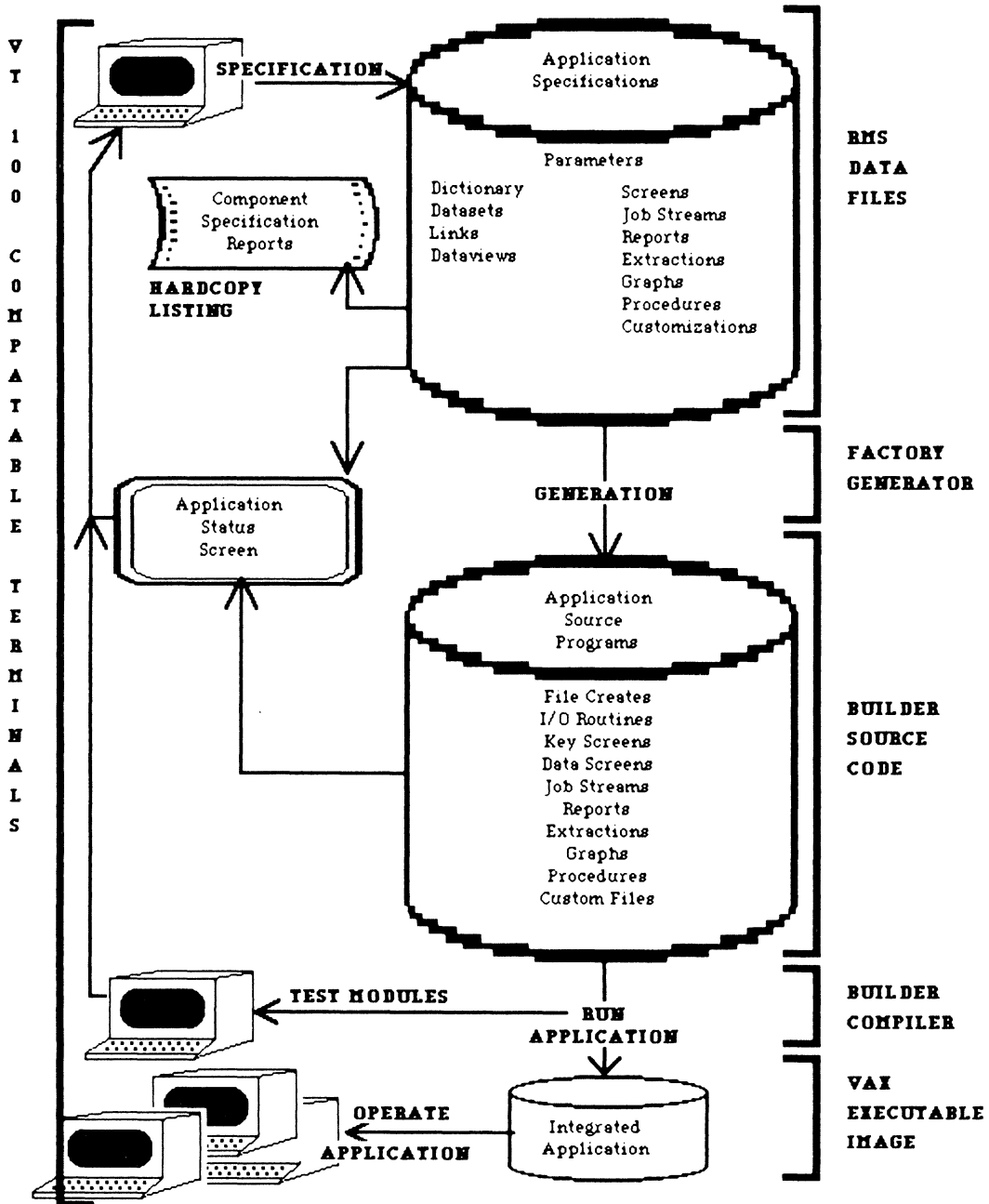


Figure 2: Representative List of Factory Applications

Job quotation and interface to order entry system

International purchase order tracking

Consultant job and time tracking and reporting

Software maintenance contract tracking and renewal letter production

Integrated inventory, accounts receivable/payable and payroll for wholesaler in Bermuda

Hotline call tracking and bug reporting

Application Factory user interface

Medical office management, patient registration, scheduling, billing

Manufacturing material tracking, weighing, packing, labelling, storage, history

Inventory and reorder for municipal utility water Department

Order entry, inventory and production scheduling for a variety of industries

Student registration, student and teacher scheduling, and reporting

Sample testing and quality control for beverages and pharmaceuticals

Field service scheduling for software hardware/software customers

Travel club membership, mailing list, trip discount tracking

Commodities daily price tracking, analysis and reporting

Document tracking for large engineering firms

Customized payroll packages for universities or foreign countries

Health club membership, checkin, billing and dietary analysis

Figure 3: Number of Data Screens in an Application

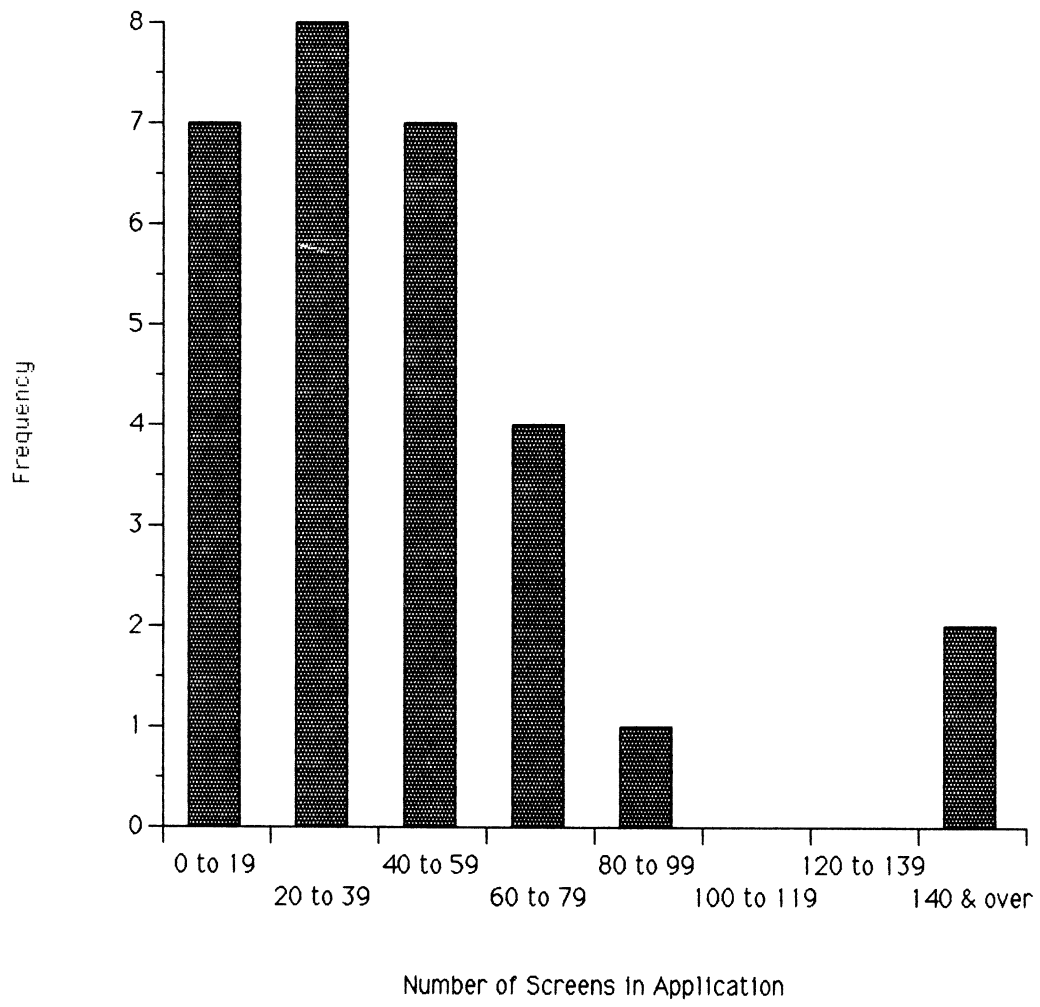


Figure 4: Number of Reports in an Application

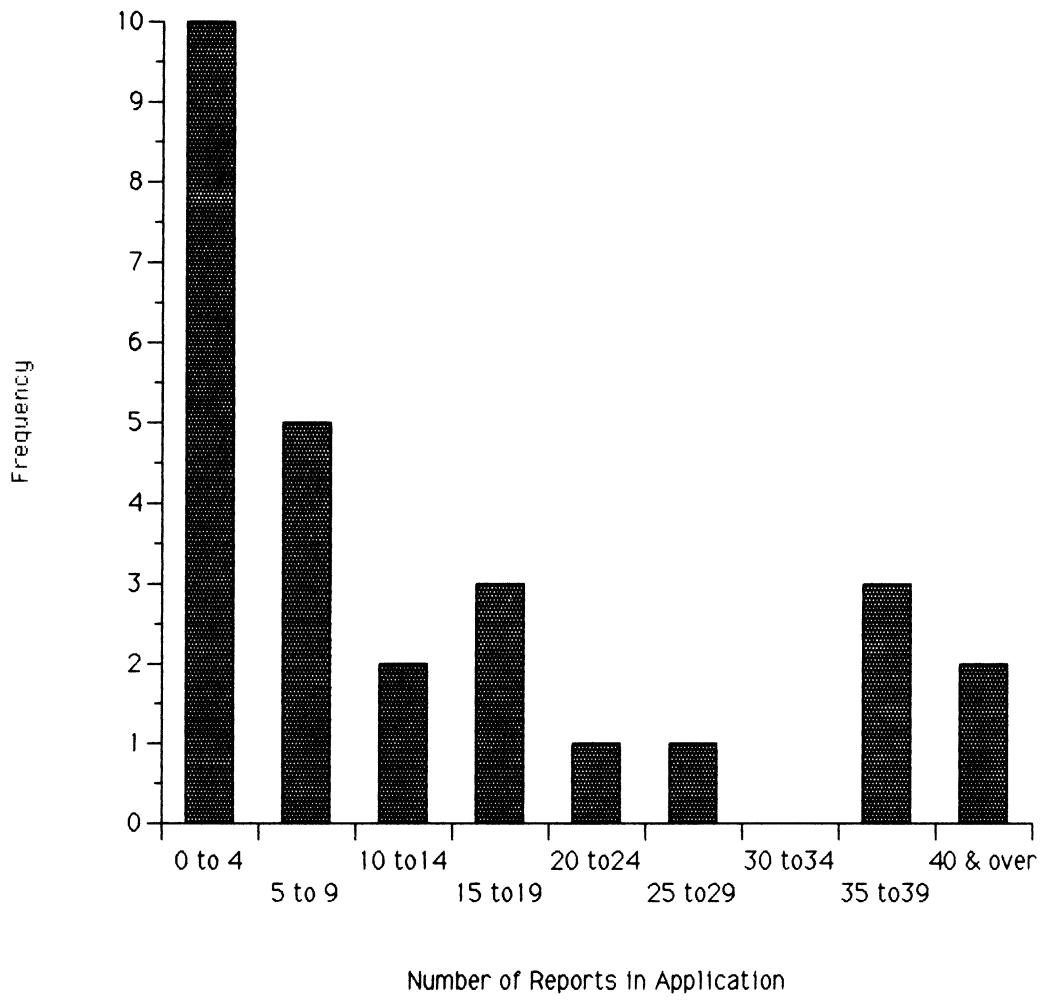


Figure 5: Number of Datasets in an Application

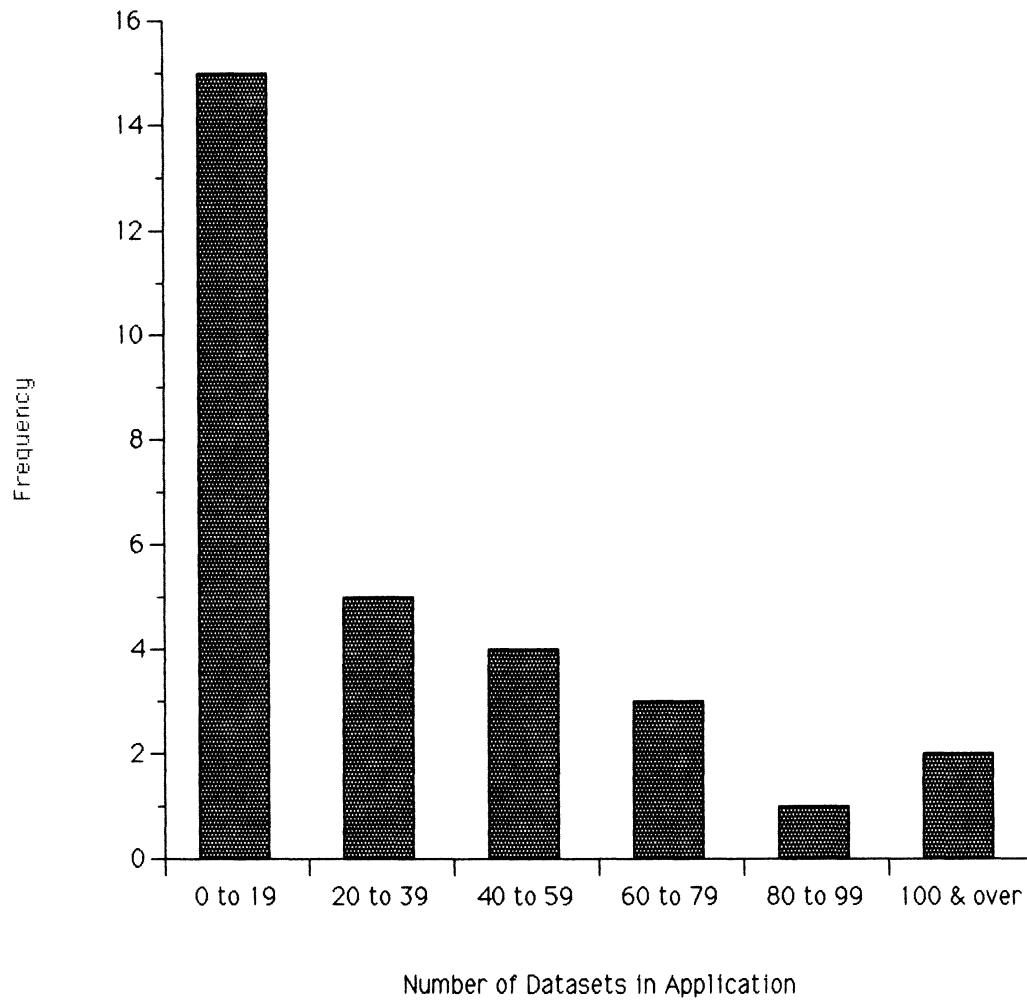


Figure 6: Function Point Measure of Application Size and Complexity

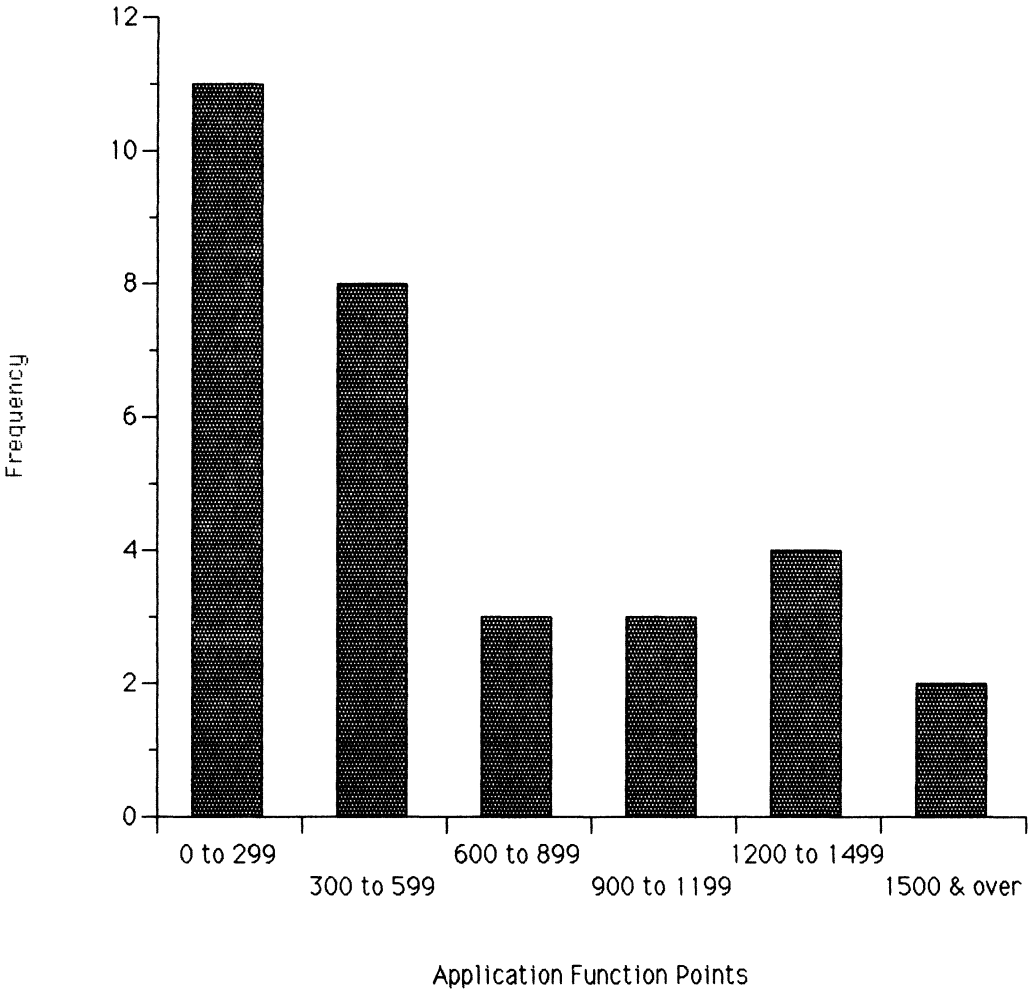


Figure 7: Ratio of Estimated COBOL Effort to Actual Effort Using the Application Factory

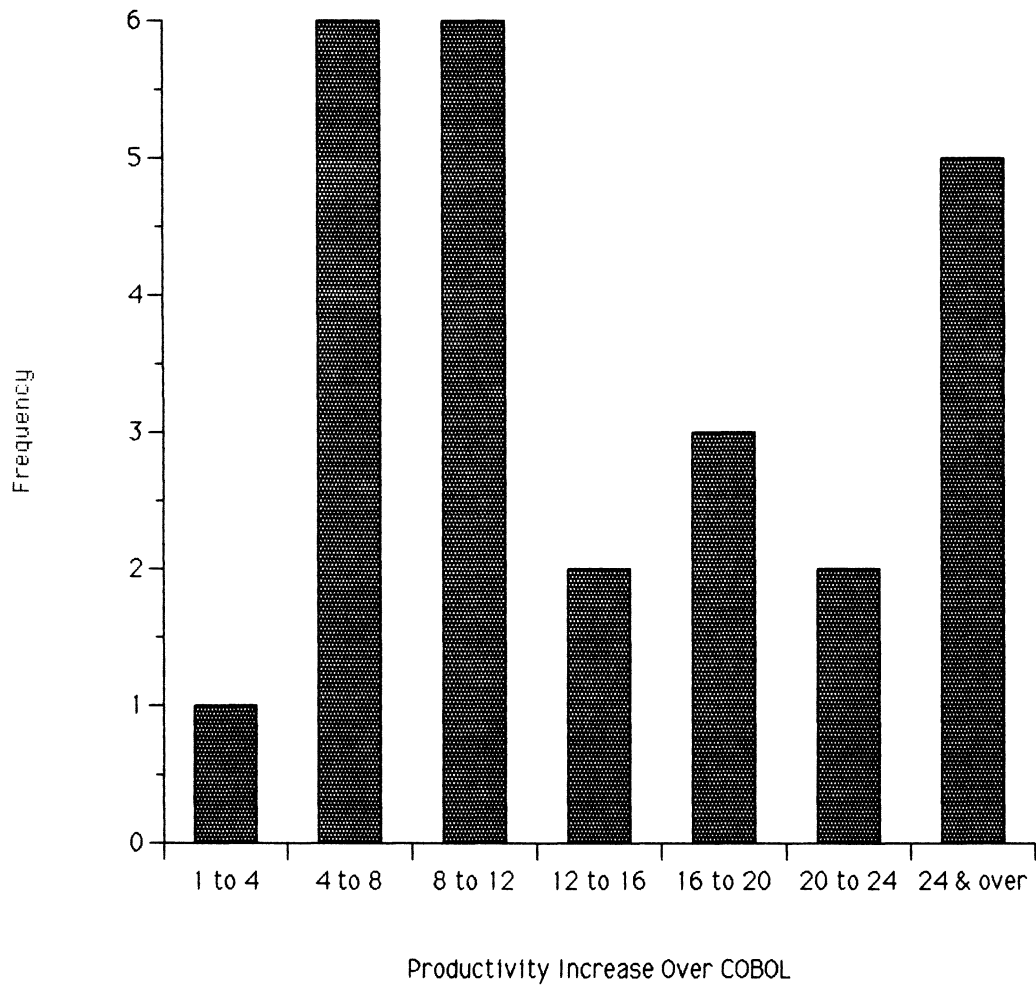


Figure 8: Relationship Between Application Size and Complexity and Development Resources

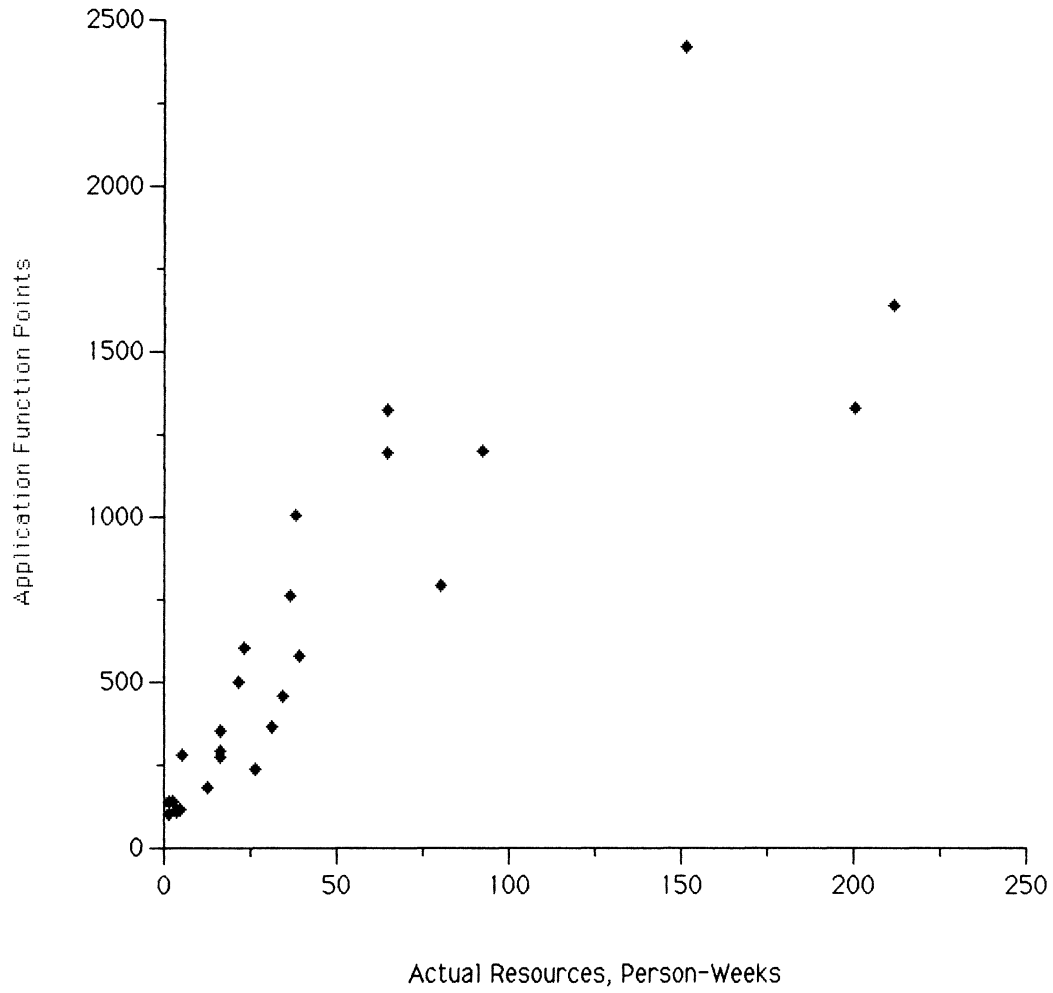


Figure 9: Relationship Between Application Size and Complexity and Development Efficiency

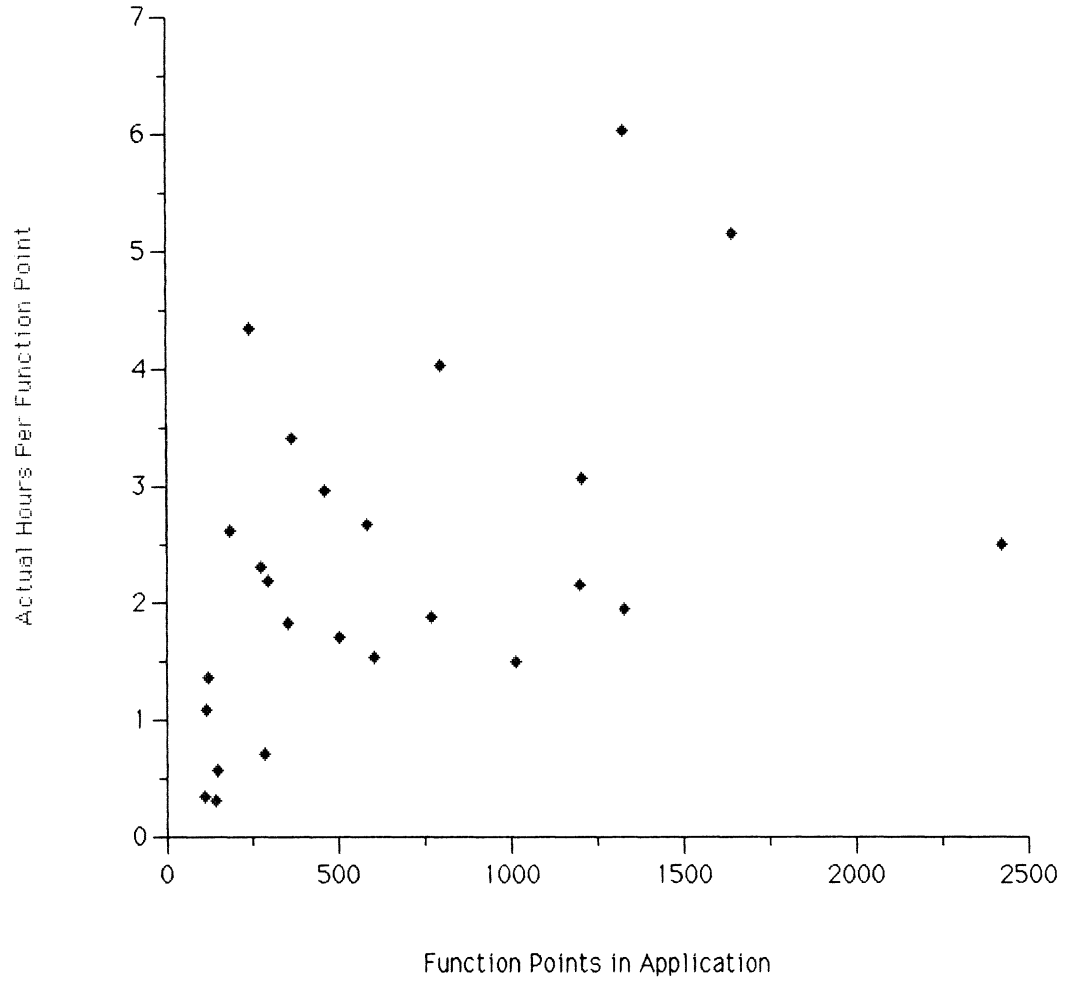
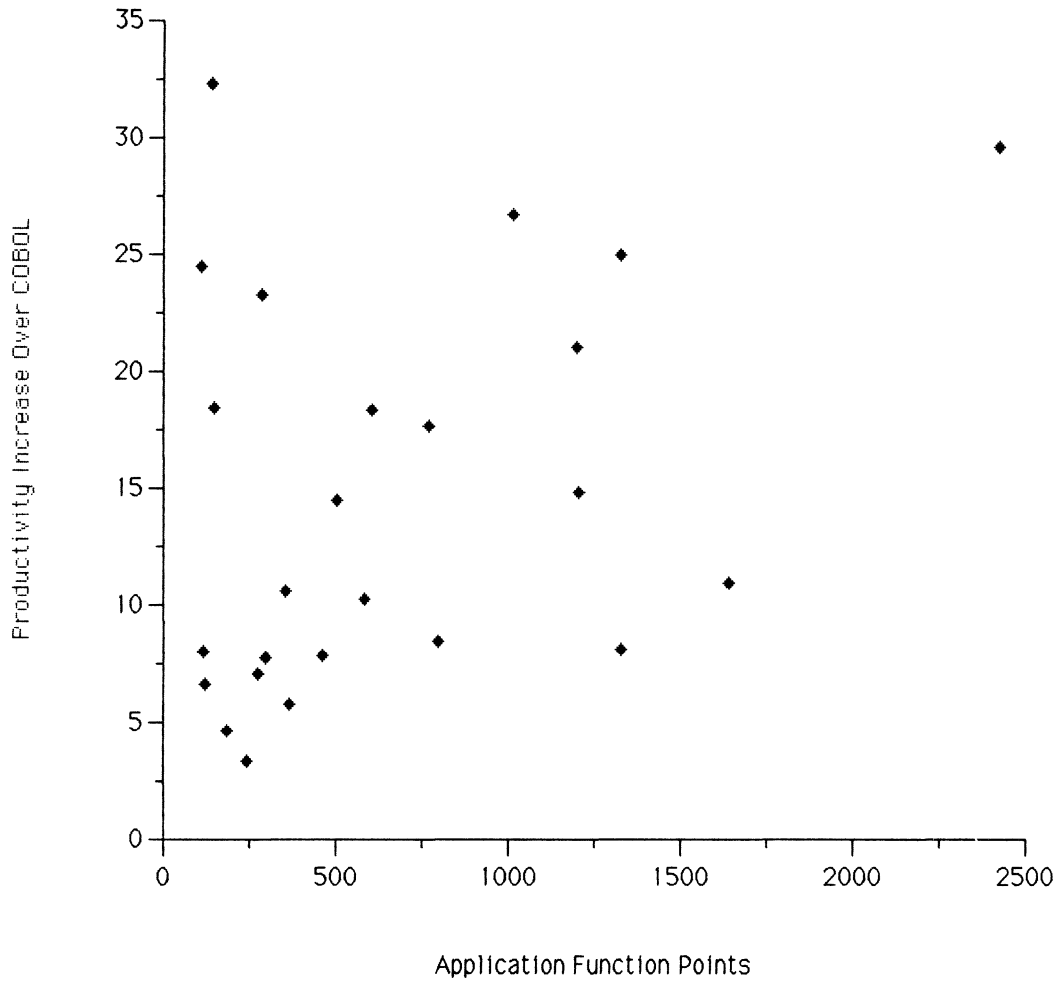


Figure 10: Relationship Between Application Size and Productivity Increase Relative to COBOL





Rainbow Color/Graphics Option Use in an Assembler Language Programming Course

Robert Workman
Southern Connecticut State University
New Haven, Connecticut

ABSTRACT

The Rainbow Color/Graphics Option is used in an assembler language and computer architecture course to demonstrate macro assembler programming and coprocessor control and architecture. Topics covered are graphics option software level architecture, interface with the 8088, display of graphics data, and the use of Macro Assembler Language to implement a graphics standard.

The study of The Rainbow Color/Graphics Option is an excellent supplementary topic for use in 8088 Assembler Language and Architecture courses. The Rainbow Color/Graphics Options is based on a NEC uPD7220 Graphics Display Controller (GDC) and a 64K byte video memory. The Rainbow Assembler Language programmer can control the GDC by use of the Rainbow's 8088 microprocessor's I/O ports. The details of how the Rainbow interacts with the GDC is documented in the "Rainbow Color/Graphics Option Programmer's Reference Guide"[1]. This manual provides a clear description of the GDC architecture and how the Rainbow interacts with it. A serious shortcoming of the manual is that there are no demonstration programs. This paper will comment on the advantages of including GDC programming in an Assembler Language course and present a sample program that uses procedures presented in "The Color/Graphics Option Programmer's Reference Guide".

Assembler Language texts generally include sections on the software level architecture of a microprocessor; machine language coding; the use of software development tools such as an editor and a debugger; the use of an assembler such as Microsoft's MASM; the microprocessor's instruction set; and several short application examples that illustrate

topics such as sorting and searching[2]. It is unusual to find examples of programs that take advantage of procedures written by others. This omission can leave students with the false impression that "real" programming consists of starting a project from the beginning and writing all the code that is needed to complete it. Including an exercise that uses the Rainbow's GDC has a number of advantages. Students get to create a small part of a large system. They learn how to set up a large assembler program. They get to work with professionally written and documented code. Finally they are able to work on an interesting problem that is fairly easy to solve with assembler language and that has many interesting expansion possibilities.

User programming of the GDC is possible only because of the excellent documentation and demonstration assembler code provided in the "Color/Graphics Option Programmer's Reference Guide". The guide includes material on the Graphics Option's operating principles, programming guidelines with over thirty demonstration assembler language procedures, and a reference section that documents the Graphics Option's instructions and software level architecture. The demonstration procedures were developed by Digital to test the Rainbow's Graphics Option. Procedures documented included routines that

initialize the GDC, set up the GDC registers, and do area, vector, and text write operations. Read operations and vertical scrolling are also demonstrated. The assembler language source code is an excellent model of what professionally written assembler code should look like. An example of this code is shown in Figure 1.

An additional feature of using the GDC assembler language procedures is that this source code has been widely available on Rainbow Fido network bulletin boards making this code available in machine readable form. As previously pointed out a problem exists in that the "Color/Graphics Option Programmer's Reference Guide" does not include any working examples. The user is left with the task of setting up their own Assembler Language EXE file. The program in Figure 2 shows how to set up an EXE file with separate Code, Data, and Stack Segments. Also shown in the program is a sample programming exercise and the commands that must be given to the LINK program. The object code that is linked with the exception of TEXT, the main program, is assembled code from the "Rainbow Color/Graphics Option Programmer's Reference Guide". The demonstration

program uses several GDC procedures, these procedures are listed in extrn statements. The program includes a demonstration macro that displays a graphics character in a user selected row and column on the graphics screen. The GDC procedure used to select and place the text is named GTEXT. As commented machine readable source code is available it is relatively easy to change the characters. A small part of GTEXT where characters are defined is shown in Figure 3. It is also possible to modify GTEXT to change the size of the characters.

```

;*****
;
;   p r o c e d u r e   o p t i o n   p r e s e n t   t e s t
;
;   purpose:          test if Graphics Option is present.
;   entry:            none.
;   exit:             dl = 1           option present.
;                   dl = 0           option not present.
;
;   register usage:  ax,dx
;*****
cseg   segment byte   public 'codesg'
       public option_present_test
       assume cs:cseg,ds:nothing,es:nothing,ss:nothing
option_present_test   proc   near
       mov     dl,1           ;set dl for option present
       in     al,8           ;input from port 8
       test   al,04h        ;test bit 2 to see if option present
       jz    opt1           ;if option is present, exit
       xor    dl,dl         ;else, set dl for option not present
opt1:   ret
option_present_test   endp
cseg   ends
       end

```

Figure 1. A sample procedure from the Color/Graphics Option Programmer's Reference Guide, page 5-1.

```

;This program demonstrates the use of the Rainbow Graphics option
;text write option.  It shows:
; EXE file set up with separate code, data, and stack segments.
; The use of several external procedures and public data.
; A macro definition and use.
;     Use of the GTEXT procedure from the GDO reference manual.
; User defined text location and character selection
; The use of the FGBG register to create light on dark text.
;Programming Assignment 12:
; a.  Accept characters from the keyboard
; b.  Display the character.
; c.  Accept additional characters adjust the cursor
;     and display the new characters.
; d.  Allow the user to exit this routine.
; e.  Print what has been entered using JOBSDUMP.
; f.  Modify procedure gtext display 6 nonstandard letters
;
; See extern statements for procedures used.
; Link parameters are:
;     link text+ex03+ex04+ex05+ex08+ex09+ex10+ex16+ex19+ex20+ex21+ex24;
;     Text is the object file from this program.  All other object files
;     are from the "Color/Graphics Option Programmer's Reference Guide".
;
;*****
; m a c r o d i s p l a y _ t e x t                                     *
;                                                                           *
; purpose: display 8 by 10 graphics text                                  *
; entry:  row is the column location of the character                    *
;         column is the column location of the character                *
;         character is the ascii code for the character                 *
; notes:  This macro makes use of procedure gtext                       *
;         which is defined in the C/GO Reference Guide.                  *
;*****
;

```

```

DISPLAY_TEXT MACRO ROW,COLUMN,CHARACTER
    mov bx,row                ;row number
    mov ax,column            ;column number
    mov dl,character         ;character number
    mov dh,0f0h              ;set fgbg register for light on dark
    call gtext
ENDM      DISPLAY_TEXT
;***** Set up Code Segment *****
cseg      segment byte public 'codesg'
    extrn gtext:near        ;text write
        extrn  init_option:near,graphics_on:near,graphics_off:near
    extrn gdc_not_busy:near,imode:near
    extrn set_rectangle:near,pixel:near,pattern_register:near
    extrn pattern_mult:near
        assume cs:cseg,ds:dseg,ss:sseg,es:dseg
        org      100h
        mov      ax,dseg                ; set up data segment
        mov      ds,ax
        mov      es,ax                ; set up extra segment
        mov      dx,2                ;select high resolution
        call     init_option          ;initialize graphics option
        call     graphics_on          ; turn on the graphics
; Display characters 80 through 0 in columns 80 through 0.
    mov cx,80
text:
    push cx
    DISPLAY_TEXT 1,CX,CL
    pop cx
    loop text
    display_text 10,15,80 ;use display_text macro with constants
; Character 80 which is a capital P
; will display in row 10, column 15
    mov ah,01h ;wait for character to be typed
    int 21h
        call     graphics_off        ;leave graphics mode
        mov      ah,4ch              ;end of job
        int      21h
cseg      ends
;***** Set up Stack Segment *****
sseg      segment stack
        db 50 dup (" ")
sseg      ends
;***** Set up Data Segment *****
dseg      segment byte public 'datasg'
    extrn yinit:word,yfinal:word,xinit:word,xfinal:word
    extrn xstart:word,xstop:word,ystart:word,ystop:word
    extrn gbmod:byte,curl0:byte,curl1:byte,curl2:byte
    extrn xinit:word,yinit:word
dseg      ends
end

```

Figure 2. A program that displays graphics mode, high resolution text.

```

greatr  db      11111111b
         db      10011111b
         db      11001111b
         db      11110011b
         db      11111001b
         db      11110011b
         db      11001111b
         db      10011111b
         db      11111111b
         db      11111111b

ques    db      11111111b
         db      11000011b
         db      10011001b
         db      11111001b
         db      11110011b
         db      11100111b
         db      11111111b
         db      11100111b
         db      11111111b
         db      11111111b

```

Figure 3. Graphics character definition of "greater than" and "question mark" from procedure GTEXT. "Rainbow Color/Graphics Option Programmer's Reference Guide", page 9-11.

An excellent source of programming projects is the development of routines that conform to established graphics standards. The Core Graphics System [4] for example, includes a series of text attribute commands. These commands include, SETCURRENTCHARSIZE which is used to set relative character size and CHARSPACE which is used to set X-axis and Y-axis displacement between the starting points of adjacent characters. Implementing commands such as these, while not trivial, is approachable through the use of procedures provided in the "Rainbow Color/Graphics Option Programmer's Reference Guide".

Digital has provide excellent documentation and commented assembler language source code in machine readable form for users who wish to use the Color/Graphics Option. Once working demonstration programs such as the one provided here are available the assembler language programming of the GDC is reasonable easy and makes an excellent supplementary exercise for courses in assembler language programming.

REFERENCES

1. "Rainbow Color/Graphics Option Programmer's Reference Guide", Digital Equipment Corporation", Product Number AA-AE36A-TV, June 1984.
2. *IBM PC/8088 Assembly Language Programming*, Avtar Singh, Walter A. Triebel, Prentice-Hall 1985.
3. "Graphics Programming Using the Core System", R. Daniel Bergeron, Peter R. Bono, James D. Foley, *ACM Computing Surveys*", Volume 10, Number 4, December 1978.

INTRODUCTION TO SPEAKEASY

David H. Saxe

Speakeasy Consultant
159 Wilson Crossing Road
Auburn, NH 03032
603-625-2019

ABSTRACT

"Speakeasy is a conversational computer language that has evolved over the past two decades through the continued use by a large and varied international community of users. A large audience of economists, research scientists, statisticians and students from a large variety of disciplines find Speakeasy a powerful yet natural means for using a computer. The modular structure of the language enables each group of users to adapt the system to its own needs by adding new words to the existing Speakeasy vocabulary."⁽²⁾ Speakeasy contains facilities for defining and operating on a variety of data structures including scalars, matrices, sets, time series and character data. This paper discusses the VAX implementation of Speakeasy and examines its use in an elementary statistical analysis. A method for extending the vocabulary and porting extensions between the VAX and IBM versions is described.

INTRODUCTION

Speakeasy is a conversational computer language in wide use as an interactive problem solving tool on VAX and IBM systems. Speakeasy provides an extremely user-friendly interface to a powerful set of tools for data analysis and presentation. Speakeasy was originally developed in the mid 1960's for large IBM mainframes to provide a data analysis tool to a research scientist community. The natural syntax made possible the direct use by the research scientist at a time when other languages required extensive familiarity with the computer system. Also, Speakeasy's design allowed for simple addition of new operations to meet the analysis needs of the user. Thus, the early and continued involvement of the end-user community in directing the evolution of the language contributed to the rapid expansion of language features and capabilities.

LANGUAGE FEATURES

Speakeasy offers both an interactive and program mode. In the interactive mode, a user is prompted with `:_` for a line of input. The user then types a Speakeasy statement consisting of references to data objects defined by the user and operators from the Speakeasy vocabulary. When the line is read, Speakeasy parses the line and controls the execution of operations that the user has specified. Results from the processing may be printed or assigned to a Speakeasy object. When the execution is complete, the user receives another prompt and may enter the next statement. In the program mode, collections of Speakeasy statements are executed as a single program. Special statements for use in the program mode allow for flow control.

Many different types of data objects are allowed by Speakeasy, including scalars, one and two dimensional arrays, vectors and matrices, time series and sets. Real, complex and character data may be used in most structures. A number of these are illustrated below.

Much of Speakeasy's power results from its ability to operate on collections of numbers or text without the user having to be concerned about dimensioning. Operators deal with entire objects, thus generally eliminating the need for looping and subscript operations. Presently, there are about 800 operators in the Speakeasy vocabulary, including numerical operations, such as `SQRT` which takes the square root of elements in an object, text operations, such as `TABULATE` which automatically formats and prints objects, and graphics operators which allow the presentation of results on a variety of device types. The operators are used in a natural syntax which resembles that of Fortran but is far more error tolerant. A general guideline is that if a line makes sense in an unambiguous way, then Speakeasy should be able to understand it.

Speakeasy's vocabulary may be extended and tailored to fit the needs of an individual user community. Large numbers of statistical, econometric and graphics operators have been added to the language by the user community.

Speakeasy offers extensive online documentation in the form of interactive tutorial sessions for learning to use the language, help documents for locating and using operators and examples of operator use.

LANGUAGE EXAMPLES

Speakeasy is best understood by actually looking at some simple examples. In this section, examples of the use of scalars, arrays and matrices are given. Later sections discuss the online documentation and demonstrate the use of Speakeasy in performing an elementary statistical analysis.

In the examples below, the user input is typed in lowercase after the `:_` prompt and the computer output has been set for uppercase. First, some examples of elementary scalar operations:

```
:_ 2 + 2
2 + 2 = 4

:_ 2 * 3 + 1
2 * 3 + 1 = 7

:_ 2 - 2
2 - 2 = 0

:_ sqrt(8) - 2/3
SQRT(8) - 2/3 = 2.1618

:_ answer
ANSWER = 2.1618

:_ answer + 4.94
ANSWER + 4.94 = 7.1018

:_ x = 5 * log(2)
:_ x
X = 3.4657

:_ angles in degrees
:_ y = cos(x) - 2.8
:_ y
Y = -1.8018

:_ names
X, Y, ANSWER
```

Arrays are objects with multiple elements arranged in a list (one dimensional array) or table (two dimensional array). Operations may be performed on a whole array as shown in these one dimensional array examples:

```
:_ a = 1, 3, 9, 6, x
:_ a
A (A 5 COMPONENT ARRAY)
  1      3      9      6      3.4657

:_ sum(a)
SUM(A) = 22.466
:_ answer/noels(a)
ANSWER/NOELS(A) = 4.4931

:_ mean a
MEAN A = 4.4931

:_ 2 * a + 3
2 * A + 3 (A 5 COMPONENT ARRAY)
  5      9      21      15      9.9315

:_ i = locs( a .gt. 3)
:_ b=a(i)
:_ tabulate i,b

I      B
* *****
3      9
4      6
5      3.4657
```

Two dimensional arrays are also provided. Note that arithmetic is performed element by element.

```
:_ a = a2d( 2, 3: integers(1,6) )
:_ a; 1/a
A (A 2 BY 3 ARRAY)
  1  2  3
  4  5  6

1/A (A 2 BY 3 ARRAY)
  1      .5      .33333
  .25    .2      .16667

:_ total=sumrows(a)
:_ tabulate a,total

A      TOTAL
***** *****
1 2 3   6
4 5 6  15

:_ a + 1/a
A + 1/A (A 2 BY 3 ARRAY)
  2      2.5      3.3333
  4.25   5.2      6.1667

:_ sqrt(a)
SQRT(A) (A 2 BY 3 ARRAY)
  1      1.4142  1.7321
  2      2.2361  2.4495

:_ a - 2
A - 2 (A 2 BY 3 ARRAY)
 -1  0  1
  2  3  4

Matrices obey the rules of matrix algebra. All of the elementary matrix operations are included. Several are demonstrated here:

:_ m = matrix(3,3:3 4 2 4 5 6 1 3 4)
:_ 1/m
1/M (A 3 BY 3 MATRIX)
 -.1  .5  -.7
  .5  -.5  .5
 -.35 .25 .05

:_ answer * m
ANSWER * M (A 3 BY 3 MATRIX)
  1      5.5511E-17  1.1102E-16
  0      1      -1.1102E-16
  7.8063E-18 -3.4694E-18  1

:_ eigenvalues(m)
EIGENVALUES(M) (A VECTOR WITH 3 COMPONENTS)
 -.89  2.0785  10.811
```

The online documentation provided includes tutorials and a tree structured help facility. A sample of the main menu for the tutorial is shown below along with a small portion of the tutorial dealing with matrix use. In the HELP sample, assume that the user wishes to perform a correlation but does not remember the correct word. By following the tree structure, the user may arrive at the document describing the word CORREL.

```
:_tutorial
INDEX PAGE 0
CONTENTS
Index to the Speakeasy tutorial sessions    December 1979
SESSION      SUBJECT
Start        An introduction to Speakeasy
Arrays       Array definitions and operations
Matrix       Matrix definitions and operations
Vector       Vector definitions and operations including
              vector-matrix operations
Logic        Use of logical and relational operators
Edit         Use of the editor
Stat         How to use the statistical routines
Keep         Saving information between runs
Tektron      Using the Tektronix Graphics Package
Tek          Tektronix graphing (older package)
Printgraph   Graphics for a printer.
Sets         Set definition and operations
Misc         Miscellaneous information
Type TUTORIAL XXX to begin the tutorial session called XXX.
Type TUTORIAL XXX N to display page N of the session, XXX.
(TUTORIAL XXX will give you a table of contents for that session.)
Type MORE to continue a session.
```

```
:_tutorial matrix 5
MATRIX PAGE 5
MATRIX ADDITION AND MULTIPLICATION
Rules for adding and multiplying matrices in Speakeasy are just
those utilized for matrices in mathematics. Hence, if
X=MATRIX(2,2:1,2,3,4) and Y=MAT(2,2:5,6,7,8),
then,
X+Y=  6  8      X-Y= -4 -4      X*Y= 19 22      Y*X= 23 34
      10 12      -4 -4          43 50          31 46.
Remember that the order of multiplying matrices does matter.
Let
A=MAT(3,2:1, 2, 5.6, 45, 23) and B=DIAGMAT(2:10,1).
So, A=  1  2      B= 10  0      A*B=  10  2
      5.6 45      0  1          56 45
      23  0          230  0.
Typing B*A will result in an error message because the sizes of A and
B are incompatible in matrix multiplication. Notice that both A+B
and B+A are undefined and therefore Speakeasy will also print out an
error message for them.
```

```
:_help
HELP explains how to use the HELP processor.
QUIT      is the command to leave Speakeasy.
OBJECTS   lists words dealing with structured objects.
MATH       lists mathematical functions.
ECONOMETRICS lists words which perform econometric analysis.
IOWORDS   words about data input, storage, output, and graphing.
DATAWORDS lists words relating to data organization or type.
PROGRAMS  lists words used in writing and running programs.
MISCELLANEOUS lists words not falling under any other classification.
DOCUMENT  explains how to use the Speakeasy documents.
EXAMPLE   explains how to use the Speakeasy Examples.
NEWS      lists information on new features in Speakeasy.
TUTORIAL  tells how to use the Speakeasy tutorial.
TSOPERATIONS lists words relating to TIMESERIES objects.
GRAPHICAL lists words relating to graphical operations.
HELP XXX  gives an explanation of the word XXX.
          XXX is any vocabulary word.
```

The following is the name of a tree structure document for operations which have not been included in the standard Help data set. CONTRIBUTIONS lists words contributed by users.

:_help math

MATH lists categories of mathematical functions.
DIFFEQUATIONS are words used to solve differential equations.
ELEMENTAL are elemental mathematical structures and functions.
FITTING are words which are used to fit or interpolate fcns.
INTEGRATION are words dealing with numerical integration.
LP are words dealing with linear programming.
PHYSICS are functions of interest primarily to physicists.
SINGLEVAR are functions of one variable.
SPECIAL are special mathematical functions.
STATISTICS are words related to statistical analysis.

To obtain the words in a given subclass SC, enter
HELP SC

:_help statistics

STATISTICS are words related to statistical analysis.
AUTOCOR returns a vector of autocorrelation coefficients.
AUTOCOV returns a vector of autocovariance coefficients.
AVERAGE returns the average value of the elements of an object.
CHIPROB calculates chi-squared probabilities.
CHISQUARED performs a chi-squared test.
COMBINATIONS gives the combinations of X items taken Y at a time.
CORREL returns a correlation matrix.
CORRELATION gives the correlation coefficient between 2 sets of data.
COVARIANCE returns a covariance matrix.
FPROB calculates f-statistic probabilities.
GETRANDOM returns the random number seed.
GETSEED returns the seed for the next invocation of NORMRAND.
KURTOSIS produces a coefficient of kurtosis.
LSQPOL finds a least-squares polynomial fit for two sets of data.
MEAN returns the mean of the elements of an object.
MEDIAN returns the median.
MODE returns the most frequently occurring value in an object.
MULTIREGRES performs multiple linear regression.
NORMRAND returns random numbers from a normal distribution.
PARTIALAUTO returns an array of partial auto-correlation coefficients.
PERMUTATIONS gives the permutations of X items taken Y at a time.
PROBIT scales data for a probability plot.
RANDOM generates random numbers.
RANGE returns the range of a series of real numbers.
RMS returns the root mean square.
SETRANDOM sets the random number seed.
SETSEED sets the seed for the next invocation of NORMRAND.
SKEWNESS returns a coefficient of skewness.
STANDDEV returns the standard deviation.
STANDERROR returns the standard error of the mean.
TINDEPT performs a t-test on two independent sets of data.
TPROB returns a significance value for a t-statistic.
TRELATE performs a t-test on two related sets of data.
TSAMPPOP performs a t-test on a sample and a population mean.
VARIANCE returns the variance of the elements of an object.

See also the MATRIXOPS tree structure document.
To obtain a description of a given word XXX, enter
HELP XXX.

:_help correl

CORREL(X1,X2) returns a correlation matrix.
For 1-dimensional arrays or vectors use CORREL(X1,X2,X3,...XN). The
(i,j)th entry is the coefficient of correlation between the ith and the
jth input arguments. X1 to XN must be (1-dimensional) arrays or vectors
with equal numbers of elements.

If CORREL is called with a 2-dimensional array or matrix as its only
argument, as in CORREL(X), the (i,j)th entry of the correlation matrix
is the coefficient of correlation between the ith and the jth rows
of X. X must have at least two rows and two columns.

Examples of the action of Speakeasy words use stored commands to demonstrate how a given word performs:

```
:_example median
```

EXAMPLES OF THE USE OF MEDIAN. SEE HELP STATISTICS FOR A LIST OF RELATED WORDS. RW

```
INPUT...MEDIAN(1,2,3)
MEDIAN(1,2,3) = 2
INPUT...MEDIAN(1,2,3,4)
MEDIAN(1,2,3,4) = 2.5
INPUT...
INPUT...A=(1,2,3,4)
INPUT...B=(5,6,7)
INPUT...C=(8,9)
INPUT...MEDIAN (A,B,C)
MEDIAN (A,B,C) = 5
```

STATISTICS EXAMPLE

In the following section, a set of data representing fuel economy ratings are used to demonstrate the use of a variety of statistical words. The variables involved are miles per gallon, MPG, make of car, CARMODEL, cubic inches of displacement, CID, number of cylinders, CYL, transmission type, TRANS, and number of gears, GEARS. HEAD was previously defined as a character object for the title of the TABULATE. Methods for obtaining simple statistics are presented and followed by a regression. Finally, a printer plot is made of MPG on CID. Using other plotter devices, regression lines could be drawn. While this example is extremely simple, it demonstrates the ease of interacting with the data to arrive at an understanding of any underlying relationship in the data.

```
:_tabulate(mpg,carmodel,cid,cyl,trans,gears:title=head)
```

EPA FUEL ECONOMY RATINGS FOR 1985 MODELS
TWO SEATERS - CITY DRIVING
SOURCE: USA TODAY, 9/24/84

MPG	CARMODEL	CID	CYL	TRANS	GEARS
21	Alfa Spider 2000	120	4	M	5
23	Bertone XI/9	91	4	M	5
16	Chevrolet Corvette	350	8	L	4
16	Chevrolet Corvette	350	8	M	4
25	Ford Exp	98	4	A	3
25	Ford Exp	98	4	M	5
23	Ford Exp	98	4	M	5
28	Honda Civic Coupe	91	4	L	3
31	Honda Civic Coupe	91	4	M	5
49	Honda Civic Coupe HF	91	4	M	5
17	Mazda RX-7	70	2	L	4
17	Mazda RX-7	70	2	M	5
16	Mazda RX-7	80	2	M	5
16	Mercedes 380SL	234	8	A	4
16	Nissan 300ZX	181	6	A	4
17	Nissan 300ZX	181	6	L	4
17	Nissan 300ZX	181	6	M	5
19	Nissan 300ZX	181	6	M	5
20	Pininfarina Spider	122	4	A	3
23	Pininfarina Spider	122	4	M	5
25	Pontiac Fiero	151	4	L	3
26	Subaru XT-DL	109	4	M	5

```
:_mean(mpg); median(mpg)
MEAN(MPG) = 22.091
MEDIAN(MPG) = 20.5
```

```
:_standdev(mpg) ; standdev(cid)
STANDDEV(MPG) = 7.5145
STANDDEV(CID) = 79.85
```

```
:_extrema(mpg)
EXTREMA(MPG) (A 2 COMPONENT ARRAY)
16 49
```

```
:_covariance mpg,cid,cyl,gears
COVARIANCE MPG,CID,CYL,GEARS (A 4 BY 4 MATRIX)
56.468 -250.68 -3.7749 .72727
-250.68 6376 130.15 -13.29
-3.7749 130.15 3.1948 -.24242
.72727 -13.29 -.24242 .62338
```

```
:_correl mpg,cid,cyl,gears
CORREL MPG,CID,CYL,GEARS (A 4 BY 4 MATRIX)
1 -.41778 -.28105 .12258
-.41778 1 .91188 -.2108
-.28105 .91188 1 -.17178
.12258 -.2108 -.17178 1
```

```
:_eigenvals(answer)
EIGENVALS(ANSWER) (A VECTOR WITH 4 COMPONENTS)
.075776 .7944 .92719 2.2026
```

```
:_regress(mpg,c,cid,cyl,gears)
```

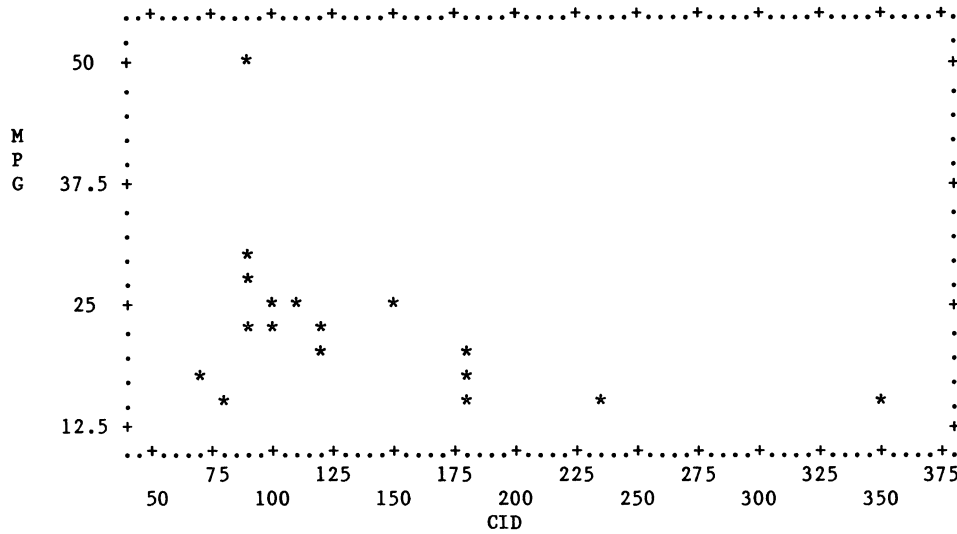
ORDINARY LEAST SQUARES ESTIMATION

DEPENDENT VARIABLE: MPG

NAME	LAG	COEFF	STD ERROR	T-STATISTIC
#C	0	22.466	10.369	2.1666
CID	0	-.089503	.047719	-1.8756
CYL	0	2.4815	2.1153	1.1731
GEARS	0	.22353	2.0107	.11117

R-SQUARE = .23433
R-SQUARE (CORRECTED) = .10671
NUMBER OF OBSERVATIONS = 22
DURBIN WATSON STATISTIC= 1.4148
SUM OF SQUARED RESIDUALS = 907.95
STD ERROR OF REGRESSION = 7.1022

:_graphz mpg cid



PROGRAM EXAMPLE

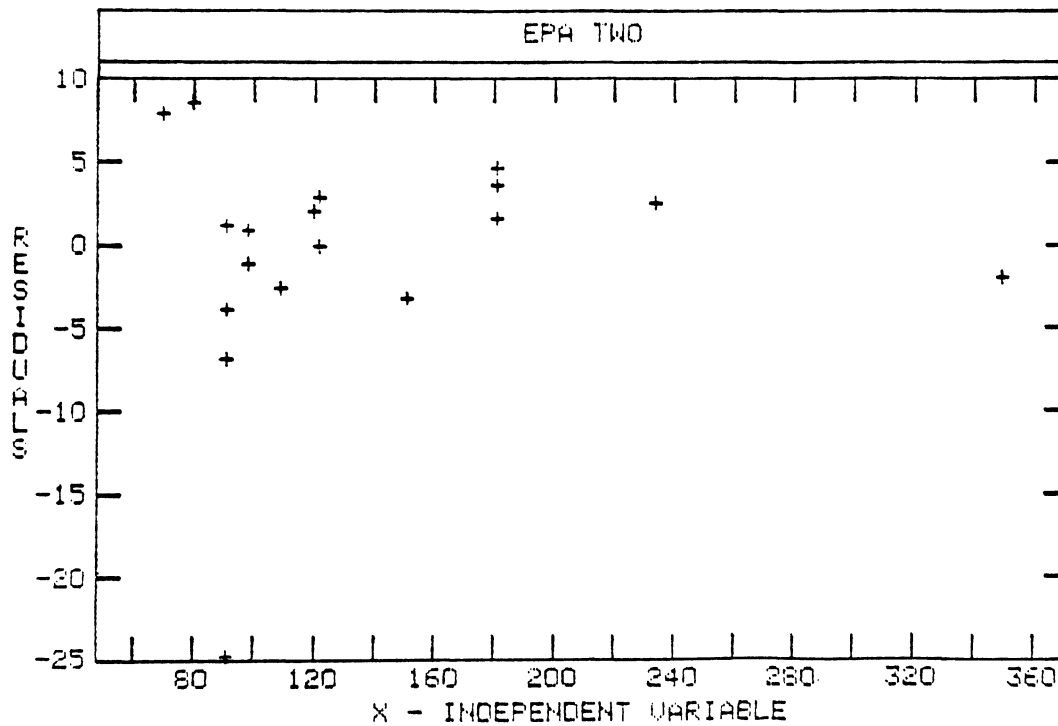
Once the procedures for an analysis have been set, a Speakeasy program may be written to perform the analysis using different sets of data. The program shown below performs a simple regression and draws plots of the variables, predicted values and residuals. The program REGPLOT was prepared using a standard editor and was then invoked in Speakeasy by simply typing its name. Only one of the plots is shown.

```
EDITING REGPLOT
1 PROGRAM
2 GRAPHICS TEK4010
3 ERASE
4 "ENTER NAMES OF VARIABLES OR RETURN TO KEEP THE SAME VARIABLES"
5 ASK ("DEPENDENT VARIABLE (Y) ", "Y=")
6 ASK ("INDEPENDENT VARIABLE (X) ", "X=")
7 ASKLIT("ENTER TITLE","SETTITLE")
8 N=NOELS(X)
9 N
10 MEAN(X);STANDDEV(X)
11 MEAN(Y);STANDDEV(Y)
12 CORRELATION(X,Y)
13 COEF=MULTIREGRES(X,Y:RESID,MULTR)
14 "REGRESSION OF Y ON X GIVES FOLLOWING COEFFICIENTS"
15 TYPE "Y = ",COEF(1)," + ",COEF(2)," * X"
16 SUMSQ(RESID)
17 HARDCOPY
18 SETXLABEL("X - INDEPENDENT VARIABLE")
19 SETYLABEL("Y DEPENDENT VARIABLE")
20 LINECODE=-2
21 GRAPH(Y:X)
22 X1=MIN(X)-1000,MAX(X)+1000
23 Y1=COEF(1)+COEF(2)*X1
24 LINECODE=1
25 ADDGRAPH(Y1:X1)
26 X1=MEAN(X) ; Y1=MEAN(Y)
27 LINECODE = -5
28 ADDGRAPH(Y1,X1)
29 HARDCOPY
30 LINECODE=-2
31 SETYLABEL("RESIDUALS")
32 GRAPH(RESID:X)
33 HARDCOPY
34 SETXLABEL("PREDICTED Y VALUES")
35 GRAPH(RESID:RESID+Y)
36 HARDCOPY
37 SETXLABEL("ACTUAL Y VALUES")
38 GRAPH(RESID:Y)
*39 HARDCOPY
```

```

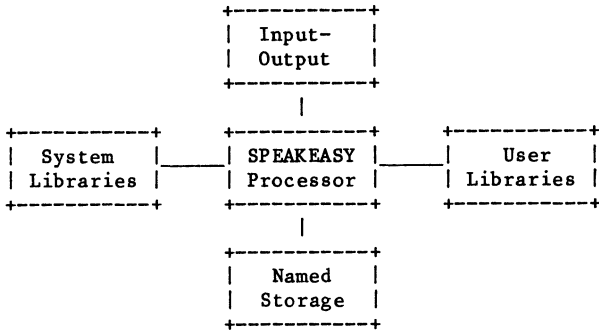
:_regplot
EXECUTION STARTED
ENTER NAMES OF VARIABLES OR RETURN TO KEEP THE SAME VARIABLES
DEPENDENT VARIABLE (Y) mpg
INDEPENDENT VARIABLE (X) cid
ENTER TITLE epa two
N = 22
MEAN(X) = 143.64
STANDDEV(X) = 79.85
MEAN(Y) = 22.091
STANDDEV(Y) = 7.5145
CORRELATION(X,Y) = -.41778
REGRESSION OF Y ON X GIVES FOLLOWING COEFFICIENTS
Y = 27.738 + -.039316 * X
SUMSQ(RESID) = 978.85

```



SYSTEM STRUCTURE

One of the most important features of Speakeasy is the ability to add words to the language to tailor it to a specific user community's need. Words in Speakeasy are really just Fortran function subprograms, so that it becomes possible to convert local libraries of analysis routines to operate in the Speakeasy environment. To understand how this works, the following section briefly describes Speakeasy's internal structure shown in figure 1.



Speakeasy System Structure

figure 1

The Speakeasy processor is responsible for accepting input from the user, parsing it and directing the execution of the various operations that have been requested. The processor interfaces with the (Speakeasy) system libraries to map, activate and unmap the operators as required. A very few operators are actually implemented in the processor. The processor also maintains an area of memory called named storage used to store all objects used in a session.

All input and output during a Speakeasy session is controlled through the processor. In addition to the user prompts and replies shown above, Speakeasy also provides error handling and, at the user's request, will log any portion of a session. A number of the internally used routines are available for use when adding operators to Speakeasy.

Named storage is an area of memory maintained by the processor that contains all Speakeasy objects that are in use. Objects may be defined, read, modified and freed. Named storage is dynamically maintained with efficient algorithms for locating and using objects.

The system libraries contain the help, tutorial and example files, data to be stored between sessions and the operators which are also known as "linkules." Each user may also create libraries corresponding to the system copies for individual modifications or enhancements.

SPEAKEASY LINKULES

As mentioned above, Speakeasy linkules are just Fortran functions. Each linkule is an executable file that is mapped and activated by the processor when first used. Speakeasy maintains internal tables of which linkules have been used and will not unmap a linkule until necessary. This retention of linkules speeds repeated use since the linkule does not have to be remapped. The processor communicates with the linkule through a standard calling sequence which provides information about named storage and how the linkule was invoked by the user. The linkule may use named storage to examine values of objects and to compute and define results. It is the responsibility of the linkule writer to check the calling sequence specified by the user for errors and to produce error messages. If the arguments specified are acceptable, space may be reserved for a result and the result computed and returned to the user. Typically, a linkule is written by writing a code fragment that handles these details and then calls a computational routine. For example, SQRT would check whether it had received a positive real argument and then pass that argument to the correct routine for computing square roots of real numbers. Finally, the linkule's Fortran function value is used to tell the processor if errors occurred and if a result was defined.

A Fortran macro preprocessor, Mortran, may be used to assist in preparing linkules. If linkules are being used only on the VAX, Mortran can be used to generate the standard calling sequence and define a number of important Fortran variables. Mortran's primary purpose, however, is to isolate machine-dependent code in macro form. For instance, Mortran has a macro which represents the largest real number. The linkule writer types CONSTANT(BIGEST) in the Mortran source file, specifies a target machine and runs the preprocessor. Mortran expands the macro and outputs a file of Fortran statements that will compile on the target system. On the VAX, CONSTANT(BIGEST) expands to 'FFFFFFFF7FFF'X and on IBM to Z7FFFFFFFF FFFFFFFF. Only the Fortran compile has to be run on the target system as long as the appropriate macro files exist. Thus, it is possible to use the VAX as a development system for linkules which will run under the IBM version of Speakeasy. Speakeasy itself is written in Mortran and uses this facility to support versions for different machines.

CONCLUSION

Speakeasy's ease of use and extensive vocabulary make it an ideal tool for interactive data analysis. Analytical techniques may be developed and formulated as programs for general use. Existing libraries of Fortran subprograms may be added to the vocabulary, thereby extending and tailoring the language to meet special needs of virtually any user community. Moreover, by using Mortran, additions to the language are portable across different machine versions.

REFERENCES

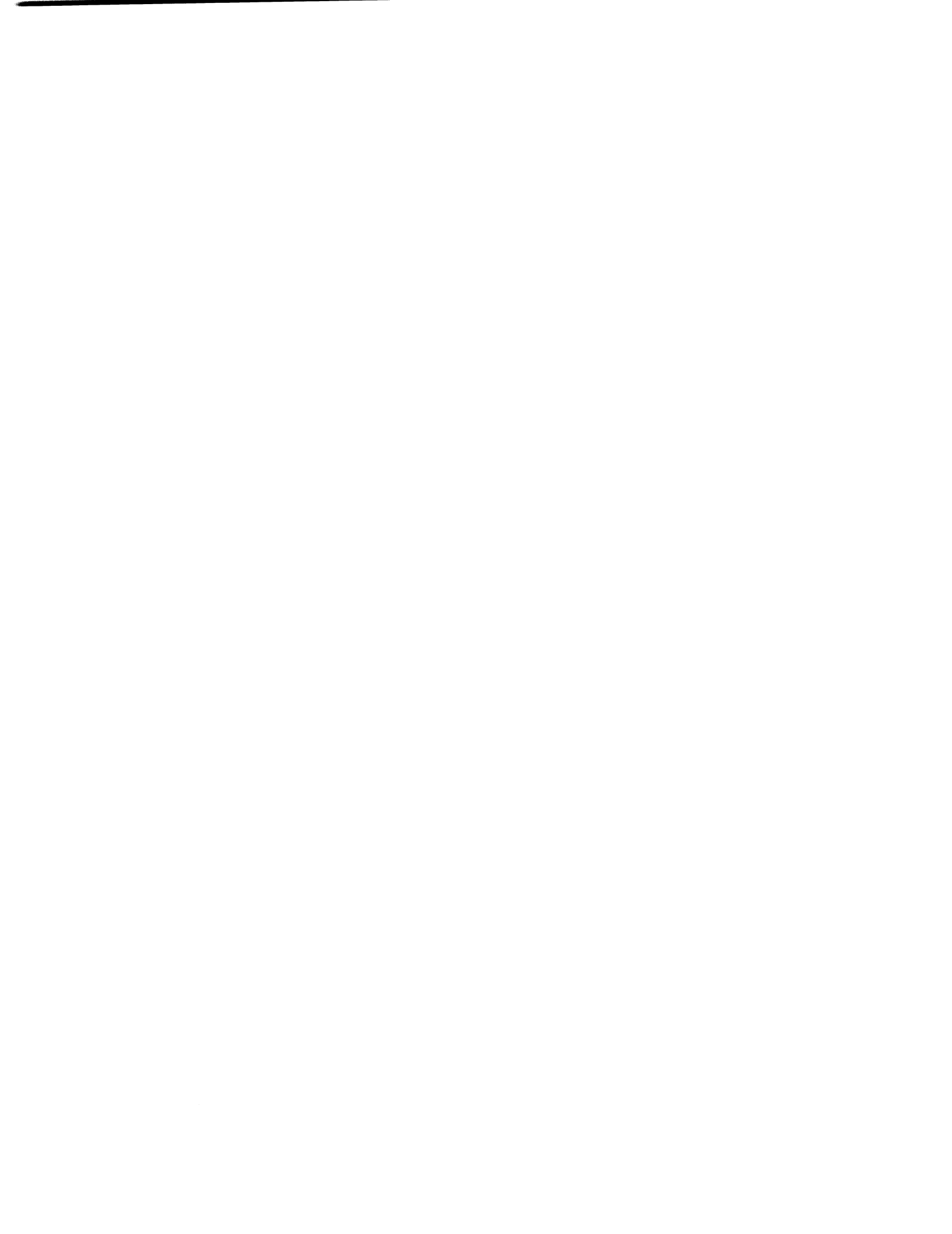
For the reader who wishes to learn more about Speakeasy, the following reading list is suggested.

- [1] Cohen, Stanley, "A Look at Speakeasy, The Interactive Computing System That Found a Home in VAX", VAX RSTS Professional, August, 1984, Vol. 6, No. 4, pp. 26 - 34.
- [2] Cohen, Stanley, "Speakeasy: A Conversational Language on VAX", Proceedings of the Digital Equipment Corporation User Society, Spring, 1983, pp. 1 - 7.
- [3] Saxe, David, "Introducing Speakeasy to the New User", Speakeasy Meeting: 13th Annual Conference Proceedings, 1985, Speakeasy Computing Corporation, Chicago, IL.
- [4] Introduction to Speakeasy IV Linkule Writing, 1985, Speakeasy Computing Corporation, Chicago, IL.
- [5] Lectures on Speakeasy, 1984, Speakeasy Computing Corporation, Chicago, IL.
- [6] Sampler, 1985, Speakeasy Computing Corporation, Chicago, IL.
- [7] Speakeasy IV Help Documents, 1985, Speakeasy Computing Corporation, Chicago, IL.
- [8] The Speakeasy Reference Manual, 1983, Speakeasy Computing Corporation, Chicago, IL.

ACKNOWLEDGEMENTS

SPEAKEASY and :_ are trademarks of the Speakeasy Computing Corporation.

The author wishes to thank Stanley Cohen of the Speakeasy Computing Corporation for the use of the Speakeasy development VAX in preparing the example sessions used in this paper.



Extraction of 1022 Data to PC Files:
New INIT and PRINT Features in V117B

John Duesenberry
Software House

1.0 OVERVIEW

This paper describes new features in V117B for the extraction of 1022 data to spreadsheet and other applications on personal computers. Enhancements to the System 1022 PRINT and INIT commands provide users of Lotus 1-2-3 and Symphony with data in Lotus Worksheet file format from within 1022. 1022 also provides data in Data Interchange Format (DIF).

The following examples illustrate command sequences such as a user might employ to obtain data at his or her PC. We will assume that the user's link between the PC and the host computer is MOBIUS.

2.0 CREATING LOTUS PRN FILES WITH FORMATTED PRINTS

Before considering the new 117B features, let us briefly look at what is probably the most direct method previously available for extracting 1022 data into a file that Lotus 1-2-3 can translate.

FIGURE 1 is an example program; MAKPRN.DMC creates a file in Lotus 'PRN' format. In this format, all data items are separated by commas, text strings are delimited by quotes, and each CRLF delimits a spreadsheet row.

```
!                               MAKPRN.DMC
!       A DMC to write out 1022 data in Lotus PRN format.
!
OPEN MOBDEM RO.
F SYSID BET 1 10.
SORT LN FN.
INIT 2 DEMO2.PRN.
PRINT ON 2 "From 1022 Dataset: "+$TRIM(SYSDSNAME)+" in " -
+$TRIM(SYSDSFILE) FMT "' ,lx,',' , ' ','A,'" END.
PRINT ON 2 SYSDATE FMT -
"' ,lx,',' , ' ','"Extracted on: ",',' , ' ','D2,'" END.
!skip a row
PRINT ON 2 FMT "' ,'" END.
PRINT ON 2 "FIRST NAME" "LAST NAME" "# CHILDREN" "CITY" -
"STATE" "ZIP" FMT 5(' ','A,',' ,') "' ,A,'" END.
!skip a row
PRINT ON 2 FMT "' ,'" END.
PRINT ON 2 FN, LN, NCH, CITY, STATE, ZIP FMT -
2(' ','A,',' ,'), I, 2(' ',' ','A, ' "' , ' , ' ) "' ,A,'" END.
!skip a row
PRINT ON 2 FMT "' ,'" END.
PRINT ON 2 MEAN(NCH) FMT -
"' ,lx,',' , ' ','"Average # Children:",',' , ' , F2.1 END.
RELEASE 2.
TYPE "DEMO2.PRN has been created on host. -
It can be FILE IMPORTed to 1-2-3."
```

Fig. 1

FIGURE 2 is the actual output of MAKPRN.DMC. Note the use of quote-delimited spaces in order to 'indent' 1 cell at the start of some rows. The null string is printed to skip a row.

```
" ", "From 1022 Dataset: NEW in MOBDEM.DMS"
" ", "Extracted on: ", "Jul-29-1985"
""
"FIRST NAME", "LAST NAME", "# CHILDREN", "CITY", "STATE", "ZIP"
""
"CHARLES", "CARAGIANES", 3, "DEDHAM", "NY", "02138"
"RICH", "GARLAND", 4, "BRISTOL", "CT", "02138"
"CHARLES", "GOTT", 2, "BRISTOL", "CT", "22209"
"KATHY", "HOUSMAN", 0, "AUGUSTA", "GA", "43220"
"MARK", "JONES", 0, "DEDHAM", "NY", "60064"
"ROGER", "LEVINSON", 3, "BRISTOL", "CT", "11729"
"LOUIS", "MERZ", 0, "ROXBURY", "TX", "77056"
"OLGA", "PONG", 3, "STONEHAM", "MA", "02238"
"ALFRED", "SAVIO", 1, "ROXBURY", "IN", "46225"
"ALFRED", "STEVENS", 0, "BRISTOL", "CT", "02238"
""
" ", "Average # Children:", 1.6
```

Fig. 2

FIGURE 3 shows the sequence of commands the user would give in order to create the .PRN file on the host and load it into 1-2-3 after having defined the device D: as the host area where the datasets and output files reside and having created the PC command "1022" with the MOBIUS MAKE feature:

```
A>1022
*USE MAKPRN
DEMO2.PRN has been created on host.
It can be FILE IMPORTed by 1-2-3.
*QUIT
A>123
/File Import Numbers D:DEMO2
```

Fig. 3

Having imported the file and changed the width of a few spreadsheet columns, the user sees the 1-2-3 screen of FIGURE 4.

From 1022 Dataset: NEW in MOBDEM.DMS
Extracted on: Jul-29-1985

FIRST NAME	LAST NAME	# CHILDREN	CITY	STATE	ZIP
CHARLES	CARAGIANES	3	DEDHAM	NY	02138
RICH	GARLAND	4	BRISTOL	CT	02138
CHARLES	GOTT	2	BRISTOL	CT	22209
KATHY	HOUSMAN	0	AUGUSTA	GA	43220
MARK	JONES	0	DEDHAM	NY	60064
ROGER	LEVINSON	3	BRISTOL	CT	11729
LOUIS	MERZ	0	ROXBURY	TX	77056
OLGA	PONG	3	STONEHAM	MA	02238
ALFRED	SAVIO	1	ROXBURY	IN	46225
ALFRED	STEVENS	0	BRISTOL	CT	02238

Average # Children: 1.6

Fig. 4

To the end user, this is quite straightforward, thanks to MOBIUS. However, if we return to the DMC of FIGURE 2, various deficiencies become apparent, from the viewpoint of the programmer who must write such DMC's:

- * Coding the format statements is tedious and error-prone. The code becomes virtually unreadable, and making a simple change in the program, such as adding a new item to a PRINT list, is more difficult than one would wish.
- * Kludges (the aforementioned blanks and null strings) must be used if empty cells or rows are desired.
- * Ad hoc queries are difficult, especially for end users who are likely to know little of 1022 PRINT formats.
- * The resulting file is not native to 1-2-3, in the sense that the PRN file must be translated and loaded into the spreadsheet and the result saved to a WKS file.

3.0 CREATING A LOTUS WORKSHEET FILE DIRECTLY FROM 1022

By way of contrast, consider FIGURE 5. MAKWKS.DMC is a working program that uses the V117B enhanced INIT and PRINT commands to produce a 1-2-3 worksheet file directly from 1022. A cursory glance shows that the FORMAT statements have been eliminated or greatly simplified.

```
!           MAKWKS.DMC
!           A program to create a Lotus WKS file
!           using V117B INIT/PRINT features
OPEN MOBDEM RO.
F SYSID BET 1 10.
SORT LN FN.
INIT 123 2 DEMO3.
PR ON 2 "From 1022 Dataset: "+$TRIM(SYSDSNAME)+" in " -
+$TRIM(SYSDSFILE) FMT 1X,A END.
PRINT ON 2 SYSDATE FMT 1X,"Extracted on: ",L8 / END.
PRINT ON 2 "FIRST NAME" "LAST NAME" "# CHILDREN" "CITY" -
"STATE" "ZIP" .
PRINT ON 2 ALL.
PRINT ON 2 MEAN(NCH) FMT /,1X,"Average # Children: ",L1.1 END.
RELEASE 2.
TYPE "DEMO3.WKS has been created on host. -
It can be FILE RETRIEVED by 1-2-3."
```

Fig. 5

Before walking through the code, let's look at how the user would produce the file with MOBIUS and 1022 (FIGURE 6):

```
A>1022
*USE MAKWKS
DEMO3.WKS has been created on host.
It can be FILE RETRIEVED by 1-2-3.
*QUIT
A>123
/File Retrieve D:DEMO3
```

Fig. 6

FIGURE 7 is a screen dump from 1-2-3 after loading the resulting file. The results are practically identical to those obtained with the PRN file, with the exception of the formatting of SYSDATE. (This will be explained below).

From 1022 Dataset: NEW in MOBDEM.DMS
 Extracted on: 30-Jul-85

FIRST NAME	LAST NAME	# CHILDREN	CITY	STATE	ZIP
CHARLES	CARAGIANES	3	DEDHAM	NY	02138
RICH	GARLAND	4	BRISTOL	CT	02138
CHARLES	GOTT	2	BRISTOL	CT	22209
KATHY	HOUSMAN	0	AUGUSTA	GA	43220
MARK	JONES	0	DEDHAM	NY	60064
ROGER	LEVINSON	3	BRISTOL	CT	11729
LOUIS	MERZ	0	ROXBURY	TX	77056
OLGA	PONG	3	STONEHAM	MA	02238
ALFRED	SAVIO	1	ROXBURY	IN	46225
ALFRED	STEVENS	0	BRISTOL	CT	02238

Average # Children: 1.6

Fig. 7

Returning to FIGURE 5, notice the following features of the DMC:

- * The new '123' keyword in the INIT command informs 1022 that any output directed to PRINT channel 2 must be formatted as 1-2-3 data cells, rather than the usual ASCII strings. The extension for the output filespec is defaulted to .WKS, and several other actions are taken in the background at INIT-time. The most significant action is the initialization of internal counters which, in effect, always point to the spreadsheet cell to which 1022 will next PRINT data. These counters are automatically updated in the course of printing, and are also accessible to user programs in the form of system variables. They will be discussed in more detail below. For now, it suffices to say that in our example, after the INIT command the counters will point to cell A1, by default.
- * There are two instances in the example of a new format spec - "L" format. (The L stands for Lotus). This format serves a dual purpose:
 1. It enables a 1022-to-1-2-3 data transformation which maps 1022 datatypes (integer, real, date, double integer, or text) into Lotus datatypes (integer, real, or text [label]).

2. It allows the user to specify a Lotus display format to be used with the item. In our example, a date(SYSDATE) is printed under L8 format. This causes a Lotus binary date to be written, with its format code set to (D1) - 1-2-3's day-mon-yr format. (This is why SYSDATE shows up differently in FIGURE 7). In the last PRINT of the example, a 1022 function result (type real) is printed under L1.1 format. The 1022 real is converted to a Lotus real, with a format code set to Fixed format, 1 decimal place."L1" format signifies Fixed format, with the argument after the "."indicating the desired number of decimal places.

* Unformatted PRINT statements also occur in the DMC. Unformatted PRINTS simply default to "L" format by virtue of the fact that they are directed to a PRINT channel that has been INITED to a 1-2-3 file. Thus, in the PRINT ALL statement, 1022 looks at each attribute it is to write out, and produces a 1-2-3 label, integer, or real record on the basis of the attribute's datatype.

The data transformations made under L format are summarized in FIGURE 8:

1022 TYPE	LOTUS TYPE
INTEGER	INTEGER or REAL *
DOUBLE INTEGER	INTEGER or REAL *
REAL	REAL (8087 DP floating point)
DATE	INTEGER or REAL **
TEXT	LABEL***

* Integers or double integers that exceed the range +/- 32767 will be converted to 1022 reals and then to Lotus floating point.

** The stored binary date is offset such that Jan.1,1900 = 1. The Lotus format byte is set for (D1) format.

***The maximum length of a LABEL is 240 characters, including a Label-Prefix ('') and a terminating null, both of which are automatically added to the string by 1022. Any text expression or literal longer than 238 characters will be truncated to 238 characters.

Fig. 8

FIGURE 9 summarizes the possible L format specs and the resultant Lotus format types:

Form of an L-format spec: rLm.n

where r = repeat count
m = integer signifying Lotus format type
n = number of decimal places ($0 \leq n \leq 15$)

Default for m = 12

Default for n = 2, in accordance with 1-2-3 default
(n is applicable only to types 1-5 below)

Values of m	Lotus type
1	fixed
2	scientific
3	currency (\$)
4	percent
5	comma (xxx,xxx.xx)
6	+/- horiz. bar graph
7	general
8	day/mon/yr
9	day/mon
10	mon/yr
11	text
12	default

Fig. 9

- * Notice also in FIGURE 5 the presence of conventional 1022 format specs: "A" format, "/" format, "X" format and quoted literals. Conventional 1022 formats, in the context of PRINTing to 1-2-3 files, work differently than in normal printing.

"X" format is a means to skip some number of cells within a row. Thus, in our example the 1X format in the first PRINT command repositions the internal cell counter such that the first actual data item in the worksheet is in cell B1.

"/" format is a means to skip some number of rows. The "/" spec at the end of the 2nd PRINT command in the example repositions the internal cell counter such that the next item will be written to cell A3, instead of to the next consecutive row.

Most of the other conventional 1022 formats (such as the "A" format in the first PRINT command or the literal in the second) will cause a LABEL to be written to the worksheet, whether the type of the 1022 expression being printed was text or not. The content of the label will be an ASCII string identical to that which 1022 would have produced while printing 'normally'. For example, consider the following command sequence:

```
*INIT 123 3 FOO.WKS.  
*DEFINE INTEGER Q. LET Q 9999.  
*PRINT ON 3 Q Q+1 FMT L I.  
*RELEASE 3.
```

When loaded into 1-2-3, the spreadsheet will contain the binary integer 9999 in cell A1, while cell B1 will contain a label consisting of the DIGIT STRING '10000'. Since, presumably, most of the data that 1022 users will want to extract will be for computational purposes, they will therefore wish in most instances to use L format, which is the default.

3.1 Summary Of 1-2-3-Related INIT/PRINT Features

Our example has covered the basics of PRINTING to Lotus 1-2-3 files. Let us now summarize the points covered so far, and explore further options available under this file format:

3.1.1 L Format - is the default format used when PRINTING to a channel that has been INITED to a 1-2-3 file. The optional specification m.n following the L selects the Lotus format type and (if applicable) number of decimal places. L format MUST be used to derive computational spreadsheet data from 1022 numeric data.

3.1.2 Conventional Formats - such as I, F, E, A, D, etc. are used to produce labels. /, X, and \$ are among the means available for controlling the cell location of data. H format and literals also produce labels.

3.1.3 Data-Positioning Options - One requirement of Lotus worksheet format is that the cell coordinates of every data item be included in the data record. In order to do this, 1022 maintains three counters for each channel that has been initialized to a 1-2-3 file. These counters are available to user programs in the form of three system variables, indexed on channel number N.

3.1.3.1 Cell Location Counters SYSPCROW, SYSPCCOL, SYSPCRESET -

- * SYSPCROW(N) points to the spreadsheet row to which 1022 is currently printing or is about to print. By default, SYSPCROW points to row 1 at INIT-time and is incremented upon completion of every PRINT command. "/" format may be used to increment SYSPCROW at any time. SYSPCROW is user-settable.
- * SYSPCCOL(N) points to the spreadsheet column to which 1022 is currently printing or is about to print. By default, SYSPCCOL points to column A at INIT-time. It is then incremented once for every cell produced in a given PRINT command. "nX" format adds n to SYSPCCOL, effectively skipping cells. Whenever a new-row action is triggered (as at the conclusion of a PRINT command or the execution of a "/" format), SYSPCCOL is reset to point to the column whose value is stored in a third PC-related variable, SYSPCRESET.
- * SYSPCRESET(N) points to the column at which each new row is to start. By default, SYSPCRESET points to column A at INIT-time.

SYSPCCOL and SYSPCRESET, like SYSPCROW, are user-settable.

NOTE that all the SYSPC variables are "zero - origin": that is, when the column and row counters are pointing to A1, they are both set to 0. To advance SYSPCROW to row 5 and SYSPCCOL to column D, therefore:

```
LET SYSPCROW(N) 4 SYSPCCOL(N) 3.
```

3.1.3.2 \$SYSPCPOS Function - As a convenience, the string function \$SYSPCPOS(N) has been added. \$SYSPCCOS(N) returns the ASCII representation of the cell coordinates to which SYSPCROW(N) and SYSPCCOL(N) currently point. An example:

```
*LET SYSPCROW(2) 5 SYSPCCOL(2) 4
*TYPE $SYSPCPOS(2)
*E6
```

3.1.3.3 Controlling Data Position Via The SYSPC Variables -
 FIGURE 10 shows an example of manipulating the position of data
 within a target spreadsheet by changing the values of the SYSPC
 variables.:

```
!program fragment showing placement of data in spreadsheet
!via alteration of SYSPCxxx variables.
!attributes AT1...AT5 are from GOO.DMS
!attributes VV...ZZ are from POO.DMS
!
      OPEN GOO.DMS POO.DMS. FIND SYSID BET 1 10.
      INIT 123 2 FOO.WKS.
!start at A1
!SYSPCROW(2),SYSPCRESET(2),SYSPCCOL(2) all=0
!fill COLS A-E of ROWS 1-10
      PRINT ON 2 AT1 AT2 AT3 AT4 AT5.
      DBS POO.DMS. FIND SYSID BET 1 10. !go to another ds
! right now, SYSPCROW=10,SYSPCCOL=0
      LET SYSPCROW(2) 0. !reset to row 1
      LET SYSPCRESET(2) 6. !slide over to col.#G
!NOTE!! SYSPCRESET is now pointing to COL. G. but if we
!do not also set SYSPCCOL, SYSPCCOL will be set to COL. A
!when the first record is printed below.
      LET SYSPCCOL(2) SYSPCRESET(2).
      PRINT ON 2 VV WW XX YY ZZ !fill cols G-K on rows 1-10
      :
      :
```

Fig. 10

3.1.3.4 Setting The SYSPC Variables At INIT-Time - Two new
 arguments have been provided in order to allow the user to set
 initial values of SYSPCROW, SYSPCCOL, SYSPCRESET prior to
 PRINTing data. The syntax is as follows:

```
INIT 123 [COL c] [ROW r]...
```

If COL is present, SYSPCRESET and SYSPCCOL are set to c.
 If ROW is present SYSPCROW is set to r. The defaults for c and
 r are A1, as mentioned previously.

3.1.3.5 Columnwise Vs. Rowwise Data Formatting - In all our
 examples thus far, data cells have been written in left-to-
 right fashion within each PRINT command, with the row counter
 advancing down the spreadsheet upon each new PRINT. This is
 termed "columnwise" representation in spreadsheet parlance, and
 is the default action. However, a "rowwise" representation is
 also possible when printing to a 1-2-3 file. We include an
 optional CWISE/RWISE clause in the INIT command for this
 purpose:

```

                                CWISE
INIT 123 [COL c] [ROW r] {      } .....
                                RWISE

```

FIGURE 11 shows a DMC using these options. FIGURES 12A-12B show the spreadsheet results:

```

!this DMC prints the same data to two Lotus WKS files.
!the first files is INITED CWISE (by default)
!while the second is INITED RWISE.
INIT 123 2 EELS1.WKS.
INIT 123 RWISE 3 EELS2.WKS.
DEF TEXT 10 A B C D E F.
LET A "My" B "Hovercraft" C "is" D "full" E "of" F "eels." .
PRINT ON 2 A B C D E F.
PRINT ON 3 A B C D E F.
RELEASE.

```

Fig. 11

```

B1: 'Hovercraft                                     READY
      A          B          C          D          E          F          G
1  My  Hovercraft  is      full      of      eels.
2
3
4
5
6
7

```

Fig. 12A - CWISE

```

A1: 'My                                             READY
      A          B          C          D          E          F          G
1  My
2  Hovercraft
3  is
4  full
5  of
6  eels.
7

```

Fig. 12B - RWISE

As one might imagine, the underlying effect of the RWISE option is simply to switch counters; SYSPCROW is auto-incremented as individual items are printed, "X" formats executed etc. SYSPCCOL is auto-incremented on the end of each PRINT command, by "/" formats, etc. Programmers who want to change the counters would do well to remember this.

3.1.3.6 "\$" Format - "\$" format behaves as an analog to "\$"-format in normal printing: it disables automatic incrementing of SYSPCROW and resetting of SYSPCCOL at the end of a PRINT command. This enables your program to (for example) print some data to a given row, "save its place" and do more calculation, and resume printing in the same row.

3.1.4 NAMED RANGE (NRANGE) Option - The final INIT option for 1-2-3 files is the ability to designate a block of cells within a worksheet as a Named Range:

```
INIT 123 NRANGE FRED B3 D10
```

uses the defaults for COL,ROW, and CWISE, and creates the Named

Range FRED in the worksheet. One could use this with the File Combine feature, for example, to extract the subset FRED of the 1022 data into 1-2-3.

The user may define as many Named Ranges as desired by using multiple NRANGE clauses. There is no limit on the number of such clauses.

The default is no NRANGE present.

4.0 CREATING DIF FILES DIRECTLY FROM 1022

DIF is an ASCII format; therefore DIF files can be written from 1022 using normal formatted PRINTS. FIGURE 13 is an example:

```
!This DMC extracts 1022 records and fields and produces a DIF
!file. Note that the number of 'VECTORS' = 7 (6 attr's and one
!blank cell) and IWDITH is set accordingly. The number of 'TUPLES'
!equals the number of selected records, + 1 for the 'tuple' of
!labels, + 1 for the blank 'tuple' and NTUPLES is set accordingly.
CLEAR.
OPEN MOBDEM.DMS RO.
F SYSID BET 1 10. SORT LN FN.
INIT 2 DEMDIF.DIF.
DEF TEXT 9 SKIPCELL TEXT 11 BOT EOD TEXT 2 CRLF TEXT 63 SKIPROW.
LET CRLF $CHAR(13)+$CHAR(10).
LET SKIPCELL "1,0"+CRLF+" "+CRLF.
LET SKIPROW SKIPCELL+SKIPCELL+SKIPCELL+SKIPCELL+SKIPCELL -
+SKIPCELL+SKIPCELL.
LET BOT "-1,0"+CRLF+"BOT"+CRLF.
LET EOD $REPLACE("BOT","EOD",BOT).
LET IWIDTH 7. ! kludge to setup correct VECTORS item
LET NTUPLES SYSNREC+2. !kludge works as long as we print IWIDTH
!cells for SYSNREC+ (# of label rows and
!blank rows) records
!print the header section, note vectors and tuples counts.
PR ON 2 "TABLE" "0,1" " " "VECTORS" "0," IWIDTH -
FMT 4(G / ) 2G / " " END.
PR ON 2 "TUPLES" "0," NTUPLES FMT G / 2G / " " END.
PR ON 2 "DATA" "0,0" " " FMT 2(G / ) G END.
!print the data section. start with a row of text labels, with a
!blank cell in col. A (all rows will be like this)
PR ON 2 BOT "FIRST NAME" "LAST NAME" "# CHILDREN" "CITY" -
"STATE" SKIPCELL "ZIP" FMT G 5("1,0" / " " G " " / ) G "1,0" / -
" " G " " END.
!now print a row of blank cells
PR ON 2 BOT SKIPROW FMT G G $ END.
!now print the stuff from the records
PR ON 2 BOT FN LN NCH CITY STATE SKIPCELL ZIP -
FMT G 2("1,0" / " " G " " / ) "0," G / -
"v" /2("1,0" / " " G " " / ) G "1,0" / " " G " " END.
PR ON 2 EOD FMT G $ END.
PR ON 2 $CHAR(26) FMT G $ END. !ctrl-Z eof mark
RELEASE 2.
TYPE "DEMDIF.DIF has been created on host and may be File".
TYPE "Translated into 1-2-3 format."
```

Fig. 13

Like our previous example (MAKPRN), this one suffers from intractably complicated FORMATS. Furthermore, there are two requirements imposed by DIF which our DMC does not really address:

1. The file must consist of a known number of "TUPLES" (we can consider them spreadsheet rows) and this number must be recorded in the file header;
2. Each "TUPLE" must be of equal length - i.e. each tuple must consist of the same number of "VECTORS" (we can consider them spreadsheet columns) and this number must be recorded in the file header.

Our example meets these requirements only because it was constructed knowing the correct counts in advance. Obviously this is not usually the case.

A more general way to handle requirement (1) would be to have the program keep track of the number of lines(TUPLES) printed, and to write this count to the DIF file header when done. This is in fact what the l022 DIF printing option does. The means of counting "TUPLES" is SYSPCROW, whose value is used for the count at RELEASE-time.

Requirement (2) is handled by assuming that each "TUPLE" will contain l00 cells ("VECTORS") unless the user says otherwise at INIT-time. The user specifies this via an optional NCOLS (Number of COLumns) clause in the INIT command. Given this NCOLS parameter, l022 ensures that each row is of equal width.

Given a large enough NCOLS, then, a program need not worry about uniform length of its PRINT commands. If the correct NCOLS value can be known in advance at INIT-time, however, it can be used to advantage by avoiding padding and therefore saving file space.

4.1 INIT Syntax For Printing DIF

The full INIT syntax for DIF is:

```
INIT DIF [COL c] [ROW r] NCOLS n chan filespec
```

defaults: c=A, r=1, NCOLS=l00, filespec extension= .DIF

4.2 DIF Example DMC

FIGURE 14 shows a DMC using the new INIT/PRINT features to produce a DIF file equivalent to the previous example. Note the use of "X" and "/" formats.

```
!This DMC extracts 1022 records and fields and produces
!a DIF file using V117B DIF printing features.
OPEN MOBDEM.DMS RO.
F SYSID BET 1 10. SORT LN FN.
INIT DIF 2 DEMDIF.
!INIT wrote the header.... no fuss, no muss.
!Note use of X and / fmts in next command
PRINT ON 2 "FIRST NAME" "LAST NAME" "# CHILDREN" "CITY" -
"STATE" "ZIP" FMT G 1X G / END.
!now print the data from the records
PR ON 2 FN LN NCH CITY STATE ZIP FMT 5G 1X G END.
RELEASE 2.
TYPE "DEMDIF.DIF has been created on host nd may be File" .
TYPE "Translated into 1-2-3 format.".
```

Fig. 14

"X" and "/" (as well as the COL and ROW options) are functionally identical to their usage in printing 1-2-3 files. However, they actually work by writing out "padding" (blank cells).

FIGURE 15 is the result of using the Lotus File Translate utility to derive a WKS file from the DIF file produced by the program of FIG. 14, and loading the WKS file into 1-2-3:

FIRST NAME	LAST NAME	# CHILDREN	CITY	STATE	ZIP
CHARLES	CARAGIANES	3	DEDHAM	NY	02138
RICH	GARLAND	4	BRISTOL	CT	02138
CHARLES	GOTT	2	BRISTOL	CT	22209
KATHY	HOUSMAN	0	AUGUSTA	GA	43220
MARK	JONES	0	DEDHAM	NY	60064
ROGER	LEVINSON	3	BRISTOL	CT	11729
LOUIS	MERZ	0	ROXBURY	TX	77056
OLGA	PONG	3	STONEHAM	MA	02238
ALFRED	SAVIO	1	ROXBURY	IN	46225
ALFRED	STEVENS	0	BRISTOL	CT	02238

Fig. 15

4.3 1022 --> DIF Data Transformations

DIF format recognizes only two datatypes: text and numeric. Numeric data is represented by ASCII digit strings. The rules for data transformation and formatting are quite simple:

1. The type of the 1022 item being printed determines the DIF datatype:

1022 type	DIF type
Integer	Numeric
Double Integer	Numeric
Real	Numeric
Date	Text
Text	Text

(Note the date --> text transformation. If a computational date is desired, \$INT(date-item) should be printed. This does not guarantee that the resultant number will be correct when read into the target spreadsheet or other program, which is likely to represent dates differently than 1022. The \$INT result may have to be offset by an amount that will cover the difference.)

2. Conventional 1022 formats are employed when printing to DIF files; there is no DIF equivalent to "L" format. The actual text or digit string that results is the same as that which would normally be produced under a given format spec. It is the programmer's responsibility to ensure that such a result is appropriate for the spreadsheet or other program which is to receive the DIF file. To mention a fairly obvious example, PRINTing an integer under "O" (octal) format would produce a numeric DIF item which would be interpreted as a string of decimal digits by any program that adheres to the DIF standard.



Using Mobius to Extend 1022 and 1032 Capabilities to Personal Computers

by
E. William Merriam, President
FEL Computing
PO Box 200
East Dover, VT 05341
U.S.A.

(802) 348-7171

Mobius is a micro/host integration package that extends the capabilities of Software House's 1022 or 1032 systems to personal computers. The personal computer user can now extract data from a large central data base and process it using the vast array of readily available personal computer software. This can all be accomplished without ever leaving the familiar environment of the microcomputer. In addition, host system capabilities, such as its mail system, are now directly available to the microcomputer user.

This paper shows how the 1022 system interacts with Mobius to provide a smooth interface between the host and micro computers. While the examples given are for 1022, the principles and most of the details are identical for 1032. How Mobius meets the varying needs of the microcomputer end user, host computer user, programmer, and information manager will also be addressed. More complete information and technical details about Mobius may be obtained by contacting the author.

1.0 MOBIUS, 1022, AND THE MICROCOMPUTER END USER

Mobius allows the microcomputer user to access host programs, data, and other resources (such as printers) in exactly the same way that the micro's own programs, data, and resources are accessed. Thus, the end user only needs to master one computing environment - that of the microcomputer. Mobius handles access to host resources completely transparently, so that the user can be totally unaware of where the programs and/or data actually reside. Therefore, the user is left to concentrate on the task to be accomplished, unencumbered by difficult and error-prone file transfer and communication tasks. This results in more efficient use of the person's time, both because no additional training is required and because the operations being performed are handled smoothly and easily.

1.1 An Example

An example will illustrate how easily the micro-to-host interaction becomes to the end user. Here, the host 1022 data base management system is used to extract information from a central data base, and then that information is loaded into a Lotus 1-2-3 spreadsheet on the micro. With Mobius, this task is performed completely on the user's microcomputer with the following sequence of commands:

```
(1)      A>1022
(2)      *USE MAKWKS
(3)      *QUIT
(4)      A>123
(5)      /File Retrieve D:DEMO3
```

Line (1) of this example shows the "A>" prompt displayed on the user's personal computer (IBM-PC, Rainbow, etc.). The user now wants to run the 1022 system which is written for and runs on the host machine, so he enters the "1022" command to the microcomputer's prompt. Notice that this is exactly how the user would start a program that was written for and runs on the microcomputer. As far as this user is concerned, he is simply running a program; he does not know or need to know where it actually resides.

Between lines (1) and (2) of the example, Mobius operates invisibly so as to make the host program access completely transparent to the user. First, Mobius starts the host 1022 program and then it automatically causes the microcomputer to operate as a VT-100 terminal. Thus, when 1022 outputs its "*" prompt, it appears on the screen just as would the prompt from any microcomputer program.

Lines (2) and (3) are commands which the user enters into the 1022 system. These commands can be as simple or elaborate as the application requires, and all features of the host 1022 system can be utilized. In this example, MAKWKS is a program that extracts data from a 1022 data base file and outputs a file in a format that can be read by the Lotus 1-2-3 spreadsheet program which runs on the user's microcomputer. The file itself is stored in a directory on the host computer, but the user need not be concerned about this. All the user in this example needs to know is that when the MAKWKS program is run, it produces a file called "DEMO3" on the microcomputer's "D" drive which is internal to his machine and which he can't actually see. In fact, later on, Mobius will perform the appropriate tasks, invisibly to the user, which cause this file to be retrieved when the "D" drive is referenced.

Between lines (3) and (4), Mobius again operates invisibly. First, it detects that the host 1022 program has terminated;

then it causes the microcomputer to operate as it normally does, instead of as a VT-100 terminal; and finally, it causes the micro's "A>" prompt to again appear.

Now, when the Lotus 1-2-3 program is started (line 4), all that the user needs to do is to retrieve the file D:DEMO3 that was created by the 1022 system, just as any other file would be retrieved with 1-2-3 (line 5). Again, Mobius operates invisibly to retrieve the file from the host system and to make it available to the 1-2-3 program.

1.2 An Even Simpler Example

The above example illustrates how Mobius operates to provide truly integrated micro/host interaction. The entire process can be even further simplified by using still other features of Mobius. For example, for users who do not know how to use 1022, but still have a need to access its data, Mobius provides a facility where lines (1) through (3) of the example can be combined into what appears to the user simply as a microcomputer program. If we call this program "GETWKS", then the following user commands to the micro perform the same function as the previous example:

```
(6)      A>GETWKS
(7)      A/123
(8)      /File Retrieve D:DEMO3
```

In this example, the user is freed from needing to know anything about the host 1022 system. This is particularly useful in the somewhat common situation where the user wishes the same type of updated data on a regular basis.

Since Mobius is completely integrated into the microcomputer's operating system, its "batch" facility can be used to even further simplify the action required by the user. For example, if lines (6) and (7) are combined into a batch file called "START123", then the entire process of accessing the host, starting the 1022 system, extracting data from the data base, outputting the extracted data into 1-2-3 file format, and loading that data into a 1-2-3 spreadsheet can be performed with only two microcomputer commands:

```
(9)      A>START123
(10)     /File Retrieve D:DEMO3
```

Note that in these examples, Mobius has worked completely invisibly and has not required the user to deviate from normal microcomputer procedures in any way.

1.3 Additional Versatility

The above is only one illustration of how Mobius allows end users to access host resources without needing to know any of the details of the host system. While the examples used the 1022 and 1-2-3 programs, they are equally valid for any host and/or micro program or combination of them. For instance, the host MAIL program can be run just as conveniently as 1022 was run in the example, thus providing the micro user with access to all of the features of the host mail facility as if that facility resided on the micro.

As another example, a microcomputer text editor, such as WordStar, can be used to edit files that have been created by a host program. In this case, Mobius allows the host file to be read directly into WordStar, eliminating the need to perform any complex file transfer tasks.

As can be seen by all of these examples, the integrated applications which Mobius makes available are virtually unlimited, since every host program can now be run as if it were on the micro, and every micro program can directly access host data and other resources. Mobius imposes no constraints on these whatsoever, thus eliminating user retraining and preserving current investments in software.

2.0 HOW MOBIUS SUPPORTS THE HOST USER WHO HAS A MICROCOMPUTER

While Mobius allows the microcomputer end user complete transparency when accessing a host machine, such transparency may not always be desired by a person who is familiar with the use of the host. Also, this type of person is most likely to be setting up applications for end users, and therefore needs a mechanism to accomplish this quickly and conveniently.

2.1 Switching Between the Micro and Host

Switching directly between the microcomputer and host environments can be accomplished in a variety of ways with Mobius. The way most familiar to most host users is simply to type the following microcomputer commands:

```
A>PUSH      (if the host is TOPS-20)
A>SPAWN     (if the host is VMS)
```

When this is done, Mobius invisibly starts a new host process and causes the microcomputer to operate as a VT-100 terminal. At this point, any host program or function can be performed, such as editing a file, reading mail, running a data base system, etc. When Mobius detects that the process is terminated (ie: the user

enters "POP" on TOPS-20, "LOGO" on VMS), it causes the microcomputer to operate as it normally does, instead of as a VT-100 terminal, and then to display the "A>" prompt again.

2.2 Configuring the Micro/Host Environment

Rather than requiring direct access to the host operating system functions as above, the user may wish instead to access that portion of the Mobius system itself which resides on the host machine. It is this portion of Mobius that contains an easy-to-use set of commands which allow the user to configure the Mobius environment, as was necessary for the 1022 example given above. To do this, the user simply enters a single keyboard character (initially defined as "CONTROL-A", but resettable by the user). Then, the current activity taking place on the microcomputer is instantly suspended (so that it can be resumed later), the host Mobius system is activated, and its "MOBIUS>>" prompt is displayed. At this point, Mobius is waiting for a command to be entered by the user.

The host Mobius commands provide a tremendous amount of convenience and capability for setting up applications as well as for performing useful host functions. In the 1022 example above, a file called "DEMO3" was written to a host directory, and that file was seen by the microcomputer to reside on its disk drive "D:". This relationship between host and micro resources is established using the Mobius "DEFINE" command. For instance, the command

```
MOBIUS>>DEFINE (micro device) D: (to be) Host (resource) *.*
```

tells Mobius that whenever the microcomputer's "D:" device is referenced (such as was done with Lotus 1-2-3 in the example), the files of the currently accessed directory (as specified by the "*.*") are to be accessed. Thus, as an additional example, the microcomputer command

```
A>DIR D:
```

will display all of the files on the user's currently accessed host directory. If the list of files specified to the DEFINE command had been "*.DOC,*.MEM", then only those files with "DOC" and "MEM" extensions would be listed. Similarly, if "SYS:" had been specified, then all of the files associated with that logical name would be displayed, no matter how many directories that represents.

The DEFINE command can also be used to specify that output normally destined for the microcomputer's printer will instead be routed to a host device. For example,

```
MOBIUS>>DEFINE (micro device) PRN: (to be) HOST  
          (resource) PRINTR.OUT
```

would route all microcomputer printer output to the host file "PRINTR.OUT". This output could have just as easily been routed to a host line printer or other device.

The "DEFINE" command is only one of about thirty commands that the host Mobius system provides. Some of the other commands replicate host system commands such as "COPY", "DELETE", "RENAME", etc., so that these functions can be performed easily and without leaving the Mobius system. Others allow for the tailoring of the Mobius environment for individual user's needs, such as changing the "CONTROL-A" character mentioned above, specifying the amount and type of information given when help is requested, and outputting specific application-oriented information. Still other commands allow setting the parameters of the communication channel or showing the status of the Mobius environment.

Once it has been determined how the Mobius environment is to be configured, all of the necessary host commands can be put into a data file. This file is then read when host Mobius is started and each command is executed, just as if it had been entered from the keyboard. Thus, the entire micro/host environment can be set up automatically and invisibly to the user.

3.0 MOBIUS AND THE PROGRAMMER

For most organizations, Mobius provides all necessary micro-to-host integration functions without requiring any special programming whatsoever. However, for those organizations which wish to create specialized distributed applications, Mobius simplifies the process by providing an Advanced Programmer's Interface (API). The API is designed to give programmers direct access to the Mobius features that are available to the user at the microcomputer keyboard. For example, the user activates the VT-100 terminal emulator by typing a special character. Similarly, a program can activate the terminal emulator by using a Mobius API "system call".

The Advanced Programmer's Interface appears to the programmer as an extension of the micro's operating system. As such, it gives the programmer access to several new system calls which are utilized in exactly the same way as normal system calls are utilized. Any programming language which can make calls to the microcomputer's operating system (which is virtually all of them) can call upon Mobius to perform its micro-to-host integration tasks. Thus, the API allows end-user organizations and OEMs to create sophisticated distributed applications without requiring systems programmers or communication specialists.

4.0 MOBIUS AND THE INFORMATION MANAGER

The previous sections have shown some features of Mobius as they related to particular types of host and/or personal computer users. To the Information Manager, though, Mobius is more than a set of technical features and capabilities. Rather, it is a single unified solution to the problems created by a diverse set of micro/host users, using a variety of programs and machine types. The inherent versatility of Mobius is illustrated in the previous section by the ease of use for the microcomputer end user, host system user, and programmer alike. Mobius provides each class of user with the same environment they are already used to, thus increasing their productivity and minimizing (even eliminating) the need for retraining. Each class of user is also provided with easy access to the rich set of features that are available to the other classes of users, should they wish to take advantage of them.

This versatility is complemented by close attention to the needs of managing host data and security. Mobius provides this through a combination of host file access mechanisms and special Host Mobius features. A key element of Mobius is that first-level access security is not controlled at the microcomputer, which is the most vulnerable part of a micro-to-host system, or even by Mobius itself; but rather it is controlled through the host operating system.

5.0 HOST-BASED ACCESS PROTECTION

The Host Mobius program operates as a normal user program running under the host operating system. Therefore, Mobius can provide the microcomputer user with no more file access than that user would have if accessing the host from a normal computer terminal. This design was chosen over a "server" or "privileged program" concept because it allows easy custom tailoring to individual users without introducing security problems.

Some of the advantages of this design are:

- (a) The host system manager needs to establish directory and file access privileges only once. There is no additional mechanism needed to provide protection for microcomputer users.
- (b) No passwords can be entered by the user when running a program from the microcomputer, nor can any passwords be accidentally displayed.
- (c) Because of (b), it is useless to enter passwords into data files stored on the microcomputer since they can not be functional from such files. Storing passwords in such files is one of the most common areas of security breach.

- (d) The microcomputer user has full access to those host files normally available to that user. No additional procedures must be learned to access them.
- (e) The host system manager remains in full control of the access and integrity of the host system files.

6.0 MOBIUS-BASED ACCESS PROTECTION

By design, Mobius can not allow access to host files beyond what is allowed by the host operating system. However, it can further restrict such access. For example, if the host system allows a user to read and write all files in a particular directory, Mobius can be set to allow reading only those files written by the user during the current Mobius session.

Mobius also provides the ability to mark sets of host files as "read only". This is accomplished with the Host Mobius "LOCK" command, which not only prevents writing to files that already exist, but also prevents new host files from being created.

It is also possible to prevent the user from accessing the host except through Mobius and/or to issue any Host Mobius commands. Thus, the micro-to-host environment can be set up so that the user will never be able to change it, but all required host resources will still be available to the microcomputer user.

7.0 INTEGRATING PCS AND HOSTS

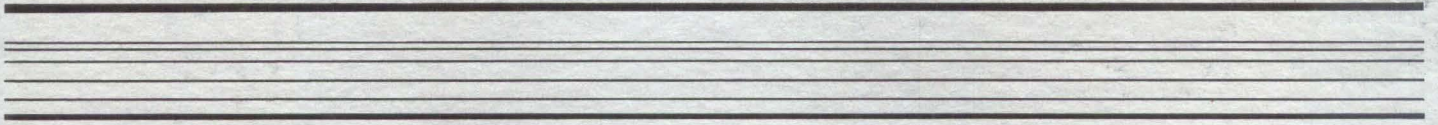
Mobius is a system which fully integrates personal computers with host machines. While traditional file transfer and terminal emulation capabilities are built into Mobius, these only scratch the surface of the tremendous versatility available to the user and/or system integrator.

The example illustrating the use of the 1022 data base system with the 1-2-3 spreadsheet shows that Mobius supplies direct access to an organization's data and facilities from a personal computer, while maintaining all of the flexibility that these machines offer the user. Also, by offloading tasks to the personal computer, host performance and user productivity is increased.

The versatility of Mobius is further enhanced by the wide variety of machines on which it is implemented, including VAX, DECsystem-10, and DECsystem-20 host computers and PC-DOS (IBM-PC and compatibles), MS-DOS, and CP/M microcomputers. This range of machines allows integration to take place not just between PCs and hosts, but between dissimilar hosts and microcomputers as well.

All of this adds up to an unusually flexible system for the 1022/1032 user. First, Mobius allows the capabilities of these systems to be immediately extended to the microcomputer user. Then, extracted host data can be used in 1-2-3, dBase, and other microcomputer programs. As the user's needs grow, additional host systems can also be extended to the micro. What may initially be viewed as an adjunct to the 1022 or 1032 system, in fact provides general-purpose capabilities that can be used to integrate virtually any micro/host application.





1

